

# **Multi-Angle Valve Seat Machining: Experimental Analysis and Numerical Modelling**

By

**James R. Fletcher**

A Doctorial Thesis

Submitted in partial fulfilment of the requirements for the award of  
Degree of Doctor of Philosophy of Loughborough University

September 2018

© James Fletcher 2018

# Abstract

---

Modern automotive manufacturers operate in highly competitive markets, heavily influenced by Government regulation and ever more environmentally conscious consumers. Modern high-temperature, high-pressure engines that use high hardness multi-angle valve seats are an attractive environmental option, but one that manufacturers find requires more advanced materials and tighter geometric tolerances to maintain engine performance.

Tool manufacturers meet these increasingly tougher demands by using, higher hardness cutting materials such as polycrystalline cubic boron nitride (pcBN), that on paper, promise to wear at a lower rate, require less coolant and deliver tighter tolerances than their carbide counterparts.

The low brittle fracture toughness of pcBN makes tools that use it vulnerable to minute chipping. A review of literature for this work pointed to no clear answer to this problem, although suggestions range from manufacturing defects, dynamic and flexibility problems with the production line machinery and fixtures, and radial imbalances in the cutting loads.

This work set about experimentally investigating those potential explanations, coming to the conclusion that the high radial imbalance of the cutting loads is responsible for pcBN cutting insert failure during multi-angle valve seat machining, and that by simply relocating the cutting inserts around the multi angle cutting tool, the imbalance can be reduced, thus extending the life of the cutting inserts.

It is not always easy to predict the imbalance due to the multiple flexibilities in the system, and simulating such a system in 3D with all its associated cutting phenomena such as friction, thermal expansion, chip flow and shearing, would call upon extraordinary computational power and extremely precise experimental inputs to reduce cumulative error.

This thesis proves that such a 3D simulation can be made, that runs in exceptionally short durations compared to traditional methods, by making a number of simplifications.

MSC Marc was used to host the simulation, with a parametric script written in Python responsible for generating the model geometry and cutter layout. A Fortran program was developed that is called upon by Marc to calculate the required cutting load outputs and generate new workpiece meshes as material is removed.

# Conferences

---

- The 8<sup>th</sup> International Symposium on Mechanics of Materials and Structures – 31 May to 3 June, 2015, Augustow, Poland
- The 12<sup>th</sup> World Congress on Computation Mechanics (WCCM XII) and The 6<sup>th</sup> Asia-Pacific Congress on Computational Mechanics (APCOM VI) – 24 to 29 July 2016, Seoul, South Korea
- The 7th International Conference on Computational Methods (ICCM2016) – 1 to 4 August 2016, University of California Berkeley, California, USA
- Powertrain Modelling and Control, Testing, Mapping and Calibration (PMC2016) – 7 to 9 September 2016, Loughborough University, United Kingdom
- 16<sup>th</sup> CIRP Conference on Modelling of Machining Operations – 15 to 16 June 2017, Cluny, Burgundy, France

# Acknowledgements

---

This project was supported by the High Speed Sustainable Manufacturing Institute and Ford Motor Company.

I would like to express my gratitude to my PhD supervisor, Dr Emrah Demirci, who has given me unwavering support, both academically and personally despite many life challenges throughout my time at Loughborough University.

I would also like to acknowledge the support of my family, without which, this submission would not have been possible. In particular I would like to thank my mother, Janet Fletcher, who has always been on hand to motivate and guide me, often turning me from the brink of giving up.

Also, my wife, Stephanie Fletcher, who despite living in another country has bravely faced the challenges of living apart, putting up with my stress and endless lack of time, all the while having the energy to support me emotionally throughout this process.

Scores of staff at Loughborough University deserve my thanks and gratitude, in particular I would like to thank Rob Hunter and Michael Porter in the Wolfson School workshop for giving their time and guidance during some experimental phases of this work, Jagpal Singh for his excellent metrology advice and willingness to donate lab time to this work and finally Peter Bracey in the Electronics Workshop for lending me measurement hardware for various experiments.

From Ford, I would like to thank Matthew Denham, whose interest in my work led to the amazing opportunity to visit Ford's plant in Craiova to perform a range of key experiments.

Thank you all.

This work is dedicated to my grandfather, Gordon Fletcher, who inspired me to pursue a life in science and technology.



Loughborough  
University





# Table of Contents

---

<b>Abstract .....</b>	<b>i</b>
<b>Conferences .....</b>	<b>ii</b>
<b>Acknowledgements.....</b>	<b>iii</b>
<b>Table of Contents .....</b>	<b>iv</b>
<b>Chapter One – Introduction.....</b>	<b>1</b>
1.1 Case Study and Motivation.....	1
1.2 Background .....	5
1.3 Research Objectives .....	8
1.4 Scope .....	8
1.5 Thesis Outline .....	9
<b>Chapter Two – Mechanics of Machining .....</b>	<b>11</b>
2.1 Process Zone.....	11
2.2 Materials.....	13
2.2.1 AR20 Copper Infiltrated High Speed Steel.....	13
2.2.2 Cubic Boron Nitride .....	15
2.2.3 Tungsten Carbide.....	17
2.3 Cutting Fluids .....	17
2.4 Chip.....	18
2.5 Cutting speed and feed rate effects .....	19
2.6 Heat in Machining.....	19
2.7 Wear Mechanisms .....	20
2.8 Cutting Tool Failure .....	21
2.8.1 Stress .....	21
2.8.2 Thermal Effects.....	22
2.8.3 Vibration .....	22
2.9 Chapter Summary.....	23
<b>Chapter Three – Non-Linear Finite Element Analysis .....</b>	<b>24</b>
3.1 Theory of Non-Linear Finite Element Analysis .....	24
3.1.1 Variational Methods .....	26
3.1.2 Large Displacement and Strain Behaviour (Lagrangian Methods) .....	27
3.1.3 Contact and Friction .....	28
3.1.4 Convergence.....	29
3.2 Basic Procedure of FEA in Solid Mechanics .....	30

3.3	Mesh Refinement .....	31
3.4	Remeshing .....	32
3.5	Model Verification and Validation.....	33
3.6	Modelling Software Selection .....	34
3.7	Chapter Summary.....	36
<b>Chapter Four – Techniques for Characterising and Modelling Cutting Systems .....</b>		<b>37</b>
4.1	Mechanical Characterisation of the System.....	37
4.1.1	Modulus of Elasticity and Poisson’s Ratio .....	37
4.1.2	Vibrational Properties .....	37
4.1.3	Plastic Flow Curve Characterisation .....	38
4.1.4	Friction.....	38
4.1.5	Temperature.....	40
4.2	Modelling Cutting Systems.....	42
4.2.1	Johnson-Cook Constitutive Model .....	42
4.2.2	Modelling Damage.....	43
4.2.3	Modelling Vibration.....	44
4.3	Numerical Model .....	45
4.4	Chapter Summary.....	47
<b>Chapter Five – Experimental Analysis of pcBN Cutting Inserts .....</b>		<b>48</b>
5.1	Schedule of Specimens.....	50
5.2	Visual Inspection using Optical Microscopy .....	52
5.2.1	Measurement of Corner Radius .....	52
5.2.2	Inspection of Large Chip on Insert 14.....	53
5.2.3	Inspection of Small Chip on Insert 25.....	55
5.3	X-Ray Computed Tomography.....	57
5.4	Scanning Electron Microscopy (SEM) .....	59
5.4.1	Insert 14 – Chipped edge.....	59
5.4.2	Insert 14 – Chemical Analysis of Speckle and Black Patterns.....	62
5.4.3	Insert 16 – Tungsten Carbide Chemical Analysis.....	64
5.4.4	Insert 16 – Wear Profile.....	64
5.4.5	Insert 2 – Rake face deposits .....	65
5.4.6	Insert 2 – Crack .....	68
5.4.7	Insert 2 – Crystal Pattern .....	71
5.5	Chapter Summary.....	72
<b>Chapter Six – Dynamic Analysis of the Valve Seat Cutting Operation .....</b>		<b>73</b>
6.1	Introduction .....	73
6.2	The Machining Process.....	74
6.3	The Effect of ZPL Bolt Torque on Seat Positional Accuracy.....	78

6.4	Stiffness Analysis .....	81
6.4.1	ZPL Preload .....	81
6.4.2	Geometry and Mesh .....	84
6.4.3	Boundary Conditions .....	86
6.4.4	Material and Physical Properties .....	87
6.4.5	Results .....	87
6.5	Cylinder Head Resonance Analysis .....	94
6.5.1	Experimental Design .....	94
6.5.2	Calibration .....	100
6.5.3	Results .....	100
6.6	Chapter Summary .....	103
<b>Chapter Seven – Experimental Characterisation of Cutting Force.....</b>		<b>104</b>
7.1	Introduction .....	104
7.2	Theory .....	105
7.3	Experimental Methodology .....	111
7.3.1	Parameter selection .....	114
7.4	Data Processing .....	114
7.4.1	Signal Conditioning .....	115
7.4.2	Time Offset Approximation and Refinement .....	115
7.4.3	Specific Force Calculation .....	115
7.5	Results and Discussion .....	117
7.5.1	Radial Force Analysis .....	117
7.5.2	Specific Force Data .....	118
7.5.3	Thermal Analysis .....	124
7.5.4	Surface Analysis .....	128
7.6	Chapter Summary .....	131
<b>Chapter Eight – Development of Theoretical Finite Element Model .....</b>		<b>132</b>
8.1	Justification .....	133
8.2	Objectives .....	135
8.3	Strategy .....	135
8.4	Model Design and Performance Constraints .....	138
8.4.1	Assumptions .....	138
8.4.2	Coordinate System .....	138
8.4.3	Geometry .....	139
8.4.4	Boundary Conditions .....	143
8.4.5	Contact .....	144
8.4.6	Material (Novofer AR20) .....	144
8.4.7	Additional Output .....	144

8.4.8	Initial Parameter Configuration .....	145
8.4.9	Parametric Compatibility .....	146
8.5	Marc / Python Interface .....	147
8.6	Python Model Generator Script .....	149
8.6.1	Setup .....	150
8.6.2	Valve Seat .....	150
8.6.3	Spindle .....	154
8.6.4	Cutters .....	155
8.6.5	Contact Table .....	159
8.6.6	Remeshing Instruction .....	159
8.6.7	Loadcase Configuration .....	160
8.6.8	Job Configuration .....	160
8.7	Chapter Summary .....	161
<b>Chapter Nine – Design and Development of Fortran Program .....</b>		<b>162</b>
9.1	Introduction .....	162
9.2	Definitions .....	163
9.2.1	Programming Terms .....	163
9.2.2	Procedure Diagrams .....	164
9.2.3	Pre-Processor Definitions .....	164
9.2.4	Mesh Definitions .....	165
9.2.5	Siding .....	168
9.3	Development Environment .....	169
9.3.1	Development Software .....	169
9.3.2	The Relationship Between the Fortran Program and Marc .....	170
9.3.3	Debugging Tools and Methods .....	171
9.4	Program Theory .....	174
9.4.1	Types and Objects .....	174
9.4.2	User Subroutine Call Order .....	177
9.4.3	Configuration File .....	178
9.5	Procedure .....	179
9.6	Runtime Setup .....	181
9.7	SplitMesh .....	182
9.7.1	Intersection Detection .....	183
9.7.2	Hull Splitting .....	185
9.7.3	2D Meshing – Cut Face .....	188
9.7.4	Significance Check .....	189
9.7.5	2D Meshing – Split Hull Facets .....	189
9.7.6	Simplification I – Sweep .....	192
9.7.7	Removal of Unwanted Hull Side .....	193

9.7.8	Simplification II – General Complexity Reduction .....	195
9.7.9	Mesh Optimisation .....	197
9.7.10	Facet Siding.....	200
9.7.11	3D Volume Meshing .....	201
9.7.12	Force Recovery and Allocation .....	202
9.8	Low-Level Functions and Subroutines .....	205
9.8.1	Volume of a Tetrahedron .....	205
9.8.2	Ray Construction .....	206
9.8.3	NURBS Surface Formula (NURBSSurface%S, CDBR) .....	206
9.8.4	Reverse NURBS Surface Formula (NURBSSurface%GetUV) .....	207
9.8.5	Circumscribed Triangle in R3 (Circumcircle) .....	209
9.8.6	Circumscribed Tetrahedron (Circumsphere).....	210
9.8.7	Ray-Ray Intersection (Ray%RayIntersect).....	211
9.8.8	Ray-Triangle Intersection (Ray%TriIntersect).....	213
9.8.9	Triangle-Triangle Intersection (TriTriIntersect) .....	215
9.8.10	Ray-NURBS Intersection (NURBSSurface%RayIntersect) .....	217
9.8.11	Triangle-NURBS Intersection (NURBSSurface%TriIntersect).....	220
9.9	High-Level Functions and Subroutines .....	221
9.9.1	Two-Dimensional Meshing (Giftwrap) .....	221
9.9.2	Three-Dimensional Meshing (TetMesh) .....	224
9.10	Chapter Summary.....	227
<b>Chapter Ten – Results and Discussion.....</b>		<b>230</b>
10.1	Model Performance & Validation.....	230
10.1.1	Speed .....	231
10.1.2	Validation of Loads .....	232
10.1.3	Validation of Geometry .....	234
10.1.4	Progression of Mesh Complexity .....	241
10.1.5	Validation of Volume Calculation .....	243
10.1.6	Cutter Face Duty Cycle Map .....	244
10.2	Multiple Cutters.....	246
10.2.1	Configuration .....	246
10.2.2	Validation of Balanced Configuration.....	249
10.2.3	Radial Imbalance Reduction Study .....	250
10.2.4	Simulation Stability.....	254
10.3	Remeshing Algorithm .....	255
10.4	Chapter Summary.....	256
<b>Chapter Eleven – Conclusions.....</b>		<b>257</b>
11.1	Summary of Results .....	257

11.2	Future Research.....	257
11.3	Conclusions.....	259
11.3.1	Objective One .....	259
11.3.2	Objective Two .....	259
11.3.3	Objective Three .....	259
11.3.4	Objective Four .....	260
11.4	Novel Contributions.....	261
<b>References .....</b>		<b>262</b>
<b>Appendix A – MarcTools .....</b>		<b>270</b>
<b>Appendix B – Fortran Source Code Procedure Headers.....</b>		<b>275</b>

# Chapter One – Introduction

---

Metal properties, such as strength and durability, make the use of metals in everyday life commonplace. It is not surprising therefore that basic metals and fabricated metal products make up more than half of the UK's manufacturing consumption (Department for Business Innovation & Skills, 2010), the vast majority of which require further processing in the form of metalworking. Metalworking is the process of shaping metals to a given specification, through processes such as casting, forging, extrusion, welding and machining. These techniques, in particular machining, have seen considerable investment in terms of research and industry experience over hundreds of years throughout many countries.

Traditional machining, in its simplest form, is the removal of material by cutting with another stronger material. The ability to machine metals is core to the automotive industry. Competition between automotive manufacturers is fierce, especially since there is considerable pressure to continually improve the quality of products, keep costs down, meet legislative demands and retain or grow market share. Newer technologies such as high temperature, high pressure engines that boast superior power to weight ratios and better emissions performance than previous generation engines necessitate higher hardness engine components, in turn calling for state of the art cutting materials to increase the reliability and quality of finished parts, and to reduce tooling cost.

Polycrystalline cubic boron nitride (pcBN) is a high hardness cutting material that has a great deal of potential in metal working, particularly in multi-angle valve seat machining. The physical phenomena governing the behaviour of materials such as pcBN is still comparatively unexplored to that of other popular tool materials such as high speed steel (HSS) and tungsten carbide. This work aims to broaden our understanding of multi-angle valve seat machining with pcBN so that the full potential of pcBN can be realised.

## 1.1 Case Study and Motivation

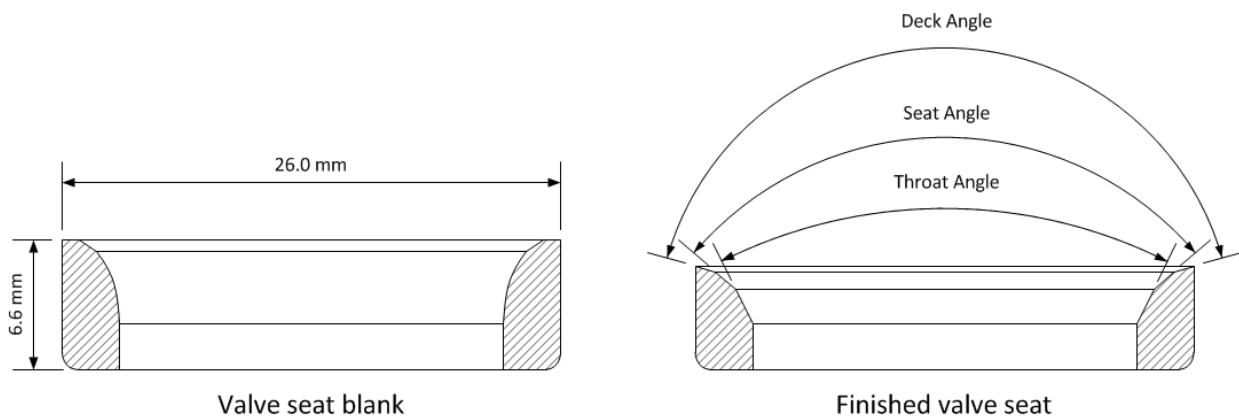
The motivation for this work stems from a production line phenomena observed by Ford Motor company during multi-angle valve seat machining on the Ford Sigma and Fox internal combustion engine cylinder heads. Ford reported that pcBN cutting inserts are prone to failure at seemingly random times throughout the process and typically long before the scheduled tool change interval. The cutting surfaces of the inserts are made using polycrystalline cubic boron nitride (pcBN). The cutting system typically uses three inserts spaced 120° apart about the axis of rotation, each of which finishes a different angle on the valve seat. The failure mode is typically minute chipping on the cutting edge, which causes rings of raised material to be left on the valve seat surface. Those raised surfaces leave the finished part out-of-tolerance and will often cause the seats to fail a leak test, necessitating scrapping of the cylinder head. It can sometimes be the case that up to 200 cylinder heads progress from the valve seat finishing step, before quality inspection identifies a failed valve seat. These heads cannot be reworked and must be scrapped at considerable cost to the company.

Although the tool life and performance of tungsten carbide cutting inserts is much more predictable, pcBN inserts boast a much a higher potential life and are successfully used by Ford in other processes such as cylinder boring. If more knowledge can be developed about pcBN inserts and the dynamics of cutting three-angled valve seats, it is hoped that the superior material properties of pcBN over tungsten carbide can be harnessed to increase productivity and reduce machining costs.

Valve seats are an essential component of modern internal combustion engines. Their use allows manufacturers to build cylinder heads from lightweight aluminium whilst still providing an extremely hard surface against which the valves can seal. The seats are pressed into the cylinder head and face into the cylinder. In this capacity they perform perhaps one of the most demanding roles of any component in the engine. They are expected to hold a very tight geometric tolerance for the entire life of the vehicle under high combustion pressures and repetitive impact from the valves as they open and close.

Ford's material of choice for this application is a sintered high speed steel. The pores left by the sintering process are filled with copper, leaving an extremely hard and durable material with excellent resistance to thermal deformation and outstanding sealing performance.

'Three-angle' refers to the geometry of the contact surfaces between the valve and the valve seat. Three surfaces of gradually decreasing angle, similar to that shown in the cross-section in figure 1-1, are used so that the valve and valve seat, seal on three independent surfaces, thus increasing the effectiveness of the seal.



*Figure 1-1 - Valve seat cross-section*

The sintering process produces rings of the approximate finished valve seat shape. Figure 1-2 shows valve seat blanks for the exhaust (left) and air intake (right) as they are delivered to the production line.

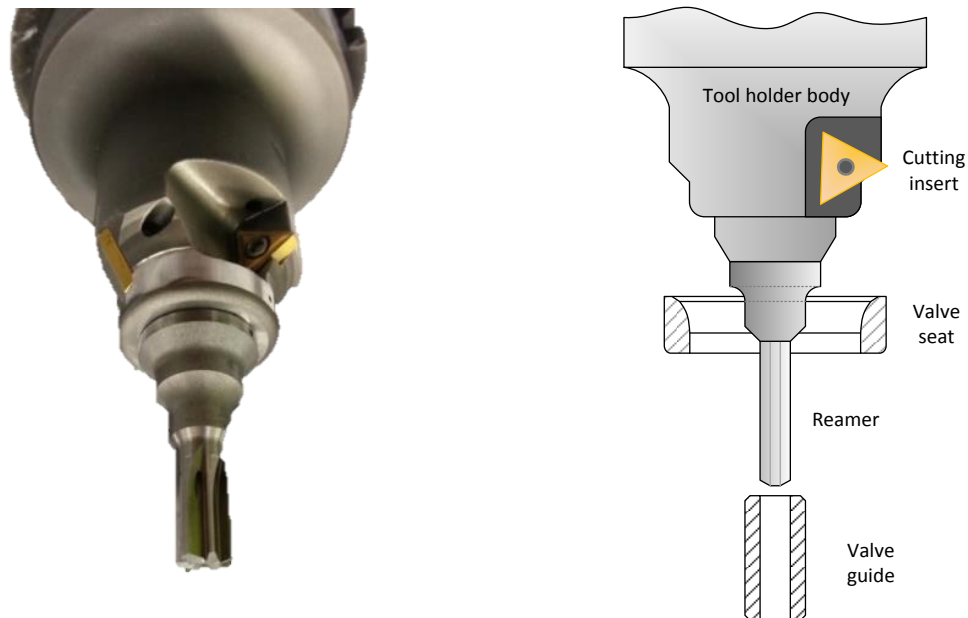


*Figure 1-2 - Valve seat blanks(scale divisions in mm)*



Valve guides and seats are pressed into the cylinder head using hydraulic rams. This interference fit is designed to keep the seats secured for the duration of their life. Some processes go a step further by cryogenically cooling oversized seats before insertion so as to allow significantly tighter interference fits.

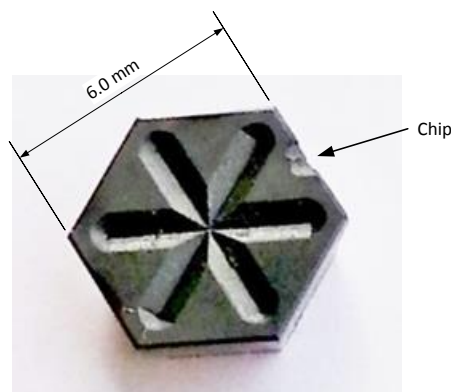
Once the valve seats are securely pressed in, the cylinder head is brought to a CNC machine where the finishing cuts are made. Figure 1-3 (left) shows a photograph of a valve seat cutting tool and (right) a simplified cross-section diagram showing the geometric relationship between the valve seat and valve guide during the cutting operation.



*Figure 1-3 – Valve seat cutting tool component layout*

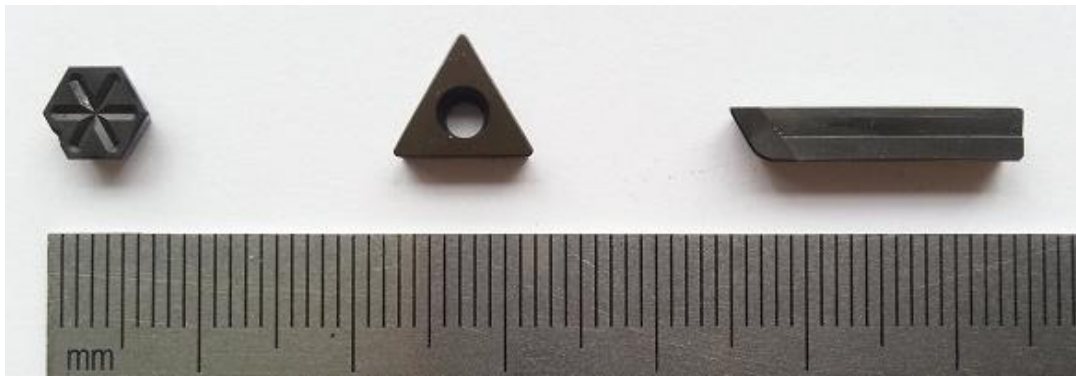
In some cases, the finishing cuts for all three angles of the valve seat as well as the valve guide are performed by a single tool. In other cases, the valve guide and seat are finished in separate steps. The tool has three cutting inserts, each of which corresponds to a different angle on the seat (these are the components with a gold appearance in the above figure).

The failure mode for the pcBN inserts is chipping. The scale of chipping ranges from clearly visible, to difficult to detect without optical inspection tools. Figure 1-4 shows one of the larger chips observed.



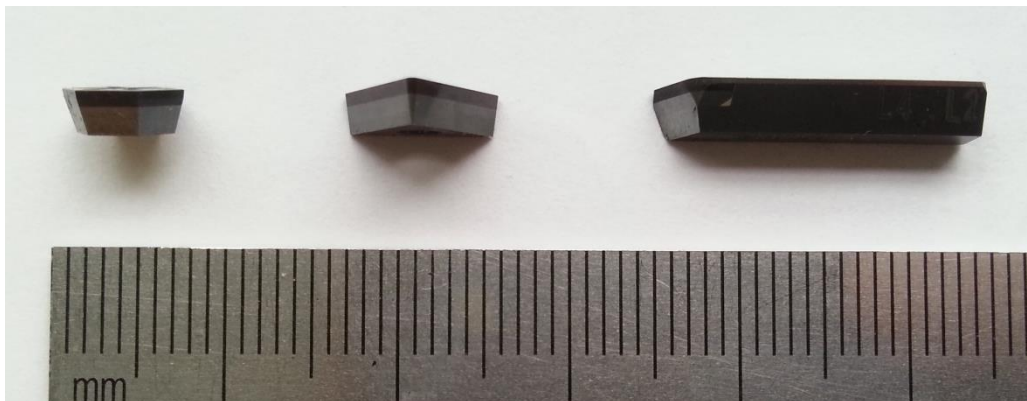
*Figure 1-4 – Chipped pcBN insert (parallel edge to edge distance is 6 mm)*

All of the cutting tools that Ford use for the operation are manufactured and supplied by MAPAL. Figure 1-5 shows images of the various pcBN cutting tool inserts used by Ford to cut valve seats.



*Figure 1-5 – pcBN cutting tool inserts*

The pcBN cutting surfaces are backed by a tungsten carbide reinforcement. Figure 1-6 shows the pcBN layer (darker material) cemented to a tungsten carbide substrate (grey material). This layer is significantly less brittle than the pcBN layer and thus provides additional strength to support the rake load during cutting.



*Figure 1-6 – pcBN cutting tool inserts (side view)*

## 1.2 Background

Ford is not the first engine manufacturer to experience issues during multi-angle valve seat machining with polycrystalline cubic boron nitride (pcBN) tools. Others have previously attempted to understand and improve the reliability and performance of valve seat machining processes using pcBN tools. Three significant studies were identified:

***Evaluation of the wear mechanisms and surface parameters when machining internal combustion engine valve seats using PCBN tools, Rocha et al., 2004.***

A collaborative, experimental case study was performed in conjunction with Fiat-GM in Brazil which specifically looked at the performance of hardened steel valve seat cutting, with pcBN. The valve seats studied were of a similar chemical composition to those used by Ford. However the paper does not clarify whether or not multiple pcBN tools are engaged at the same time (as in the Ford configuration) or how many angles are cut into the valve seat.

The study identified the importance of seal quality between the valves and their seats for ensuring good engine performance. It was found that the valve seat surface quality, and thus the cutting process, was the most influential factor dictating seal quality. Their objective therefore was to show the influence of cutting speed, feed rate and depth of cut on pcBN tool life and wear rate.

Experimental results were gathered using production line machines under normal operating conditions. Tools were periodically removed from tool holders, measured and replaced in order to monitor wear at set intervals. Care and attention was applied to ensure the entire machine, including coolant was at equilibrium temperature so as not to distort the results. Vibrational measurements were also taken using an accelerometer in order to characterise the dynamic instability of the system.

The study considered a wide variety of wear mechanisms, including chemical and attrition wear as a result of 'stick-slip' contact between the tool and workpiece. It concluded that the discontinuous chip flow, tool chipping and micro chipping observed was due to excessive vibration in conjunction with the low fracture toughness of pcBN. Other conclusions of the study were that:

- cutting speed does not affect cutting force;
- cutting speed had the largest influence on temperature;
- the vibration response of the system to changes in cutting velocity was highly non-linear, with small increases radically changing the tribological characteristics between the workpiece and tool. It was found that in some cases, a cutting velocity which yielded 30 parts before failure, could be modified by no more than 15% in order to yield 500 parts; and
- an increase in depth of cut increases the wear rate.

The study makes a series of predictions as to the performance of pcBN tools under certain conditions. However, the results are highly dependent on the particular setup and it remains unclear whether or not the same results would be observed using different equipment and workpiece materials. The study does not present a numerical model or finite element study which can be used to apply the findings to new problems and geometries.

This study is very important in the context of this work, since it proves a link between vibration, low fracture toughness of pcBN and chipping failure. In Ford's case, a number of factors can cause vibration such as flexibilities within the system and imbalanced cutting loads.

**Blade Geometry Effects on the Boring of Valve Seats of Internal Combustion Engines, Lacerda and Siqueira, 2012.**

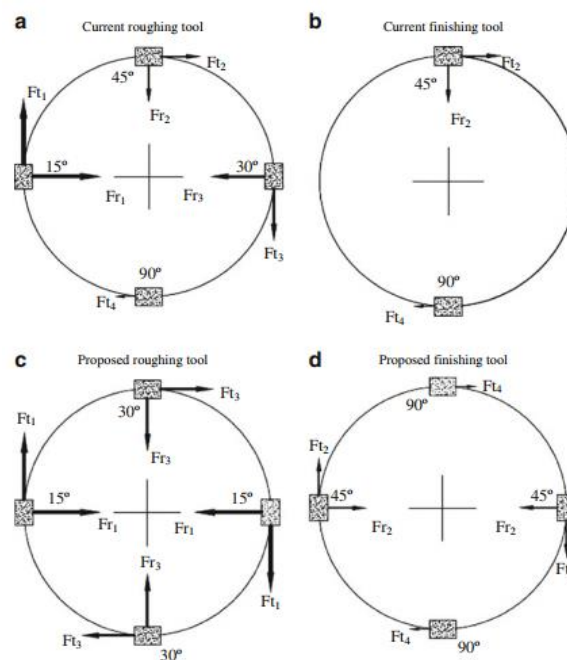
This work examined a particular valve seat cutting process which was yielding seats with a poor surface finish and was susceptible to tool failures.

The study also identified vibration as the cause, but took this a step further to identify what was causing the vibration. The possible causes they investigated were:

- inhomogeneous valve seat material in the form of hardness and porosity variations around the ring,
- poor choice of pcBN material grade;
- poorly optimised cutting parameters;
- a dynamic stability problem within the tool, workpiece and fixture geometry leading to regenerative chatter; and
- suboptimal cutting edge geometry.

The work ruled out some of these hypotheses and suggested small tweaks for others. One hypothesis however stood out and required a substantial and very significant change to the process. That factor was the layout of the cutting inserts themselves. It was suggested that because each cutter cuts a different angle on the seat, the radial forces are different causing a dynamic imbalance to arise once machining has started, in some cases exceeding 400 N of imbalance.

The research proposed a new tool holder design in which the individual cutting inserts are balanced by opposing inserts of the same angle as shown in figure 1-7 taken from their work.



*Figure 1-7 – Lacerda and Siqueira, 2012, current and proposed tool configurations*

Although they were not able to test this modification, they concluded that this imbalance was the primary factor which, in combination with the very poor fracture toughness of pcBN, led to tool failure.

The findings presented in their study bear a striking resemblance to the issues faced by Ford. Space constraints prevented the authors from proposing a tool in which all four cutters in the original roughing

tool are opposed by an additional four. They instead proposed separating the cutters onto two separate tools with each cutter being perfectly opposed. However, given the extremely large dynamic imbalance of 400 N, it may be possible to achieve stability by adding a single additional insert and changing the layout of the remaining inserts. The cost of designing and producing new tooling is extremely high; simulation could be undertaken to assist in the design, testing and optimisation of the reconfiguration of cutters in the way proposed, whilst keeping costs to a minimum.

***Wear Mechanism of CBN Inserts during Machining of Bimetal Aluminium-grey Cast Iron Engine Block, Malakizadi, Sadik and Nyborg, 2013.***

This work presented a 3D finite element model of the specific wear interaction between pcBN inserts and Grey-Cast Iron (GCI) cylinder liners within aluminium engine blocks. Individual pcBN inserts were used to machine both valve seat and parent metal materials alternately. This was significant as it demonstrated the performance of inserts when repeatedly exposed to the very different cutting conditions (different forces and temperatures) of soft non-ferrous aluminium and hard abrasive GCI. In the case study from Ford, it was assumed that the pcBN insert only ever cuts the hardened steel valve seat, although preliminary visual inspection of a cylinder head supplied by Ford suggested that occasionally the pcBN does in fact touch the aluminium body of the cylinder head and, therefore, the influence of bimetallic machining should not be ignored.

The study found through inspection using Scanning Electron Microscopy (SEM) that the propagation of comb cracks was accelerated by the alternating thermal and mechanical stresses mentioned previously. It was suggested that these comb cracks were responsible for the accelerated rate of chipping of the pcBN insert.

The study went on to develop a 2D non-linear finite element model using the commercially available DEFORM 2D package. The Johnson-Cook model was used to model plastic behaviour. The results of the 2D analysis showed good agreement with experimental results. A 3D model was also developed using the DEFORM 3D commercial software package. Using friction constants derived through validation of the 2D model, the researchers simulated 5° of cutting. However it was acknowledged that access to computational processing power was a limiting factor.

The study managed to successfully produce a 3D model of both aluminium cutting and CGI cutting. However, the 3D model was not verified using experimental results and instead relied upon parameters refined during the design of a 2D model. Neither the 2D nor 3D model implemented a damage model for the pcBN insert. Perhaps because in each simulation only a tiny proportion of the machining operation was simulated and so there was little opportunity for damage accumulation to be substantively expressed within the model. Instead, the study used the simulations to predict peak stress and temperature points at which chipping due to comb cracks would occur.

The literature review offered in the study conducted in 2013 suggested that there have been few successful attempts to model valve seat cutting with pcBN tools in 3D.

In summary, the three studies presented here all identify issues when machining with pcBN cutting inserts. In each case there is a role to play for simulation, however access to suitable methods and computational processing power appears to have been a limiting factor in all cases.

### 1.3 Research Objectives

The objectives of this work are to:

1. investigate Ford's cylinder head and fixture geometry and determine whether or not it undergoes resonance at typical valve seat cutting feed rates and speeds;
2. design and execute an experiment aimed at capturing specific feed and rake forces for the valve seat cutting operation, using a range of feed rates and spindle speeds for both dry and minimum quantity lubricant (MQL) conditions;
3. justify, design and develop a substantial body of code capable of calculating cutting forces for a sub-segment of the valve seat cutting operation at typical feed rates and speeds; and
4. test the simulation code by using it to calculate the sub-segment cutting load of a single cutter up to and beyond the typical cutting depth.

### 1.4 Scope

The scope of this project is limited to studying the failure mechanisms affecting the pcBN cutting insert samples provided by Ford. A narrow range of processes will be considered, specifically those used to finish valve seats at the Ford plant in Bridgend, Wales UK and the Ford plant in Craiova, Romania for the Sigma and Fox engines respectively.

These operations are fundamentally the same, involving three-bladed cutters, although there are variations within the fixture design, feeds and speeds. The relative pitch between the blades and thus the cone angles left on the seats is tightly controlled by Ford's design.

With regards to materials, although several cutter materials can be used for valve seat cutting, for example, tungsten carbide, this work will focus purely on pcBN. Although the reliability of tungsten carbide tools is generally very good, pcBN has a superior wear rate than that of tungsten carbide and is therefore theoretically able to hold the tight relative cone angles for longer durations than tungsten carbide. The study of pcBN based cutting systems provides a greater opportunity to contribute to new knowledge.

For the valve seat material and geometry, this work focused purely on the specific sintered AR20 high speed steel composition and blank part geometry used by Ford. Only the exhaust seat will be studied, but the tools developed will be suitably parametric, such that they can be readily applied to other geometries including the intake seat.

## 1.5 Thesis Outline

The first three chapters following this introduction present a literature survey. The literature survey is divided according to three specific areas:

- **Chapter Two – Mechanics of Machining:** Presents an overview of the current understanding of metal cutting. The chapter also provides a review of theoretical and experimental methods used to characterise various metal cutting phenomena. This chapter also addresses damage specifically relating to pcBN cutting tools.
- **Chapter Three – Non-linear Finite Element Analysis:** Presents the fundamental concepts of non-linear finite element analysis and its application in this project. The chapter also introduces the MSC Marc Finite Element Solver used in this project.
- **Chapter Four – Numerical Modelling of Machining:** Presents an overview of the knowledge that emerged at the juncture of the first two chapters, specifically how metal cutting physics is implemented in finite element simulations. This chapter highlights the complexity of modelling various machining physics.

As stated in the background for this project, significant pieces of previous work offered a plausible explanation for the premature tool failure observed by Ford. However, following the literature review for this work, many other factors were identified that could result in damage and mechanical failure of pcBN cutting inserts and these must be investigated first.

- **Chapter Five – Experimental Analysis of pcBN Cutting Inserts:** This is an experimental chapter which provides an analysis of several failed cutting inserts using a range of techniques including optical microscopy, X-Ray computed tomography and Scanning Electron Microscopy (SEM) to look for the presence of failure mechanisms and manufacturing defects identified in the literature, such as comb cracking, chemical diffusion, fatigue failure and delamination between the two layers of the insert and voids within the pcBN substrate itself. This chapter concludes that although the presence of various damage phenomena can be detected, the root observation of wholly unpredictable failure cannot be explained by these discoveries alone.
- **Chapter Six – Dynamic Analysis of Real Valve Seat Cutting Operations:** Another experimental chapter which studies the cutting process cycle in particular to discover whether or not chipping can be attributed to excessive vibrational amplitude brought about by resonance in the cylinder head and fixture structures. In this experiment, evidence of resonance was found in some cases.

The knowledge gathered during the literature survey for this project and the two experimental investigations provided a strong argument to suggest that the underlying causes for seemingly random premature of failure of cutting inserts was due to vibration in conjunction with the low fracture toughness of pcBN, caused by the dynamic imbalance that arises during cutting due to the different radial loads on each cutter.

**Chapter Seven – Experimental Characterisation of Cutting Forces:** Presents the design and application of an experiment aimed at determining which cutting parameters affect tool force when cutting AR20 valve seats with pcBN tools. This chapter also presents experimental results and a cutting force prediction model suitable for input into the numerical model developed in the final part of this thesis.

**Chapter Eight – Development of the Numerical Model:** Presents the design of a three-dimensional numerical model capable of simulating valve seat cutting. Due to the high complexity, iteration count and number of elements, this chapter also presents the numerous simplifications and optimisations required to

make the simulation possible within a reasonable time frame. This chapter also presents how the model is made parametric through the use of a bespoke Python model building script and individual configuration files for each process setup.

**Chapter Nine – User Subroutine Design and Implementation:** Presents the design of a Fortran program intended to add functionality to MSC Marc in order to process cutting increments according to the simplifications proposed in the previous chapter. This chapter explains how each component of the Fortran program works, including the mathematical theory underpinning the more advanced features such as NURBS surface generation and decomposition into triangles, ray-triangle intersection testing, triangle-triangle intersection testing and other features.

**Chapter Ten – Results and Discussion:** Validates the model created in the body of this work by applying and comparing it to the experiment developed in Chapter Seven. This chapter also discusses the quality of various outputs from the Fortran program such as mesh stability, element density and force recovery. Finally, this chapter applies the model to a hypothetical multi-angle problem and a proposed solution in order to compare radial imbalance.

**Chapter Eleven – Conclusion:** A short chapter concluding this work and summerising the results with reference to the objectives list given earlier in section 1.3. This chapter also presents ideas for future research and how the tools developed can be used by tooling designers.



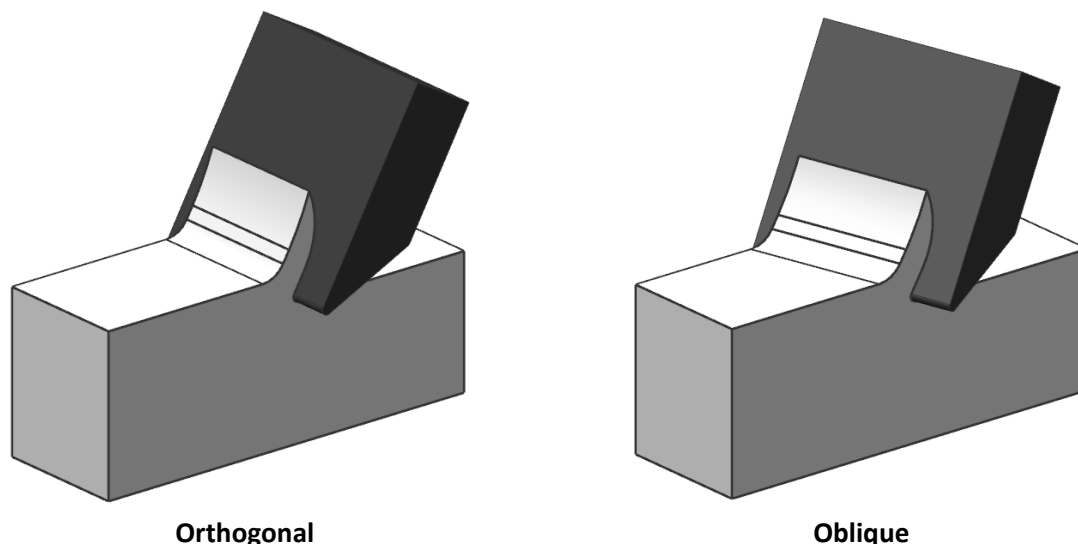
# Chapter Two – Mechanics of Machining

---

This chapter provides introduction to the mechanics of machining. Machining is a subset of metalworking techniques concerned with material removal from a workpiece using a tool material of superior mechanical properties to that of the workpiece. The material that is removed is referred to as *chip* or *swarf*, with the former term preferred in research literature (Oxley, 1989). The variation in machining configurations is vast, encompassing many different combinations of tool material, workpiece material, cutting geometry, speeds and fluids (lubricants and coolants). Parameter selection is based on a number of factors such as cost, quality and reliability. This chapter will focus on various properties of machining of relevance to the objectives stated in the introduction to this work. This review is written with reference to the book 'Fundamentals of Machining and Machine Tools' (Knight and Boothroyd, 2005).

## 2.1 Process Zone

Cutting geometry of the process zone is either orthogonal or oblique. Examples of each are given in figure 2-1. In these diagrams, the tool is the darker shaded body.



*Figure 2-1 - Orthogonal vs. oblique cutting geometry*

In orthogonal machining, chip flow is perpendicular to the cutting edge of the tool and forces can be fully described in the context of two dimensions thus making numerical models of this configuration significantly less complex than in oblique machining. In oblique machining, chip flow is directed by the inclination angle of the tool and three dimensions are required to fully describe the forces.

Figure 2-2 illustrates the cutting process zone during orthogonal machining and establishes the nomenclature used throughout this work.

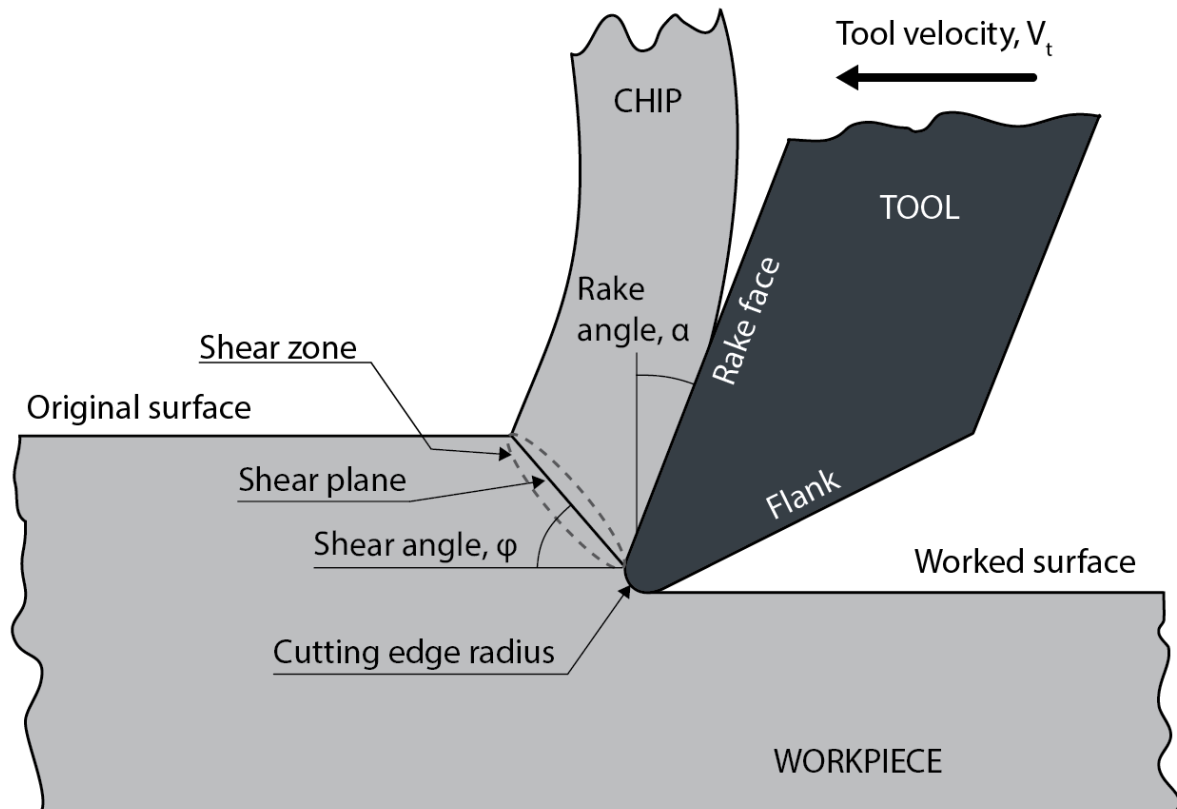


Figure 2-2 - Process zone diagram

Material separation is primarily achieved by shearing. High forces cause the workpiece material to shear along a shear plane, or more accurately a 'shear zone'. As the shear angle decreases, thickness of the chip increases. Friction between the tool rake face and chip play a fundamental role in maintaining the regularity of the shearing events, controlling the chip length and controlling the shear plane angle. The friction on this face is highly influential on cutting forces and heat generation (Olgun, Compton and Chandrasekar, 2003).

The rake angle of the tool is another important factor influencing chip geometry and shear angle. In experimental testing, increasing rake angle is found to correspond to increased efficiency since lower cutting forces are required. The trade-off is that an increased rake angle shifts the higher stresses to the narrowest part of the tool and therefore chipping and damage to the tool are more likely to occur (Astakhov, 2006).

An increased rake angle can lead to a reduction in material accumulation between the chip and tool referred to as built up edge (BUE). In general, BUE may reduce surface quality in the form of increased surface roughness, sporadic deposits and poor dimensional control. However BUE can occasionally have desirable effects such as reduced tool wear rate for chemically sensitive tool materials such as diamond (El-Gallab and Sklad, 1998). The rake angle can also be negative, which can increase the tool durability at the expense of much higher cutting forces and temperatures.

The flank angle, in contrast, plays no role in chip removal but is influential in determining the rate of certain wear mechanisms. An increased flank angle reduces rubbing between the tool and workpiece and

therefore decreases wear at the cost of increased risk of fracture due to less structural support for the cutting edge of the tool (Zhu, Zhang and Ding, 2013).

Cutting edge radius can have a substantial effect on cutting stability (resistance to chatter), tool force, wear rate and surface roughness. Zhao *et al.*, 2017, show that increasing cutting edge radius dramatically increases cutting force whilst decreasing wear rate. Lacerda and Siqueira, 2012, show that increasing cutting edge radius shares a nonlinear relationship with vibration and roundness deviation when machining sintered valve seats with pcBN.

## **2.2 Materials**

The valve seats used in the case study for this work are manufactured from a sintered high speed steel, branded 'AR20'. Ideally, it should be possible to cut this material reliably with polycrystalline cubic boron nitride (pcBN) tools. In reality however, cutting with pcBN has proved unreliable. This section introduces and discusses the materials referred to in this work.

### **2.2.1 AR20 Copper Infiltrated High Speed Steel**

High speed steel (HSS) is known for its superior thermal durability and wear resistance compared to that of conventional carbon steels. These properties make HSS ideal for cutting at higher speeds than could be achieved with carbon steel, hence the name 'high speed' was given. The increasing demands of the modern era have given HSS roles in other applications where the properties of normal carbon steel are no longer sufficient. Well known early adaptations of HSS in final products can be found in the aerospace industry where HSS was used for high durability bearings in jet engines. Nowadays the use of HSS in final products is commonplace. The promotion of HSS brings with it the demand for even harder tool materials such as tungsten carbide and more recently pcBN with which to cut HSS.

HSS can be hardened up to a level of 1000HV and resist softening up to 600°C depending on the chemical composition (Hoyle, 1988). A typical HSS is an alloy of carbon steel and tungsten, molybdenum, vanadium and chromium depending on requirements. More advanced HSS will contain cobalt. These alloying elements have, throughout most of HSS's history, been expensive. Consequently, studies in literature into ideal compositions often focus on cost (Dobrzański, 1995). The precise balance of alloying chemicals is a complex science. Generally speaking, tungsten and molybdenum contribute significantly to the superior hardness of HSS and ensure these properties are maintained at elevated temperatures. Vanadium confers wear resistance and abrasive qualities (Liujie *et al.*, 2007). Alloys of these compositions require high temperature heat treatment to complete the transition to HSS.

The valve seats studied in this work consist of Novofer AR20 which is a specially developed high speed steel for valve seats. The valve seat blanks are sintered from a highly compacted AR20 powder, and the pores are filled with copper in a process called copper infiltration. The material datasheet gives the material composition as shown in table 2-1 (Bleistahl, 2006).

Element	Wt.%
C	0.80 – 1.30
Co	15.0 – 22.0
Mo	9.0 – 14.0
W	2.5 – 4.5
V	1.3 – 2.3
Cr	3.5 – 5.5
Si	0.5 – 2.0
Mn	0.3 – 1.5
S	0.15 – 0.75
Cu	10.0 – 20.0
Fe	balance
Others	< 3

Table 2-1 - NOVOFER AR20 chemical composition

Figure 2-3 A) shows a scanning electron microscopy image of the valve seat surface. The data sheet describes the microstructure as consisting of “*fine distributed carbides and uniformly dispersed intermetallic phases in a tempered martensitic matrix. The solid lubricates are uniformly distributed and most of the pores are filled with copper*”. Figure 2-3 B) shows an optical image of the surface indicating the presence of a pore filled with copper.

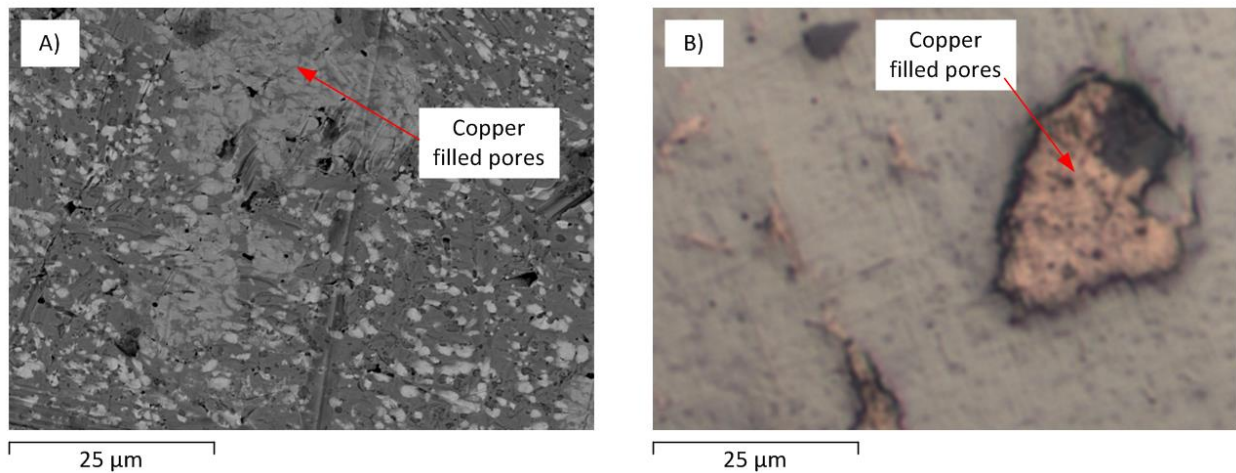


Figure 2-3 – A) SEM and B) optical images of valve seat surface

Figure 2-4 shows the microstructure of AR20 taken from literature, giving a clearer sense of the grain boundaries and sizes (Pierce *et al.*, 2019).

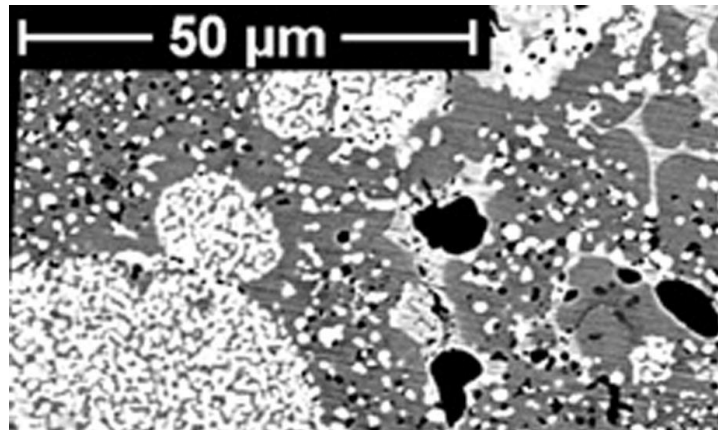


Figure 2-4 – Novofer AR20 microstructure (Pierce *et al.*, 2019)

Since the alloy is formed through sintering, it is expected to be porous. In the case of AR20 however, the material has been infused with copper. During the high temperature sintering process, the copper forms a liquid phase which readily flows into pores which would otherwise be empty. *Materials & Design*, state that this process produces an alloy with increased density and toughness (Wong-Ángel *et al.*, 2014). This type of process represents a relatively rare and novel approach to further improving the qualities of HSS for use as a valve seat material. Other product variations are gradually making their way to market also, for example, Dura-Bond's "Killer Bee" copper infiltrated valve seat material introduced in their 2014 catalogue ('Dura-Bond Catalog', 2014).

### 2.2.2 Cubic Boron Nitride

Cubic boron nitride (cBN / CBN / c-BN) super abrasive cutting tools are renown worldwide for their superior tool life, high material removal rates, ability to cut hard and tough materials, premium surface finish and low unit cost.

A reliable process for synthesising cBN was first developed by General Electric's corporate Research and Development Laboratory in 1957, to be later marketed under the trade name *Borazon* (Krar and Ratterman, 1990). cBN is comprised of equal parts boron and nitrogen and is the hardest of several allotropes of BN, with hardness second only to diamond (Monteiro *et al.*, 2013). cBN's principle advantage over diamond is that it can be used to machine ferrous alloys whereas diamond cannot. This is because at temperatures exceeding 800°C, the chemically pure carbon of diamond reacts with iron and alloying agents in steels to form carbides. At high temperatures diamond will also oxidise in air, which can only be avoided if machining takes place in an inert atmosphere.

cBN is not normally found in nature, and therefore the primary source of cBN for tool materials is synthesis. Synthesis of cBN is similar to that of diamond. Crystals averaging around 0.5 mm in diameter can be formed by placing boron and nitrogen mixtures under pressures approaching 45,000 atm in the presence of catalysis including magnesium, calcium or lithium nitrides (Wentorf Jr., 1961).

Liu, Wan and Ai, 2004, group cBN tools into three categories, high content (80 - 90 %wt), low content (50 – 65 %wt) and CBN coatings. These categories are described as follows:

- High content 'Pure' polycrystalline cubic boron nitride (pcBN) consists of many tiny individual crystals of cBN which have been fused together using high temperatures and pressures. Tools of

this grade often exhibit a self-sharpening effect, since the cBN crystals tend to fail along internal shear lines, exposing fresh surface (Fujimoto and Ichida, 2008). The crystals within pcBN materials are randomly oriented in all directions and so pcBN tools often show excellent isotropy. High Content pcBN tools are now the most popular variant of cBN based cutting material; and

- Low Content pcBN, also known as cBN composites, in which cBN crystals are interspersed within a binder material such as tungsten carbide. Composites can show greater resistance to wear under light cutting conditions, but with significantly reduced hardness (Edwards, 1993). Low content pcBN tools were originally developed as a compromise between the superior qualities of cBN based tools and the cost and expense of producing cBN (Eda, Kishi and Hashimoto, 1981). Improvements in the synthesis and manufacturing process mean that this grade is manufactured less frequently in favour of high content grades.
- Coatings (thin films), in which a thin cBN layer (typically  $2\ \mu\text{m}$ ) is applied to a durable substrate such as tungsten carbide through processes such as:
  - Physical Vapour Deposition (PVD); and
  - Chemical Vapour Deposition (CVD).

The coating is best applied over other coatings such as titanium aluminium nitride (TiAlN) in order to increase cBN binding. PVD and CVD coated carbides allow users to apply the superior hardness and wear resistance of pcBN to a durable high-strength substrate material, thus avoiding pcBNs brittle fracture failure mode (Uhlmann, Fuentes and Keunecke, 2009).

pcBN insert cutters are manufactured in a range of different shapes and sizes, including triangular, square and circular shapes. The pcBN inserts that Ford use for valve seat cutting typically have a maximum dimension less than 8mm. Figure 2-5 shows an engineering drawing of a typical triangular shaped insert. The pcBN layer of the insert (dark hatching) is bonded to a tungsten carbide substrate (light hatching). Tungsten carbide is used for its high impact resistance. The composite adds some toughness to the otherwise brittle pcBN layer.

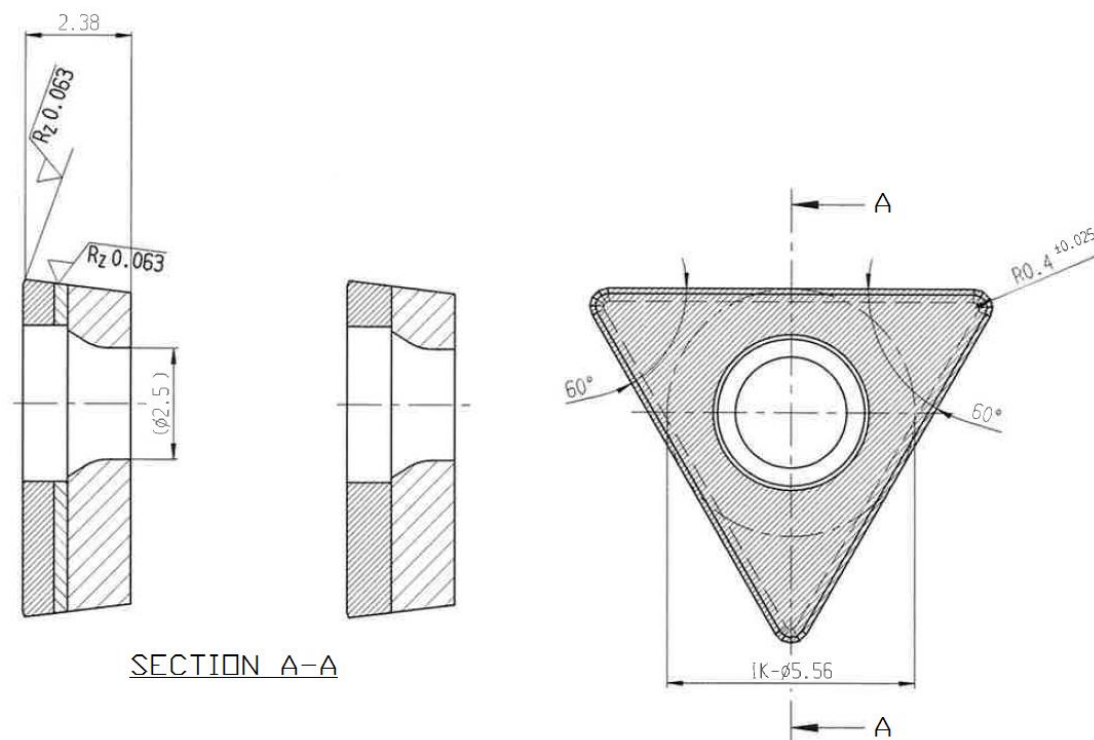


Figure 2-5 - Triangular-shaped pcBN insert

This type of insert would be secured with a tapered screw fixing through the hole in the centre. Alternatively, tools can be secured using clamps or by brazing to a tool holder. pcBN tools are sometimes provided as a 'tipped insert' where just one corner consists of pcBN.

### 2.2.3 Tungsten Carbide

Tungsten carbide cutting tools are ubiquitous worldwide. Renowned for their high working temperature, high cutting speed and high wear resistance they are an obvious choice for all but the very hardest workpiece materials for maximising cutting productivity.

Tungsten carbide tools belong to a family referred to as *cemented carbide tools*. They are made by combining carbon and tungsten powders at a ratio of 94:6 by weight, respectively. The mixture is then combined with a binder (cobalt), compacted and sintered in a furnace at approximately 1400°C (Knight and Boothroyd, 2005).

Tungsten carbide tools made in this fashion cannot be shaped after sintering and therefore normally take the form of indexable inserts. Common problems affecting tungsten carbide tools include cratering when machining steels. Resistance to this effect can be improved by adding tantalum carbide and titanium carbide to the parent composition.

## 2.3 Cutting Fluids

Cutting fluids can improve cutting performance by reducing temperature and cutting force. Often it is the case that certain operations cannot be carried out without them. Fluids can be applied directly to the process zone as a paste, gel or liquid using a brush, applicator or jet. In high-speed and CNC machining it is commonplace for cutting fluids to be applied as a medium or high pressure jet targeted at the process zone. Run-off is collected by the machine at the base, filtered, cooled and recirculated. Cutting fluids serve the following purposes (Trent and Wright, 2000):

- lubrication, which helps reduce the friction between the tool and workpiece, and thus helps reduce the cutting forces and power required;
- chip removal, which reduces the frequency of tool chipping or workpiece surface quality imperfections that result from debris being swept between the tool and workpiece;
- chemical protection, for example, preventing the exposure of surfaces to oxygen in the air; and
- cooling, which helps to prevent excess temperature accumulating in the tool and workpiece. High temperatures are associated with increase chemical wear rates and softening of the tool and workpiece. In more extreme cases, failure to control temperature sufficiently may lead to:
  - welding or fusing between the workpiece material and the tool;
  - a temporary expansion of the workpiece and tool bodies which can increase cutting forces and potentially prevent relative movement between the two;
  - poor surface quality and roughness since any thermal expansion that develops during machining will later shrink as the workpiece cools. Shrinkage may be uneven across a surface, and thus the flatness of a machined surface may not persist after cooling; and
  - an increase in the rate of chemical reactions between the tool and workpiece (such as diffusion), as well as within the tool and workpiece, including hardening and accelerated wear.

There is an increasing trend for manufactures to shift from using high quantities of coolant/lubricant fluid, to minimum quantity lubricant (MQL) processes, or no lubricant at all. These pressures are driven primarily

by ecological and financial concerns. Despite the ubiquity of cutting fluid use, it is arguably often applied when it is not actually necessary (Sreejith and Ngoi, 2000).

In MQL processes, the work zone is misted with the smallest quantity of lubricant which consistently lubricates the processes to the desired parameters.

In MQL processes, there is no recirculation or fluid cooling equipment. MQL is generally regarded as better for the environment and cheaper to maintain since less equipment (pipework, pumps and control systems) is required.

## **2.4 Chip**

Material that is removed from the process zone is referred to as chip. The nature of chip produced by a cutting process can reveal a lot about the process. For example, discolouration can indicate excess heat in the process zone, variable chip lengths can indicate issues with vibration. As previously mentioned, chip behaviour is strongly linked to phenomena such as built-up-edge (BUE) which can change the effective geometry and chemical/mechanical properties of the tool.

For metal cutting applications, chip will tend to form as either a:

- continuous ribbon in ductile materials, straight (parallel edged) ribbon, wandering ribbon (small variation in edge lengths either side), coiling ribbon (one edge of the chip longer);
- discontinuous, segmented chip, where continuous chip is prevented from forming due to periodic fracture from excessive strain in the chip; or
- powder, 'arc chip', 'elemental chip' or 'needle chip' in brittle materials characterised by the formation of very small irregular chips (Zsolt János Viharos, Markos and Szekeres, 2003).

The ductile properties of the chip are not necessarily those of the workpiece material. Chip is typically subjected to considerable strain and is heated and cooled rapidly resulting in permanent intermolecular changes.

Chip introduces a number of hazards to machine users. This is especially true for continuous chip which has a tendency to form clusters of razor sharp ribbon. Continuous chip often wraps around nearby structures, making its removal difficult. Continuous chip typically clumps in lower densities than its counterparts reducing the efficiency at which it can be stored, transported and disposed of. In brittle materials, powder and needle chipping can present respiratory and splinter risks. The more desirable type therefore is discontinuous chip. A common approach to limiting the hazards introduced by these effects is to include geometric features on the surface of cutting inserts which control and limit chip growth.

Moulded inserts, chip breakers and the use of varying rake angles are the most common approaches to chip control. Since many inserts used today are manufactured from a sintering process, it is possible to include quite complex geometries on their surface which direct the chip as required in order to break within the desired length range (Milton C. Shaw, 2005). Chip breakers take the form of any obstruction in the ejection path of the chip, typically on the tool face or tool holder.



## 2.5 Cutting speed and feed rate effects

Cutting speed describes the rate of relative motion between the tool and workpiece and is defined in terms of the following possible degrees of freedom:

- Spindle speed – the rate at which the cutting surfaces rotate, measured in RPM. This property is linked to the cutter velocity, typically measured in  $\text{ms}^{-1}$ .
- Feed rate – the rate at which the cutter moves relative to the workpiece, coaxial with the spindle axis, typically measured in  $\text{mm rev}^{-1}$ .

In metal cutting, cutting speed is the main factor dictating the rate of material removal, and therefore secondary factors such as:

- the rate of plastic deformation, leading to heat generation and other strain rate related phenomena;
- the frequency and amplitude of vibration, leading to fatigue, cracking and damage; and
- chip nucleation, growth and path due to strain rate hardening effects and chip body inertia. In high speed machining, studies show that cutting speed has a profound effect on chip shear angle and less of an effect on chip length (Sutter, 2005).

## 2.6 Heat in Machining

Most process zone cutting power is dissipated through heat. Heat is generated in the process zone primarily due to plastic deformation and friction between the tool and workpiece.

The resultant temperature rise and diffusion within a material is dictated by the material's specific heat capacity and thermal conductivity. The rate of heat rejection to the environment (to air or coolant) is governed by the thermal conductivities of the material and environment and the surface emissivity. Depending on the emissivity of the workpiece surface, the temperature during machining can be measured using non-contact methods such as infra-red thermometers or cameras.

Temperature effects can influence a wide range of material properties. High cutting temperatures are generally associated (either causally or by commonality) with,

- decreased hardness, (Zoya and Krishnamurthy, 2000), (for pcBN tools, the scale of decrease in hardness in response to high temperatures is greatly influenced by binder content, (Harris, Brookes and Taylor, 2004));
- increased cutting forces (Zoya and Krishnamurthy, 2000);
- increased rates of chemical wear processes such as oxidation, adhesive and diffusion wear (Tang *et al.*, 2019);
- thermal expansion (Hidnert, 1937; White, 1965; Roberts, White and Fawcett, 1983; Monteiro *et al.*, 2013);
- increased ductility (due to thermal softening);
- increased rates of damage mechanisms (Costes *et al.*, 2007);
- changes in the probability of chatter (Hajmohammadi, Movahhedy and Moradi, 2014); and
- increased vaporisation rate (loss rate) of coolants and lubricants, (Bell *et al.*, 1999).

Whilst these points are valid for most cases, occasionally negligible effect or the inverse effect is observed.

High temperatures are routinely used to 'age' or heat treat materials, since elevated temperature accelerates natural reactions occurring within a material. In varieties of steel, temperature can be used to

change the mechanical properties by altering the nature of ferrite phases within the steel. Annealing and case hardening are common examples of this application. This process can occur during machining, leaving patches of hardened material in regions exposed to rapid heating and cooling cycles.

## 2.7 Wear Mechanisms

Wear describes the process by which material is lost from one body to another or to the environment, as a result of physical interaction between bodies. The study of wear belongs mainly to the science of Tribology. Wear is regarded as a property of a system, rather than that of a material. Stachowiak, 2005, groups wear mechanisms in the following categories:

- mechanical wear (deformation and fracturing), which includes:
  - abrasive wear, the removal of ductile bulk surface;
  - adhesive shear and transfer, 'welding';
  - accumulated plastic shear flow; and
  - fatigue, wear by crack initiation and propagation.
- chemical wear, governed by the growth rate of a chemical reaction film formed by tribochemical interactions between the material and surrounding medium (such as air, lubricants, workpiece or tool material). This can take the form of hard tribofilms such as iron oxide, or soft tribofilms such as silica gel. The principle wear mechanisms for:
  - hard tribofilms are:
    - shear failure of ductile tribofilm; and / or
    - delamination of brittle tribofilm.
  - soft tribofilms are:
    - accumulated plastic shear flow; and / or
    - shaving of soft tribofilm.
- thermal, surface melting due to frictional heating:
  - local melting; and
  - transfer or scattering.

A principle vector for chemical wear is diffusion, or 'diffusion wear' in which chemical elements diffuse into the tool material under the intense temperatures and pressures at the tool-chip interface. After diffusing, they react with elements in the tool material (alloying agents in the case of steels or TiN or TiAlN binder substrates in the case of pcBN). The reactions result in a layer with reduced mechanical properties (Trent and Wright, 2000). The layer is either abraded slowly or sometimes removed spontaneously by chip flow exposing fresh surface on which the wear mechanism starts again (Costes *et al.*, 2007).

The potential of pcBN to replace tungsten carbide as the primary cutting material for machining hardened steels has led to considerable research into its performance in machining. Manufacturers generally desire tools with superior wear resistance. With pcBN tools, the most influential factors affecting wear rate are cutting speed, followed by feed rate and depth of cut (Huang and Liang, 2005). This suggestion, proposed by Yong Huang *et. al.* was substantiated by an experimentally-verified parametric numerical model. The model invokes the Taylor equation for predicting tool life, given in equation 2-1, where  $V$  is cutting speed,  $T$  is the tool life in units of time and  $n$  and  $c$  are derived parameters based on experimental observation for a specific machining configuration.

$$VT^n = c \quad (2-1)$$

Li *et al.*, 2002, used the Taylor equation to optimise cutting speed for coated carbide and ceramic inserts. They used experimental data gathered from turning experiments with Inconel 718 to derive values for  $n$  and  $c$  for a range of tool grades and rake angles.

The Taylor equation does not account for feed rate, depth of cut, tool geometry, lubrication or any other parameters that may change between operations. The Taylor equation was originally developed for HSS (Taylor F.W, 1907) well before pcBN tools were in use, so it is unclear how well this equation describes materials such as pcBN which have very different microstructural, inter-molecular and chemical properties.

Barry and Byrne, 2001, propose that chemical interactions are the dominant wear mechanism between pcBN tools and workpiece materials containing elements such as Mn, Si, S, O, and in particular, those containing Al. In which case the previous observation, that speed is the rate limiting factor, may be solely due to the accelerated rates of chemical reactions due to higher temperatures which are known to develop at higher cutting speeds.

The precise wear mechanisms of pcBN inserts appear to vary significantly between pcBN materials without a binder and lower grade pcBN materials with a binder. Kato, Shintani and Sumiya, 2002, found that the pure 'binderless variants' although lasting significantly longer than their binder-based counterparts do seem more susceptible to thermal strain cracking. Binder-less pcBN tools seem to remain sharp for longer and deliver a lower machined surface roughness.

## **2.8 Cutting Tool Failure**

Failure of super hard cutting tools such as pcBN typically occurs in the form of chipping on the cutting edge. This type of damage is normally catastrophic and the tool can no longer be used for machining. This section presents some of the causes and events that can weaken the tool and result in chipping of the cutting edge.

### **2.8.1 Stress**

A material loaded to a stress within its elastic limit shows no obvious damage. In cases where the loading is sustained or oscillates over extended periods, damage due to creep or fatigue is likely to occur. Although the crystalline structure of cBN crystals within pcBN tools might be expected to be vulnerable to brittle fracture, this particular failure mode is less often seen in machining, primarily due to the dominance of more aggressive life-limiting wear and damage mechanisms affecting pcBN tools such as chemical and thermal damage (Liew, Ngoi and Lu, 2003).

Stresses beyond the elastic limit or yield stress,  $\sigma_Y$ , of a material will cause plastic deformation. Under tensile loading, plastic deformation reduces the cross-sectional area of a material and therefore the effective stress increases. When stress reaches the material's ultimate tensile strength,  $\sigma_{UTS}$  the material will begin 'necking' until failure at the materials fracture stress,  $\sigma_F$ . Plastic deformation is less relevant to cBN since ceramics typically show very little plasticity before failure.

### 2.8.2 Thermal Effects

During multi-angle valve seat machining, pcBN cutting inserts are subjected to thermal cycling, which can lead to premature tool failure due to thermal damage. Thermal damage can be caused by a number of factors including:

- thermal shock, where uneven thermal expansion in a volume can cause cracking, ripping and tearing;
- oxidization or other chemical reactions that result in poor internal cohesion; and
- molecular restructuring or decomposition, such as transitions between cubic boron nitride and hexagonal boron nitride.

W. König et. al. compared the wear mechanisms of polycrystalline Diamond (pcD) and pcBN tools to conclude that recrystallisation in the binder layer induced by elevated cutting temperatures is the dominant damage mechanism in pcBN. This results in accelerated rates of crater wear during cutting (König and Neises, 1993). However, their observation was entirely dependent on the particular binder material in the study and no comment was made on damage within the cBN crystals. Other binder materials therefore may behave differently.

### 2.8.3 Vibration

Vibration in machining is well known as being one of the most aggressive factors affecting tool life and machined surface finish. Whilst high frequency and small amplitude vibration is sometimes intentionally introduced to reduce tool force (Brehl and Dow, 2008), generally lower frequency and high amplitude vibrations raise the peak cutting forces. This often leads to micro-chipping and micro-cracking which can dramatically reduce tool life (Toh, 2004). Sources of vibration include:

- machine foundation noise (from other machines in close proximity);
- machine motor vibration from:
  - noise in AC electrical power delivery;
  - motor bearings; and
  - motor imbalance;
- imbalance in moving parts including tools, gears, shafts, belts and bearings;
- imbalance in cutting force in tools with multiple cutting edges;
- fluid moment, such as coolant jets, air bubbles in fluid lines and pump action;
- shared air and fluid lines, shockwaves which can propagate through shared lines to be introduced to the machine via fixtures and pipe harnesses;
- periodic metal shearing events in the workpiece shear zone; and
- other vibration arising from the tool and workpiece interface.

Vibrational energy accumulates in the system in bodies with natural frequencies matching the frequency of vibration. Resonance is generally controlled at the machine/tool design stage by the addition of damping structures which dissipate vibrational energy as heat or by controlling the layout of masses throughout the system. Structures are selected which have natural frequencies in bands outside of the frequencies expected during cutting.

The wide range of sources and exacerbating factors in machining operations make the characterisation of vibration and its effects particularly challenging. Hamed Moradi et. al. looked specifically at nonlinearity, internal resonance, tool wear and damping effects in milling processes (Moradi *et al.*, 2013). It was found that as cutting force increases, so does the amplitude of resonance for the first resonant frequency. Higher

spindle speeds increase the probability of resonance. Machine tools of higher stiffness produce less vibration amplitudes.

Chatter is a dramatic and violent form of vibration at the tool and workpiece interface. It is caused by interaction between the tool and workpiece but can be induced from external vibration. Chatter is often regenerative in cases where extreme vibration causes deeper depths of cut (Fu and Zheng, 2014).

In this work, vibration introduced from unequal cutting radial forces between the pcBN inserts of the tool holder may cause vibration which initiates chatter. This exact phenomenon was the cause of machining problems observed in a very similar scenario studied by Lacerda and Siqueira, 2012. Production line engineers initially decreased cutting speeds to work around the problem at the expense of reduced productivity. The study allowed changes to the tool holder and tool geometry to be made which equalised the cutting forces and thus reduced the vibration significantly.

Thermal effects can, under certain circumstances, contribute to chatter if thermal expansion is permitted to increase the pressure between the tool and workpiece. Some research has been conducted predicting this phenomena using finite element modelling (Hajmohammadi, Movahhedy and Moradi, 2014). Chatter was shown to develop in both thermal and non-thermal models but the amplitude of chatter was higher in models which did not take into account thermal effects. In this regard non-thermal models give more conservative predictions as to when chatter will occur.

## **2.9 Chapter Summary**

This chapter has provided a good introduction to the fundamentals of metal cutting, in particular, how the process zone works and what physical phenomena are present during metal cutting. The relationship between cutting performance and various cutting parameters such as spindle speed, feed rate, lubrication regimes and cutting materials was presented.

Also discussed were some of the pitfalls of machining, including wear, damage and vibration with reference to investigations into each of these effects from literature. Some of these effects may be partially responsible for the cutting insert failures observed in the case study for this work and are to be investigated further.

Topics addressed in this chapter highlight the complexity of the cutting process zone. Simulating metal cutting is core to the objectives of this work. Care must be taken to make sure these effects are given due consideration in the design of the final model to ensure accuracy when simulating multiple passes of multi-angle valve seat machining.

# Chapter Three – Non-Linear Finite Element Analysis

---

Finite Element Analysis (FEA) is one of the most powerful tools available to engineers for predicting the performance of structures when subjected to real world physics. The inherent versatility of Finite Element Analysis allows the simulation of individual or multiple physical phenomena including elasticity, plasticity, fluid flow, heat conduction, magnetic flux, electrostatic attraction and electrical current.

Whilst Matrix Stiffness Methods (MSM) originate in the 1920s, the first reported computational implementation of Finite Element Analysis comes from within the commercial environment of the aerospace industry in 1956 (Turner *et al.*, 1956). Since then, its capabilities and applications have grown and diversified as computer power becomes more readily available and financial pressures shift the burden of testing from the real world to the virtual world. FEA is now increasingly being applied to multi-physics and non-linear (time dependant) problems.

This chapter looks at non-linear finite element methods with a focus on how they can be applied to meet the objectives of this work. This chapter is written throughout with reference to:

- *The Finite Element Method, Its Basis and Fundamentals*, Zienkiewicz, Taylor and Zhu, 1967;
- *The Finite Element Method For Three-Dimensional Thermomechanical Applications*, Dhondt, 2004;
- *The Finite Element Method in Engineering*, Rao, 2010;
- *Nonlinear Structural Mechanics*, Lacarbonara, 2013;
- *Non-Linear Finite Element Analysis of Solids Structures*, Crisfield, 1996;
- *Finite Elements – Their Design and Performance*, MacNeal and Richard, 1993; and
- *Programming the Finite Element Method*, Meyers, Smith and Griffiths, 1989.

## 3.1 Theory of Non-Linear Finite Element Analysis

FEA is the practical implementation of mathematical techniques such as the Finite Element Method (FEM). FEA Studies can be 1D, 2D or 3D in either linear or non-linear configurations. FEM is very calculation and memory intensive and is therefore best suited for computers.

FEM is based on the principle of simplifying a load case by dividing the geometry into elements and calculating their response to a given physical condition, such as mechanical or thermal loads.

Each element within the structure is bounded by nodes through which physical interaction propagates throughout the system. The behaviour of elements is governed by equations which approximate real-life physics. The choice of equations used depends principally on the physics of interest, but may also be influenced by factors such as speed of evaluation and the desired accuracy.

The simplest form of the FEM consists of the Direct Stiffness Method (DSM) in which a force vector  $\{F\}$  is related to a displacement vector  $\{U\}$  by an element or global stiffness matrix  $[K]$  as shown in equation 3-1.

$$\{F\} = [K]\{U\} \quad (3-1)$$

In this equation, the global stiffness matrix  $[K]$  describes both the stiffness of elements as well as their geometry. Typically a matrix solution method such as Gaussian elimination is used to resolve the unknown displacement and force components.

For a one dimensional bar problem in uniaxial tension or compression,  $[K]$  for an element,  $e$ , would take the form of equation 3-2 where  $A$  is the cross sectional area of the element,  $L$  the Length and  $E$  the elastic modulus.

$$[K_e] = \frac{AE}{L_e} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \quad (3-2)$$

Figure 3-1 shows a round bar of two different cross-sectional areas under uniaxial tension. The system can be discretised as two elements with nodes at the interfaces.

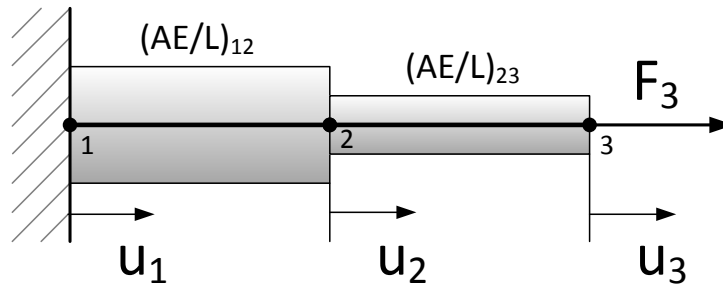


Figure 3-1 - 1D FE Example

The two elements have a common node in the middle, therefore the equilibrium equation for this node is shared. The global stiffness matrix of the system is assembled as shown in equation 3-3.

$$[K] = \begin{bmatrix} \frac{AE}{L_{12}} & -\frac{AE}{L_{12}} & 0 \\ -\frac{AE}{L_{12}} & \frac{AE}{L_{12}} + \frac{AE}{L_{23}} & -\frac{AE}{L_{23}} \\ 0 & -\frac{AE}{L_{23}} & \frac{AE}{L_{23}} \end{bmatrix} \quad (3-3)$$

The boundary conditions are then set,  $u_1 = 0$ ,  $F_1 = -F_3$ ,  $F_2 = 0$ , allowing the unknown displacements  $u_2$  and  $u_3$  to be solved as shown in equation 3-4.

$$\begin{Bmatrix} F_1 \\ F_2 = 0 \\ F_3 = -F_1 \end{Bmatrix} = \begin{bmatrix} \frac{AE}{L_{12}} & -\frac{AE}{L_{12}} & 0 \\ -\frac{AE}{L_{12}} & \frac{AE}{L_{12}} + \frac{AE}{L_{23}} & -\frac{AE}{L_{23}} \\ 0 & -\frac{AE}{L_{23}} & \frac{AE}{L_{23}} \end{bmatrix} \begin{Bmatrix} u_1 = 0 \\ u_2 \\ u_3 \end{Bmatrix} \quad (3-4)$$

The performance of materials is defined in terms of stresses and strains rather than forces and displacements. Further post-processing is therefore required to extract the stresses and strains from the equilibrium equation.

### 3.1.1 Variational Methods

In the previous example, nodal values are in direct equilibrium with one-another. For more sophisticated elements which call upon relationships governed by partial differentials, it becomes necessary to use variational or weighted-residual methods.

Variational methods in structural analysis work on the basis of energy equilibrium. The Rayleigh-Ritz method when applied to structural finite element methods requires that for all points throughout a system, the change in potential energy,  $\Pi_p$ , across a degree of freedom,  $a$ , is zero as shown in equation 3-5.

$$\frac{\partial \Pi_p}{\partial a} = 0 \quad (3-5)$$

The potential energy within the system,  $\Pi_p$  is equal to the sum of  $U$  the strain energy and  $\Omega$  the potential energy of applied loads.

A shape function  $[N]$  relates calculated intra-element displacements  $\{u_e\}$  to system nodal displacements,  $\{u\}$  as shown in equation 3-6.

$$\{u\} = [N]\{u_e\} \quad (3-6)$$

Equation 3-6 can be represented in terms of strains as shown in equation 3-7.

$$\{\varepsilon\} = \left\{ \frac{du}{dx} \right\} = \left[ \frac{dN}{dx} \right] \{u_e\} = [B]\{u_e\} \quad (3-7)$$

Strain energy in an elastic system is given by equation 3-8, and the potential of applied loads by equation 3-9.

$$U = \int \frac{\sigma \varepsilon}{2} dV = \frac{1}{2} \int E \varepsilon^2 dV = \frac{1}{2} \int \{\varepsilon\}^T [E] \{\varepsilon\} dV \quad (3-8)$$

$$\Omega = -\{u_e\}^T \{F_e\} \quad (3-9)$$

The potential energy of the system is therefore given by equation 3-10.

$$\Pi_p = \frac{1}{2} \int \{\varepsilon\}^T [E] \{\varepsilon\} dV - \{u_e\}^T \{F_e\} \quad (3-10)$$

Substituting equation 3-7 into 3-10 gives the relationship in equation 3-11. Finally, equation 3-11 can be rearranged according to equation 3-1, yielding the formulation given in equation 3-12.

$$\frac{\partial \Pi_p}{\partial \{u_e\}} = \int [B]^T [E] [B] dV \{u_e\} - \{F_e\} = 0 \quad (3-11)$$

$$[K_e] = \int [B]^T [E] [B] dV \quad (3-12)$$

With this equation it is possible to formulate an element stiffness matrix from exact governing equations which rely upon partial differentiation.



### 3.1.2 Large Displacement and Strain Behaviour (Lagrangian Methods)

The approach discussed up to this point is based on the principle of energy conservation. For each element, the work required for deformation of that element should equal the boundary work on that element and therefore, the total work required for deformation of the system should equal the boundary work on the system.

Equation 3-10 requires that energy is calculated by integrating through the volume of the element. This approach is permissible only where nodal points throughout the geometry do not move significantly from their origins, implying that element volume is constant. For larger displacements, this approach is inappropriate since the element volume will change significantly.

Any given time step within a finite element simulation has a start and end state. Both states must remain at energy equilibrium or in other words, no unaccounted energy can be allowed to enter or leave the boundary of the simulation between the initial and final state of the time step.

To account for volume changes whilst maintaining energy equilibrium creates a cyclic dependency, where the displacements of nodes as they are at the end of the time step are required as inputs to the element formulation at the beginning of the time step.

Lagrangian methods tackle this problem by introducing a transformation that allows the integrals, such as those introduced in section 3.1.1 to be evaluated over a fixed volume.

Equation 3-13 gives equation 3-10 in terms of the Piola-Kirchhoff stress tensor  $\{\sigma\}_P$  and Green's strain tensor  $\{\varepsilon\}_G$ . This relationship is known as the Total Lagrangian formulation in which all integrals are calculated using the initial undeformed state of the structure.

$$\int \{\varepsilon\}_G^T \{\sigma\}_P dV = \{u_e\}^T \{F_e\} \quad (3-13)$$

The Updated Lagrangian (UL) formulation given in equation 3-14 uses states from the next time increment.

$$\int \{\varepsilon\}_{A(t+1)}^T \{\sigma\}_{C(t+1)} dV = \{u_e\}^T \{F_e\} \quad (3-14)$$

Where  $\{\varepsilon\}_A$  is the Almansi strain tensor,  $\{\sigma\}_C$  is the Cauchy stress tensor and  $(t + 1)$  refers to the next time increment.

Lagrangian methods are sensitive to large element distortions. In structures which change shape significantly during the simulation, remeshing is often required to prevent excessive elemental distortion. Remeshing works by generating a new mesh over the deformed geometry and interpolating previous nodal values to new nodal values. This process often results in a smoothing effect between nodal values. Mesh structure and element choice have a significant effect on simulation results, and there is much interest in reducing this error when using remeshing and Lagrangian methods.

The sensitivity of quadrilateral elements in metal forming simulations using the Updated Lagrangian (UL) formulation and remeshing have been analysed and discussed in depth by Srikanth and Zabaras, 2001, where it is shown that the presence of severely distorted elements may lead to premature termination of the simulation due to inability to converge. The study highlights that robust automatic/adaptive remeshing capabilities are required in order to deploy UL methods successfully.

### 3.1.3 Contact and Friction

In FEA, many problems can be evaluated using a single body subjected to nodal loads and boundary conditions. Those that cannot, such as machining must be simulated using multi-body contact physics. Contact detection is required to determine when two bodies moving relative to one another come into contact with one another. Figure 3-2 shows a scenario where contact detection is required because the light blue mesh is approaching the static black mesh with a fixed velocity (purple).

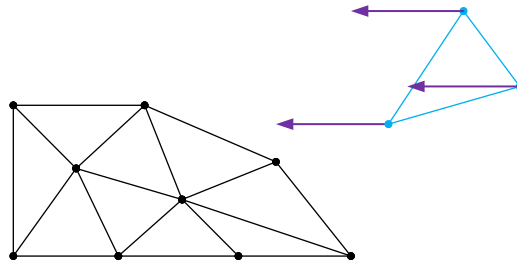


Figure 3-2 – 2D meshes approaching contact

Without contact detection, nodes and meshes that are not directly connected, will move freely through one another. This is also true for self-contact. Contact detection works only between iterations and not during.

The common types of contact handling approaches are discussed as follows.

**Node-to-node** is by far the simplest method in which the finite element code will watch for nodes moving near one another. When one node enters the detection radius of another node, the two will become 'locked' together until the direction of force changes to induce a separation or sliding motion. Figure 3-3 (left) shows where contact would be detected.

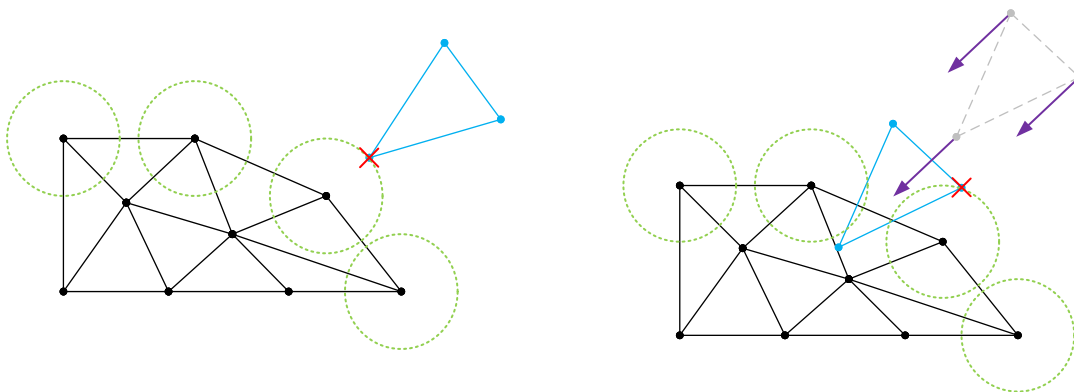
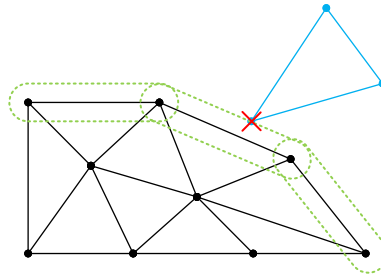


Figure 3-3 – 2D node-to-node points of contact detection

As the figure shows, the detection radius must be sufficiently large so as not to leave gaps between nodes. 3-3 (right) shows how the leading node can penetrate without detection and, in this case, how detection only occurs late when a trailing node enters a detection radius. Node-to-node contact is appropriate for contact bodies with mesh conformity, in which it can be guaranteed that approaching nodes will only ever contact on other nodes. For less conformal models, other approaches must be used such as:

**Node-to-segment**, which requires additional computational power to extend the contact detection zone between nodes. The contact detection zone will only detect nodes entering this region and so segments will not be detected. Figure 3-4 shows a node-to-segment contact detection.



*Figure 3-4 – 2D node-to-segment point of contact detection*

**Segment-to-segment** contact is the most computationally expensive approach which can detect intersecting segments. Contact-dependant effects, such as friction and sliding will be evaluated over the entire segment as opposed to approximated at nodes.

Contact detection requires that a suitable time step is used. This is important since if the time step between iterations is large, nodes may move from outside a contact detection radius to beneath the surface of bodies and skip contact detection zones altogether. This can introduce numerical error and cause remeshing algorithms to fail.

Contact detection and handling is computationally expensive, and should be reserved for bodies which are expected to come into contact with one another. Once contact has been established, contact mechanics algorithms are used to calculate the balance of forces (stress projection), friction, separation criteria and adaptive mesh requirements. Wriggers, 2005, gives an in-depth explanation of this process.

### **3.1.4 Convergence**

For any discretised system, the energy relationship given by equation 3-5 will approach zero as element density increases. For all but the very simplest problems however, it will never reach zero. The minimisation of this error in energy methods is known as convergence and is a useful criterion for judging the fitness of a particular discretisation for the problem it represents. In recursive iterative processes, it can be used to judge at which point the accuracy of the current iteration is sufficient to move to the next iteration.

The designer of a finite element study has the option of increasing element densities either globally or locally in areas of increased interest in order to achieve greater numerical accuracy at the cost of computational power. Often this relationship is exponential, and eventually, the increase in accuracy achieved in exchange for further increases in computational effort diminishes. Typically, a balance between the two pressures of computational power and accuracy is required, at this point a simulation is said to have ‘converged’. The point at which a simulation converges is up to the designer of the study, and can be influenced by project specifications, experience, safe margins or phenomena of interest.

### 3.2 Basic Procedure of FEA in Solid Mechanics

The following points outline a typical approach to solving a problem with FEA.

- Problem definition

The problem definition serves as a statement of intended outcomes, as well as the time frames allowed, boundary conditions, computer power available, desired accuracy and resources available (such as material property information). This stage is critical for on-going verification and validation.

- Simplification

It is often the case in FEA that a problem can be simplified. For example, parts of a structure which are unlikely to influence results can be removed. Symmetry is very common in design, allowing most structures to be sectioned. Some three dimensional interactions can be modelled in two dimensions, for example, uniform loading on a uniform structure.

- Discretisation

Discretisation is the process of dividing the problem or model into discrete sections or elements. In structural analyses, this step would constitute the design and implementation of a suitable element mesh. Important decisions must be made during this step, such as element density, element type, the style of mesh (for example, conformal or non-conformal) opportunities for simplification and domain boundaries (for multi-processor solvers).

In non-linear simulations involving large deformation, it is also important to configure iterative / adaptive meshing. Unlike the initial mesh structure in which nodes can be placed manually, iterative meshing is purely rules-based. Rules that are appropriate for remeshing early in a simulation may not suit interactions later. In these scenarios, manual remeshing between increments is an option. Some finite element solvers, for example MSC Marc allow parts of the simulation to be divided into sequential load cases in which rules for adaptive remeshing can vary, (MSC Software, 2016a).

- Apply boundary conditions

All FEA models must specify suitable boundary conditions. Boundary conditions are applied to nodes and typically take the form of one or more fixed displacements and one or more applied forces.

- Material Selection

The most basic material models required in structural FEA require the materials Young's Modulus, and Poisson's Ratio. Material models can be developed to include many parameters and functions including plastic strain, damage and thermal effects, which may or may not be called upon depending upon the choice of element. FEMs can use multiple material models to represent where different materials are used in the real system.

- Assembly of the Element Stiffness Matrix,  $[K]$

The Element Stiffness Matrix (ESM) represents the geometry and stiffness of all the elements involved in the study. The ESM typically takes the form of a large diagonal matrix comprised of mostly zeros. This assembly is performed by finite element software and is usually invisible to the user, but due to the size of the matrix, consideration must be made to ensure enough system memory is available to contain it.

- Solution

Construction of the system's governing equilibrium equation leaves unknowns in the applied nodal forces and nodal displacement vectors. Solving for these unknowns is done using mathematical techniques such as Gaussian elimination. Depending on the size and complexity of the model, this step is very computationally expensive and can be accelerated by using multiple processors.

- Recovery of results

The solutions obtained from the previous step yield nodal values for forces and displacements. These can be used to calculate meaningful information such as stress, strain and damage fields.

### 3.3 Mesh Refinement

A significant source of error in non-linear finite element studies derives from discretisation both in space and time. Significant errors are introduced when the element density is too low in the area of interest. Refinement techniques can be used to minimise these errors whilst making efficient use of processing power. Some common approaches to 2D mesh refinement are shown in figure 3-5, Wriggers, 2005,

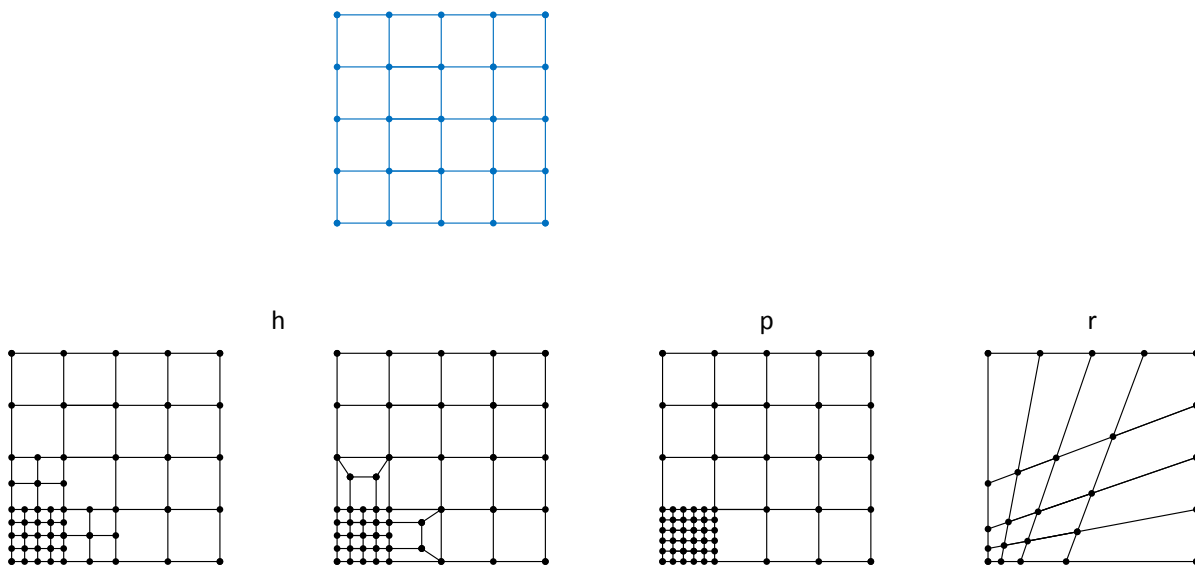


Figure 3-5 – Mesh refinement examples

Where:

- the blue mesh is the parent mesh from which the h, p and r meshes derive,
- h refinement refers to the addition of hanging nodes,
- p refinement increases the order of polynomials but requires considerably more computation power per refinement, and
- r refinement both increases the density at points of interest and reduces density where limited activity is taking place. R refinement is the optimum choice where computation power is limited.

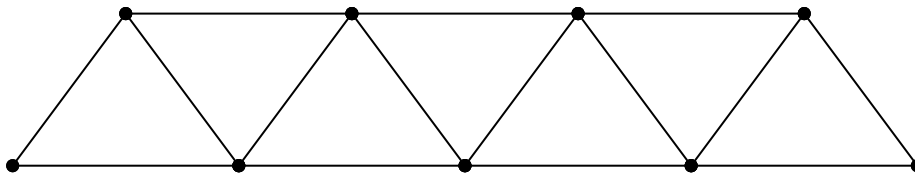
### 3.4 Remeshing

The accuracy of elements deteriorates as they are skewed by deformations taking place during simulation. This, and factors such as separation and material flow that take place in large strain models, make it necessary to periodically regenerate the geometry mesh.

Once a solution to a single time step has been calculated, remeshing algorithms have an opportunity to refine the mesh for the next iteration. For machining simulations, remeshing typically aims to further subdivide elements which, in the previous iteration, have been heavily distorted or damaged. A good remeshing algorithm will refine areas that show a high gradient in the phenomenon of interest, such that the jump from element to element in the magnitude of the phenomenon is more or less consistent throughout all elements in the model (Marusich and Ortiz, 1995).

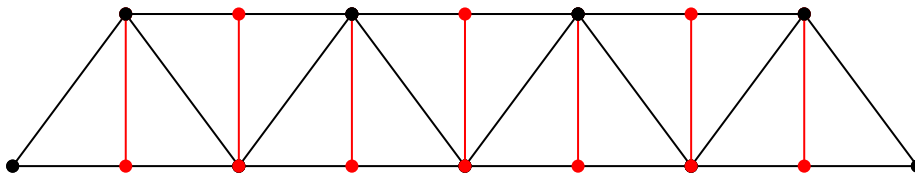
Remeshing should be used sparingly, however, since quality is lost between steps. This is because using nodes from the previous iteration is not always possible in the newly generated mesh, which can be illustrated as follows:

Consider the triangular mesh with seven elements in figure 3-6.



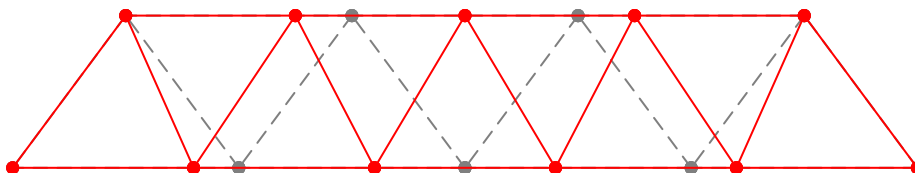
*Figure 3-6 - Original seven element mesh*

The simplest way to remesh this structure and preserve node locations and data is to subdivide each element. This produces the element mesh in figure 3-7



*Figure 3-7 – Nodes preserved, fourteen element remesh*

However, this simple subdivision has doubled the element density in this area. In practical implementations of FEA, a doubling of element density can be expected to give a four-fold increase in the time required to solve the problem. For most applications, this cost far outweighs the benefits of preserving nodal information and so remeshing algorithms will instead create new nodes as shown in figure 3-8.



*Figure 3-8 – Interpolated nine element remesh*

The remesher approximates the quantities for the new nodes based on the nodes surrounding it. The interpolation process can be linear or polynomial, or based on more advanced material models. The interpolation process itself will add processing demand but this is generally less than that from adding

elements. Any approach used will introduce error which will multiply with each repeat of this process. Common errors introduced are smoothing effects, which can reduce peak values as well as volume loss and rounding of sharp corners.

### 3.5 Model Verification and Validation

Ensuring model quality is only achieved through correct model validation and verification. Verification ensures that all inputs to the model are correct and that solution methods evaluate accurately. This includes the justifications and reasoning behind assumptions and simplifications, measured numerical values from material specimens, measured system parameters and solver source code bugs.

Verification ensures that numerical results compare well to data collected from experimentation. One does not necessarily guarantee the other. Both must be achieved, especially in parametric models where the models value is in its ability to accurately simulate the system across a wide range of different parameters.

NASA's FEMCI book is a reference inspired by real world uses of the NASTRAN finite element code, (Irish and Simmons, 2009). The book lists four basic mathematical checks applicable to almost thermo-mechanical finite element models. Specifically:

- Unit enforced displacement and rotation – where the model is translated and rotated to ensure boundary conditions behave as expected and no hidden unwanted boundary conditions exist in the model.
- Free-Free Dynamics with a Stiffness Equilibrium Check – a check to ensure that the model will act as a rigid body when unconstrained and unloaded. In this mode the geometry should not collapse, deform or inflate and there should be no significant stress fields present.
- Unit Gravity Loading – one of the most basic tests in which the model is subjected to 1 G gravity and its weight compared to the real world model weight.
- Unit Temperature Increase – where a structurally unloaded model has its temperature raised by a fixed amount and the resultant thermal expansion compared to real models.

A valid model is one which outputs data that closely resembles data that would be observed in the real physical model, (Law, 2001). This match is never absolute and so model validity is described in terms of percentage confidence. Unfortunately, definitions rarely agree upon a more definitive description than this and it is therefore left to the user to define their own validity criteria from the project objectives and specification.

Borvik *et al.*, 2001, verified their complex model of viscoplasticity and ductile damage for ballistic impact and penetration mechanics by performing numerical simulations of a simpler plate perforation test with the same material and physical models. The simulations were compared to high speed imagery of a physical experiment. The authors were satisfied with the approach, concluding that the numerical model showed good agreement with the experimental results. This is a useful example of how confidence in a model can be judged.

### 3.6 Modelling Software Selection

There are many commercial and free finite element analysis packages available. Careful selection of an appropriate package is necessary to meet the objectives of this project within a reasonable time frame. Since this work aims to deliver a fully parametric simulation tool supported by bespoke code to implement the model developed, familiarity with supported languages is also an important factor influencing the amount of progress that can be made.

The following software features have been identified as essential to meeting the objectives of this work:

- Support for large deformation / non-linear solver – The solver must support solutions for large deformation and non-linear physics. This feature is essential since the workpiece will undergo significant geometry changes during machining.
- Mesh generation – Ideally the package must have reliable built-in support for generating a mesh. Most packages offer support for importing mesh geometry generated in third party meshing packages such as MSC Patran. For the end user, this process would add additional complication and so it would be more ideal to have a fully integrated solution that doesn't require external mesh generation.
- Parametric model definitions (scripting support) – The final model must be capable of reconfiguring for different cutter layouts. Parametric support is commonly implemented through the use of scripting languages such as Python, MATLAB and Java. For this work, Python is strongly preferred due to the author's familiarity and experience with this language.
- User subroutine support – User subroutines can be used to implement supplemental physical behaviour during simulation. They are compiled from low level languages such as Fortran and C++, and can therefore execute with higher efficiency than higher level interpreted languages such as Python. The core functionality identified as necessary for the objectives of this work was not present in any of the packages reviewed and therefore must be implemented through user subroutines. And finally;
- The user interface should be easy to learn and intuitive since the final model is intended to be used by third parties.

Four commonly used packages stand out, including:

- **MSC Marc** developed by MSC Software and based on MSC NASTRAN finite element code. Marc is a very mature package and is frequently used in machining studies, such as modelling orthogonal cutting (Bil, Kiliç and Tekkaya, 2004); ultrasonically assisted turning (Mitrofanov *et al.*, 2005); and metal cutting with plasticity modelling (Svoboda, Wedberg and Lindgren, 2010);
- **Deform 3D Machining**, which has been used to model 3D turning (Ceretti *et al.*, 2000), tool wear (Attanasio *et al.*, 2008) and drilling (Majeed, Iqbal and Lv, 2018);
- **ABAQUS**, developed by Dassault Systèmes, which has been used to simulate drilling of fibre metal laminates (Giasin *et al.*, 2017) and model the influence of various friction models on finite element results (with comparisons to DEFORM 2D results) (Malakizadi *et al.*, 2017); and finally
- **COMSOL Multiphysics**, which has been used to simulate nonlinear heat flux problems on a turning cutting tool (Brito, Carvalho and Lima E Silva, 2015) and simulation of cutting tool temperature during turning (Mourad, Mourad and Abderrahim, 2017).



MSC Marc, ABAQUS and COMSOL are general purpose packages capable of simulating a number of physical phenomena including fluid flow, electrical circuits and magnetism. Deform 3D on the other hand was developed specifically for machining simulations, with particular attention paid to mesh adaptivity. All four packages support user subroutines written in low level languages such as Fortran (Marc, Deform 3D and ABAQUS) and C (COMSOL). Furthermore, all four packages support basic functionality such as three-dimensional problems, initial mesh generation and geometry modelling.

Both MSC Marc and ABAQUS can be controlled externally using Python, however COMSOL only provides Java and MATLAB programming interfaces. There is no evidence to suggest that DEFORM 3D has native support for external control by Python (Scientific Forming Technologies Corporation, 2014).

Of Marc and ABAQUS, ABAQUS has a superior user interface with a more modern design and informative error messages. This study aims to deliver a methodology that requires an end user to interact with the finite element software and so the quality of the user interface must not be ignored. However, the MSC Marc interface, shown in figure 3-9 is satisfactory and has undergone many improvements over recent years.

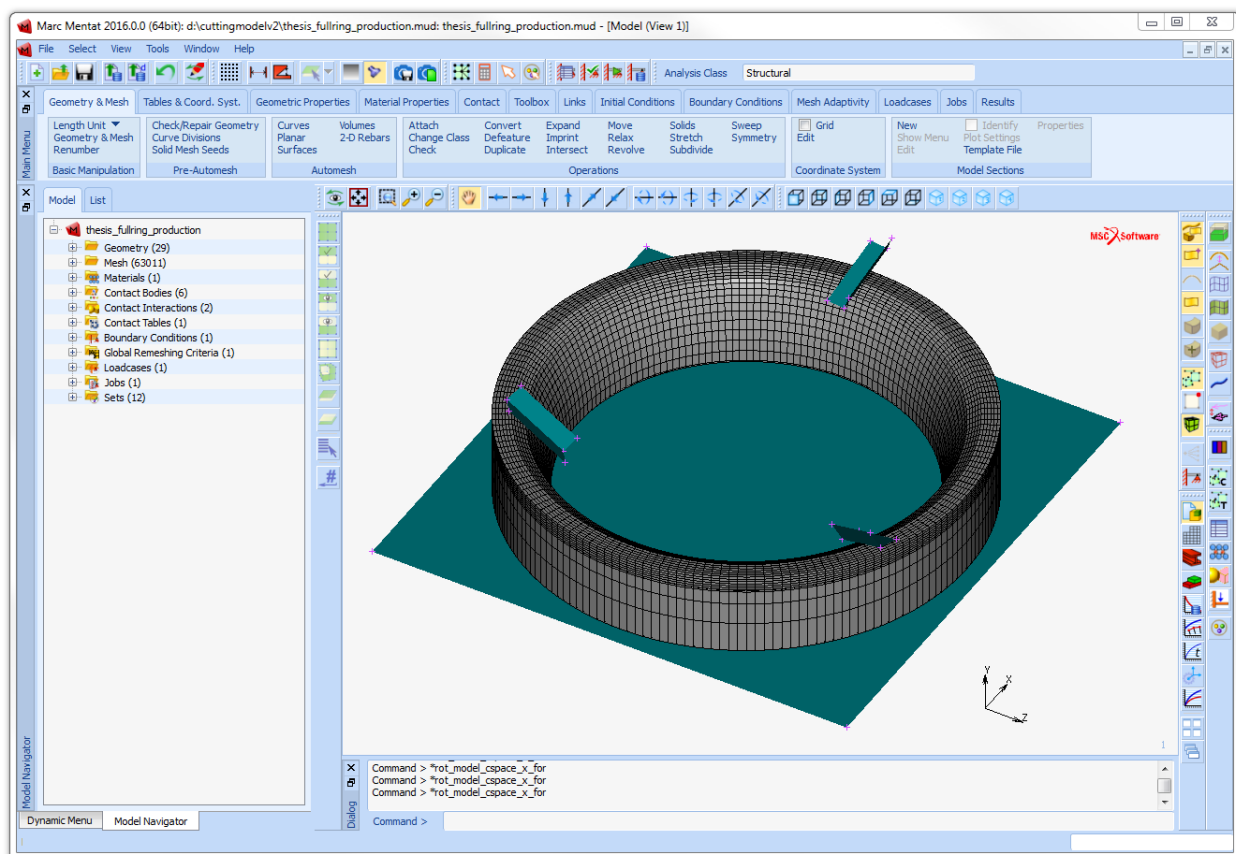


Figure 3-9 – MSC Marc 2016 Interface

Although both Marc and ABAQUS support user subroutines, ABAQUS has some limitations as to what can be overridden by the user in the form of a user subroutine. Most notably ABAQUS offers no documented method to override remeshing during simulation from a user subroutine (Dassault Systèmes, 2014), whereas Marc offers the user subroutine UMAKNET which can be used to implement a custom remeshing algorithm (MSC Software, 2016b). For this reason, and the reasons discussed previously, MSC Marc was selected as a suitable finite element analysis package to build the model in.

### **3.7 Chapter Summary**

This chapter presented the core theory underpinning how non-linear finite element methods work and how they can be used when building finite element simulations. Understanding how such approaches work is key to resolving the many issues that arise during simulation and to help optimise the inputs and make reasonable assumptions about what can be achieved with FEA and more importantly, what cannot be achieved.

This chapter has shown that a great deal of factors must be taken into account in order to verify the accuracy of simulation inputs and outputs. Key steps such as carefully considering the discretisation approach to use to ensure accuracy of the simulation and the importance of simplification in reducing simulation time have been highlighted.

Four very strong non-linear finite element packages were considered for this work. Of these packages, MSC Marc and ABAQUS stand out due their proven applicability to a wide range of problems documented in literature. Ultimately, although ABAQUS has a superior interface and documentation, MSC Marc was selected due to its ability to implement a custom remeshing algorithm through user subroutines.

# Chapter Four – Techniques for Characterising and Modelling Cutting Systems

---

The previous chapter covered the theory and methodology underpinning non-linear finite element analysis. This chapter looks at various experimental techniques available for characterising cutting systems. This chapter also discusses various numerical models for modelling machining phenomena such as friction, wear, plastic flow, etc.

## 4.1 Mechanical Characterisation of the System

The accuracy of any numerical model depends upon the quality of inputs. Accurate characterisation of the system is therefore essential to accurately model the overall machining operation. This section discusses various techniques available to capture cutting system parameters and material mechanical properties.

### 4.1.1 Modulus of Elasticity and Poisson's Ratio

Linear elastic behaviour in a material is governed by the Modulus of Elasticity,  $E$  and Poisson's Ratio  $\nu$ . Tensile testing is commonly used to determine these parameters. During tensile testing the force (and therefore stress) is measured using a load cell and the in-plane displacements (and therefore Poisson's ratio) are measured using an extensometer. Tensile testing typically requires the preparation of large material samples such as those shown in figure 4-1.



*Figure 4-1 – 'Dog bone' tensile testing sample*

Tensile testing works well for homogeneous material samples, however in the case of the AR20 valve seat material studied in this work, it is not economically possible to sinter a dog bone specimen for tensile testing. Furthermore, sintered materials often exhibit different material properties at their surfaces compared to their interiors.

### 4.1.2 Vibrational Properties

Chapter Two discussed the influence of vibration on machining processes. Ideally, vibration should be controlled and minimised to avoid chatter and damage such as chipping. Realistically, vibration will always be present in some form within the machining process. The vibrational characteristics of a system can be used as an input to numerical models, for example by specifying an oscillating displacement boundary condition at some surface or node within the model. Vibration can also be used as a means of validating a numerical model, for example by comparing the dominant vibrational frequencies and amplitudes calculated by the numerical model to those observed through experimentation.

Measurement equipment can be fitted to the machine, its spindle, the workpiece or fixtures. There are various different approaches to capturing the vibrational characteristics of a cutting system, for example:

- Laser Doppler Vibrometry (LDV), which is a technique that allows high speed, non-contact measurements of vibration. LDV can be used to measure the vibration from the surfaces of rotating structures such as machine spindles. LDV is truly non-contact unlike accelerometers and force transducers. LDV is less common, but successful studies using LDV do exist for example that presented by Tatar, Rantatalo and Gren, 2007. Although LDV is non-contact, some surface preparation is required to ensure that the measurement surface is optically smooth. This helps to reduce speckle noise;
- accelerometers, either solid-state or micro-transducers which can be fixed to a surface using adhesives or mechanical fixings. Accelerometers can output vibration data in up to three axes. Huang *et al.*, 2008, use this technique to show that accelerometer data can be used to detect tool breakage in CNC machines. Modern CNC machines often come fitted with accelerometers in key locations to monitor vibration; and,
- dynamometers and force sensors. Suh, Khurjekar and Yang, 2002, used force meters fixed to a workpiece base to record vibrational data at a sample rate of 6 kHz. They processed the captured vibrational data using Fast Fourier Transforms (FFT) to obtain the frequency spectrum for a simple milling operation. The frequency data showed strong peaks at harmonic frequencies responsible for causing chatter.

#### 4.1.3 Plastic Flow Curve Characterisation

During machining of ductile materials such as steel, material entering the process zone is deformed plastically before damage accumulates and shearing of the chip occurs. During plastic deformation, the material undergoes strain rate dependant hardening. Ludwik, 1909, developed a model for this behaviour as represented by equation 4-1, where  $\sigma_Y$  is the materials yield stress,  $\varepsilon_p$  is the equivalent plastic strain,  $C$  is the material's strength index and  $n$  is the materials strain hardening exponent.

$$\sigma_f = \sigma_Y + C \cdot \varepsilon_p^n \quad (4-1)$$

The model has previously been compared to others including Holkmon, Swift, Samanta, Voce and Misiolek, for use in FEM studies describing the behaviour of dog bone steel specimens under uniaxial tensile load. Ludwik's equation was found to give results very close to experimental results (as was Swift's), (Dan *et al.*, 2007).

ASTM details a procedure for measuring the strain hardening constants,  $C$  and  $n$  for metallic sheet materials (ASTM, 2000). Specific dimensions are given for the test sample and all experimental data are obtained through tensile testing. These data are processed using equations given in order to find the strain hardening constants.

#### 4.1.4 Friction

The friction between a cutting surface and workpiece depends on a vast array of factors such as surface roughness, material microstructure, temperature, vibration and material flow stress. Despite the complexity of characterising friction, its effect in numerical modelling is profound. Aside from the obvious contribution of friction to cutting forces, friction plays a core role in determining the character of chip formation (Maranhão and Paulo Davim, 2010). For this reason, any numerical model designed to predict the behaviour of machining should incorporate a representative friction model.

Like wear (and for many of the same reasons as wear), friction is a system property that exists where two bodies in contact move relative to one another. Friction is both dependent and influential on a number of factors in machining as shown by the relationships given in table 4-1 (Boisse, Altan and Luttervelt, 2003), using parameters defined according to the friction schematic given in figure 4-2.

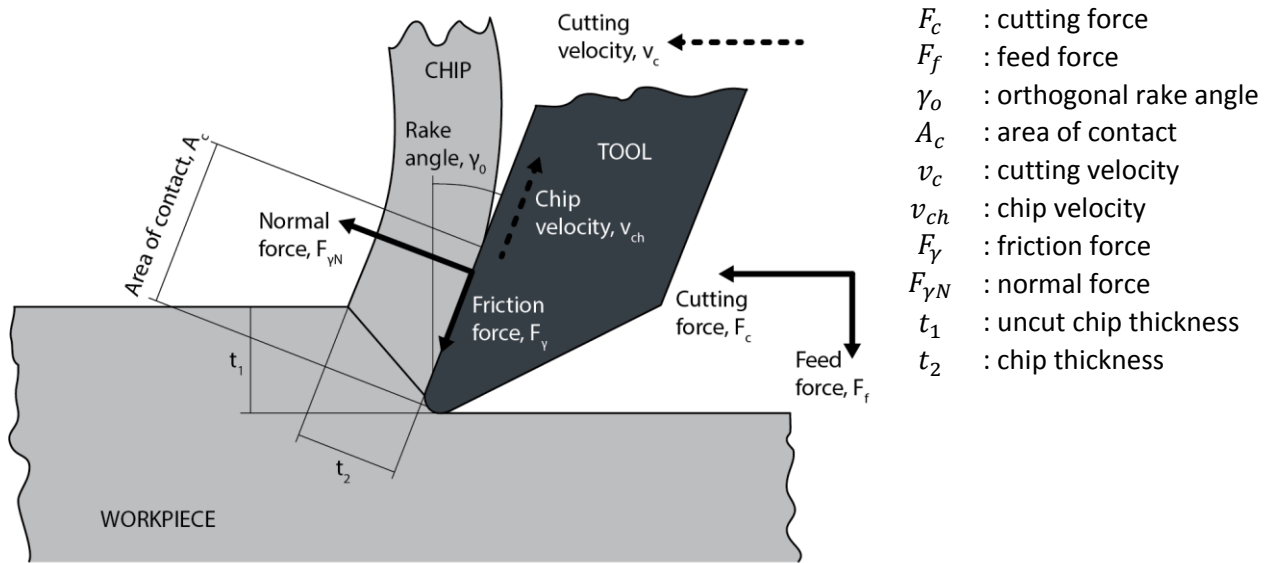


Figure 4-2 – Cutting friction schematic

<b>Friction force</b>	$F_\gamma = F_c \sin \gamma_o + F_f \sin \gamma_o$	(4-2)
<b>Normal force</b>	$F_{\gamma N} = F_c \sin \gamma_o - F_f \sin \gamma_o$	(4-3)
<b>Normal contact stress</b>	$\sigma_f = \frac{F_{\gamma N}}{A_c}$	(4-4)
<b>Shear contact stress</b>	$\tau_f = \frac{F_\gamma}{A_c}$	(4-5)
<b>Mean coefficient of sliding friction</b>	$\mu_f = \frac{\tau_f}{\sigma_f} = \frac{F_\gamma}{F_{\gamma N}}$	(4-6)
<b>Chip thickness compression ratio</b>	$\lambda_h = \frac{t_1}{t_2}$	(4-7)
<b>Frictional heat flux</b>	$q_f = \frac{F_\gamma v_{ch}}{A_c} = \frac{F_\gamma v_c}{\lambda_h A_c}$	(4-8)

Table 4-1 – Friction dependant relationships

To represent friction in a finite element study requires the development of a representative friction model. Özel, 2006, studied the influence of five different friction models on continuous chip cutting of low carbon steel, supported by experimental results. It was shown that higher complexity friction models based on both the measured normal and frictional stresses on the tool rake face are the most accurate, including when used to predict the geometry of chip formation.

Friction between moving bodies in contact can, under certain circumstances, express stick-slip phenomena. Stick-slip contact occurs when the static friction is greater than the kinetic friction. When the applied force is large enough to overcome the static friction, the movement that follows causes the kinetic friction to drop giving rise to a sudden increase in velocity. This process repeats as the bodies move relative to one another and can be a major source of vibration. Leine *et al.*, 1998, propose a practical approach to developing simple stick-slip friction models for finite element studies.

#### 4.1.5 Temperature

The temperatures of regions within a cutting system can be used as both inputs for a numerical study and as validation criteria. Most energy within cutting systems is dissipated through heat. Furthermore heat and temperature affect a wide range of cutting phenomena. These factors make temperature measurements important to understanding a cutting system. There are many approaches to measuring temperature depending on the requirements. Table 4-2 shows various temperature measurement techniques available for both contact and non-contact applications.

Contact		Non-Contact	
Point	Planar	Point	Planar
-Thermocouple -Thermistor -Resistance Temperature Detector -Mercury based -Optical fibre thermography, (Sato, Ueda and Tanaka, 2007).	-Thermoelectric -Bimetallic strip	-Pyrometer	-Thermography (Infrared Thermal Camera)

Table 4-2 – Temperature measurement methods

In this work, the main temperatures of interest are those that develop at the interface between the cutting insert and workpiece. Measurement of temperatures in these areas is complicated by an inability to access the areas or achieve a clear line of sight during cutting.

Typically the cutting system studied in this work is cooled using high pressure coolant, which floods the process zone and completely obscures the line of sight. Non-contact based methods such as thermal cameras cannot accurately measure temperatures obscured by films of water since water heavily attenuates and refracts infrared radiation. For minimum quantity lubricant (MQL) variants of the cutting system, infrared temperature measurement is ideal.

Another common issue faced when trying to measure the temperature at the tool workpiece interface is that the tool is in constant high speed motion. Sutter *et al.*, 2003, overcame this limitation by designing a representative experiment of the cutting system to be measured, in which the sample workpiece material was fired at cutting velocity past a static tool. The process zone was thus easily observed by a thermal camera since the tool was static. The thermal camera used was calibrated using a He-Ne laser which shares a common line of sight with the thermal camera by means of a beam splitter. The images captured in this example gave a good map of temperatures at the tool-chip interface. A short exposure time of 64  $\mu$ s was used to limit the effects of motion blur. This approach to thermal imagery gives superior visual access to the process zone when compared to imagery taken from unmodified machines. However this approach requires careful design and construction of a representative experiment.

Others have applied thermal imagery to dry milling processes without modifying the machining configuration. Lauro *et al.*, 2013, used an infrared camera aimed at the process zone of a simple dry milling operation. Good results were achieved with minimal experimental setup, but the quality of captured data was poor compared to the resolution, frame rate and sensitivity of the He-Ne laser calibrated setup discussed previously.

When thermal imagery cannot be used in circumstances where there is no clear line of sight, direct measurement methods must be used. In some cases, direct measurement methods may be preferred, even where a clear line of sight exists, due to the superior accuracy and measurement speed that is possible.

Embedded or surface mounted thermocouples are commonly used to monitor the temperatures of components in the process zone. When modifying a structure to embed a thermocouple, it is good practice to ensure the modification is represented in any simulations using the data, since the modification will often affect the system.

Data from the transducers must be passed along signal wires to a data recorder. Since wires must always be attached, these sensors can only normally be used on static components. Despite this limitation, the technique is routinely and successfully used to verify numerical studies, either as a component in a static workpiece (Chen *et al.*, 2013), or static tool (Saglam, Yaldiz and Unsacar, 2007).

Werschmoeller and Li, 2010, successfully demonstrated a technique to embed temperature sensors within a purpose built pcBN cutting insert. Ten thin-film thermocouples were layered in a pattern between two pcBN substrates diffusion bonded to one another. The sensors, spaced 0.1 mm apart, were arranged in an 'L' shape with three on an axis parallel to the flank face, six on an axis parallel to the rake face and the final sensor sharing their common axis in the corner. The cutting insert was evaluated for sensitivity and dynamic response, showing  $8.87 \mu V ^\circ C^{-1}$  and a rise time of 150 ns. Experimental studies using the tool showed excellent repeatability and endurance of the diffusion bond. It was found that the temperature gradients are often very steep. In a high speed test, ( $1200 m min^{-1}$   $0.313 mm s^{-1}$  feed rate) a temperature difference of  $50^\circ C$  was observed over a span of 300  $\mu m$ .

Le Coz *et al.*, 2012, were able to successfully embed thermocouples in rotating tools, by using a radio frequency connection to transfer data from the rotating spindle to a static radio frequency receiver. The study successfully acquired temperature data from both drill bits and mills and there appears to be no reason why the same technique could not be applied to cutting inserts. This method showed high repeatability and accuracy. However, implementation of a system such as this would require the design and construction of a bespoke tool holder or heavy modifications to an existing tool holder in order to incorporate a radio frequency transmitter, amplifier and power source. This particular study did not consider balance issues and other changes to the dynamic response of the system.

## 4.2 Modelling Cutting Systems

This section discusses a number of methods available for modelling various cutting system phenomena in numerical finite element simulations.

### 4.2.1 Johnson-Cook Constitutive Model

The Johnson and Cook, 1983, model has played a fundamental role in modelling the complex material response exhibited by metals which are subjected to high strains, high strain rates and high temperatures. The model was initially developed for ballistics research for use in finite element models.

Equation 4-9 gives the original Johnson-Cook constitutive equation in terms of von-Mises flow stress, where:

- $A$  is the material's yield stress;
- $\varepsilon$  is the equivalent plastic strain;
- $B$  and  $n$  describe the effects of strain hardening;
- $C$  is the strain rate constant;
- $\dot{\varepsilon}^*$  is the plastic strain rate and is equal to  $\frac{\dot{\varepsilon}}{\dot{\varepsilon}_0}$ , a dimensionless property where  $\dot{\varepsilon}_0$  is the strain used in the quasi-static tension test to determine the parameters  $A$ ,  $B$  and  $n$ , normally  $1 \text{ s}^{-1}$ ;
- $T^*$  is the homologous temperature, given by the ratio  $\frac{T-T_{ROOM}}{T_{MELT}-T_{ROOM}}$ , where  $T_{ROOM}$  is a given reference temperature, usually room temperature and  $T_{MELT}$  is the material's melting point temperature; and
- $m$  is the thermal softening exponent.

$$\sigma = [A + B\varepsilon^n][1 + C \ln \dot{\varepsilon}^*][1 - T^{*m}] \quad (4-9)$$

The expression in the first set of brackets gives the stress as a function of strain. This component closely follows Ludwik's equation as discussed in section 4.1.3. The second set of brackets modifies the equation to take into account the effects of strain rate. The final set of brackets modifies the equation to take into account the effects of temperature. This part is also known as the thermal softening function,  $K_T$ .

The Johnson-Cook constitutive equation is frequently referred to in literature that involves numerical modelling of materials. The equation naturally lends itself well to numerical modelling since within finite element simulations, the parameters,  $\varepsilon$ ,  $\dot{\varepsilon}$  and  $T$  are readily available for each element. Modified versions of the Johnson-Cook equation are also ubiquitous in literature, especially where the model is used to predict damage due to the way the Johnson-Cook model modifies the material response of individual elements as they are worked.

Thanks to its popularity, the Johnson-Cook model has benefitted from considerable validation from the scientific community. Umbrello, M'Saoubi and Outeiro, 2007, studied five different sets of Johnson-Cook parameters as used by other researchers, for their effect on the quality of finite element studies. An individual finite element model simulating orthogonal cutting was created from each set. The outputs of the finite element models were verified using experimental results. The investigation suggests that the choice of parameters has a very large effect on the results produced by finite element studies, including forces, temperatures, chip morphology and residual stresses. Only one of the five parameter sets was found to give representative results for the orthogonal machining simulation within the study when compared to experimental data. The inference from the conclusion is that, although some parameters may



work in a particular zone (as they have done for the researchers from which they were taken) they are not guaranteed to be accurate over a wider range of conditions and can often be wildly inaccurate. However, carefully selected parameters for the Johnson-Cook constitutive equation when used in finite element studies can give very accurate results in specific circumstances.

Table 4-3 gives a summary of Johnson-Cook parameters developed by *Tounsi et al., 2002* and used by *Guerra Silva et al., 2015* to create a material model for sintered AISI 316L.

A (MPa)	B (MPa)	C	n	m
514	514	0.042	0.508	0.533

*Table 4-3 – Johnson-Cook parameters for AISI 316L steel developed by Tounsi et al., 2002*

#### 4.2.2 Modelling Damage

Modelling damage is an essential component of machining simulations. Damage in machining occurs in both the tool and workpiece. In this study, damage in the tool is of particular interest. However, modelling damage in the workpiece is also an important factor, since damage can affect cutting forces and vibration. Modelling damage is also essential if material removal from a workpiece is to be modelled in terms of chip shearing and separation.

Plasticity can always be explained in terms of damage. Plasticity is expressed mainly under extreme loads and results in permanent changes to the materials shape and strength. On the nano-scale, plasticity is the result of dislocations, void formation, cracking and tearing within the material. In finite element simulations, many models are not run to failure and so simple plasticity models such as the Johnson-Cook model are both accurate and computationally efficient. If left to run indefinitely, a model governed only by the Johnson-Cook equation would show plasticity forever and never fracture.

Plasticity always requires damage, but damage does not necessarily lead to plasticity. Many models such as the Gurson and Cockcroft-Latham models calculate damage by predicting void formation and reducing material strength accordingly. There is therefore an overlap between damage accumulation modelled by damage models which results in plasticity-like effects, and plasticity modelled by the Johnson-Cook equation. This distinction separates this section on damage models and section 4.2.1 discussing the Johnson-Cook model. Care must be taken not to model the same physical phenomena twice.

The simplest criteria for modelling the effects of damage could take the form of a maximum principle stress or maximum Von-Mises stress criteria. When an element passes a given threshold it is simply deactivated. Numerically speaking, this means that the stiffness of the element would be set to zero. Whilst this approach is very easy to implement, it requires relatively high element densities in the areas where damage modelling is required. This approach can work very well in ceramics and may be appropriate for modelling failure of the pcBN cutting insert.

The Johnson-Cook model is commonly used as a template for damage modelling around which damage criteria or separation criteria can be based. In finite element simulations, these criteria might disable or weaken any elements that have passed a certain plastic strain, plastic strain rate or plastic stress. This is useful because the customisation can be highly representative of the specific material based on experimental results. Also it guarantees that there is no overlap between plasticity from the J-C model and fundamental damage calculated by models such as Gurson's.

Shams and Mashayekhi, 2012, looked at creating to develop a damage model for orthogonal cutting FE simulations that aims to be both accurate and independent of element size. Because the Johnson-Cook constitutive equation alone cannot be used in finite element simulations to determine the point of material separation, they focused on developing a separation criteria which showed good agreement with experimental results. However, their model was only developed for orthogonal (2D) cutting.

The Gurson, 1977, model estimates damage in porous plastic materials by predicting the nucleation and growth of microscopic voids and their effect on overall material strength. The more recent adaptation of the model proposed by Tvergaard and Needleman, 1995, is preferred in finite element software such as Marc and Deform 3D. The model progressively weakens elements until a virtual void fraction reaches a critical limit after which the element is disabled. Xie, Bayoumi and Zbib, 1998, used the Gurson model to predict separation in a finite element simulation of metal cutting which produced excellent results, especially with respect to predicting chip geometry. As with many finite element implementations that model was 2D, however the Gurson model is available for 3D models in Marc.

The Lemaitre, 1985, model was developed specifically for ductile damage in ferrous materials undergoing manufacturing processes. The Lemaitre model calls upon on thermodynamic concepts to predict damage and is sensitive to temperature (lacking in Gurson's model) which could be ideal for the high thermal gradients in machining. The model gradually weakens elements until failure, as opposed to abrupt deactivation. This model is also supported by Marc.

Vaz *et al.*, 2007, used the Lemaitre damage model in a 3D machining simulation. Those results are of particular interest because the model was used to predict the nucleation and development of shear within the shear zone in machining. This was then compared to a failure model based on a plastic strain criteria. The two were different with the Lemaitre model predicting crack nucleation from the top of the shear chip and the plastic strain criteria from the base. Unfortunately, the model was not compared to experimental results.

Hambli, 2001, compared finite element implementations of the Lemaitre and Gurson models used for sheet metal blanking and found that the Lemaitre model gave superior results.

#### **4.2.3 Modelling Vibration**

Dynamic instabilities in the system can be modelled in a number of ways. Perhaps the most obvious way is to accurately model the masses and eccentricities of all moving parts and allow the FEM software to resolve vibrations. However, this approach would require precise measurements of every moving part within the system. Such an approach would require potentially hundreds of measurements throughout the system and the result would likely suffer heavily from cumulative measurement error and numerical approximation (rounding) errors in the FEM software.

Alternatively, the vibrational characteristics of the system can be measured at the tool holder and workpiece fixture and input into the simulation as a reciprocating waveform which drives a fixed displacement boundary condition. To achieve this, the vibration or displacement amplitude against time for the machine and fixture should be captured at sufficient resolution to feed into the simulation.

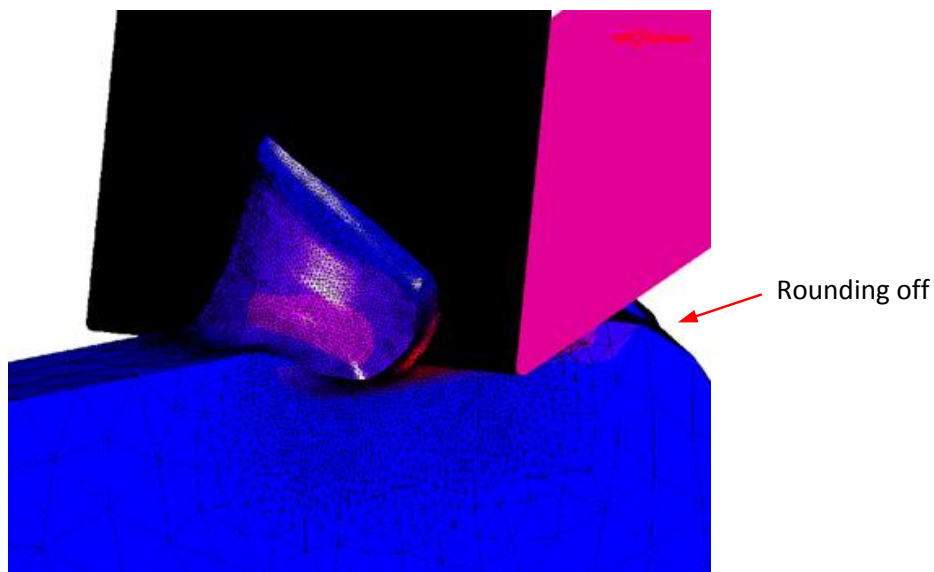
### 4.3 Numerical Model

A preliminary 3D test simulation was developed in MSC Marc to better understand the performance of Marc and become familiar with the simulation challenges. The simulation was configured to cut a thin layer of material from the top surface of a rectangular mesh. The cutter in the model was defined as rigid. Typical Young's Modulus and Poisson's ratio values for steel were used for the mesh material. An adaptive remeshing criterion was added to refine the mesh in proximity to the cutter.

The simulation was run on computer hardware according to the following specification:

- 2 x 3.1GHz Intel Xeon E5-2687W Processors (8 physical cores each, hyper-threading disabled);
- 256 GB RAM;
- 500 GB Solid state hard drive;
- 1 x NVIDIA Quadro 6000 GPU; and
- 1 x NVIDIA Tesla C2075 GPU (for GPU processing).

Figure 4-3 shows the state of the simulation at the point of failure after three days of processing. As the figure shows, hardly any significant progress has been made. Furthermore, the mesh suffers a number of degeneracies, such as rounding off of the mesh on the right hand side under the cutter. Although this model is far from optimised, it is clear that multiple valve seat cutting passes, with multiple cutters is completely impossible without extensive simplifications.



*Figure 4-3 – Preliminary oblique cutting model developed in Marc 2013*

Progression of the model is limited, mainly due to problems during remeshing. The host computer has 16 physical cores (spread across two physical chips) as well as GPU processing capability. Many of the solvers available for Marc offer parallel processing which takes advantage of all available cores (including graphics processors). However, Marc does not include support for multiple processors during remeshing. Figure 4-4 shows the load sharing pattern followed by Marc during the different phases of simulation.

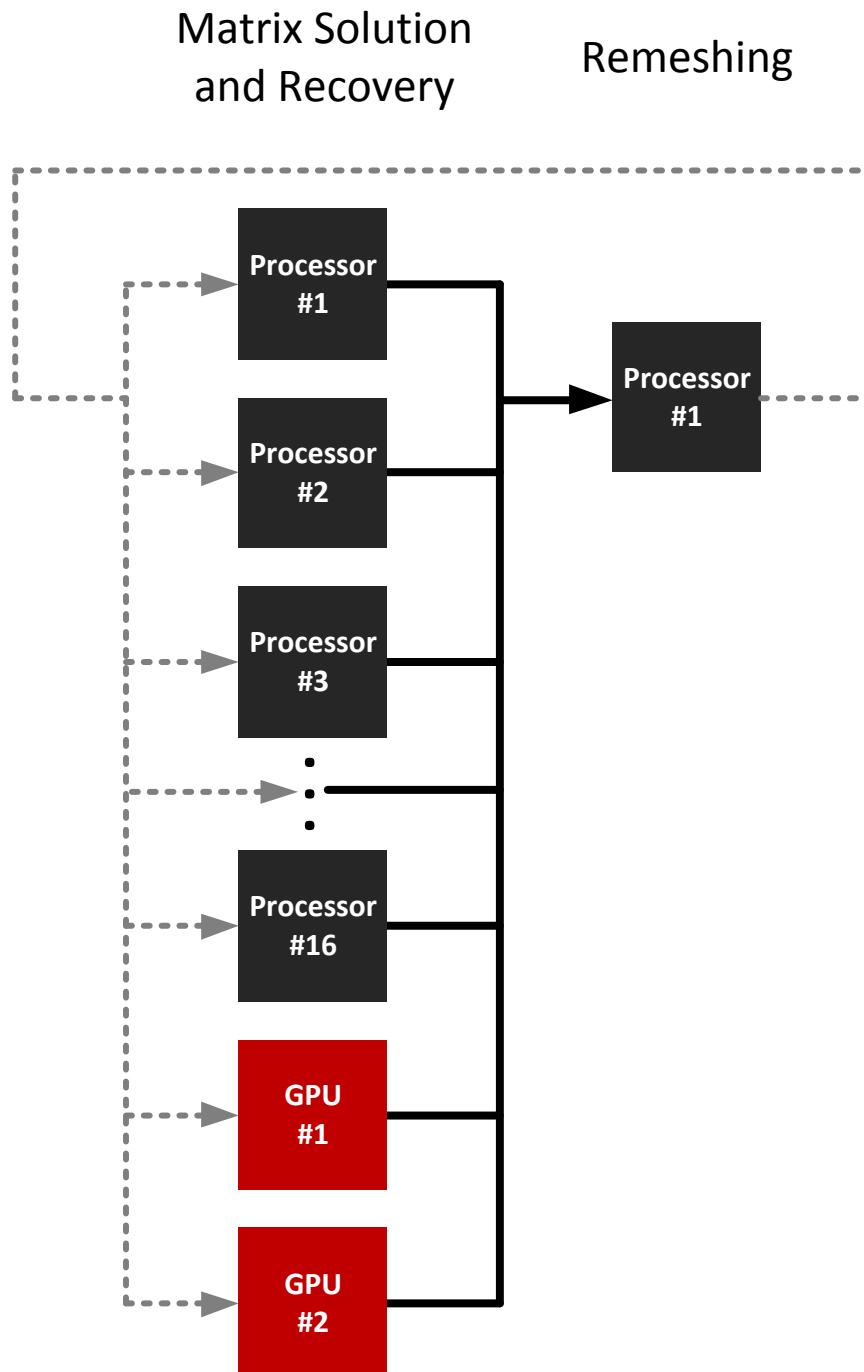


Figure 4-4 – Processor utilisation during solving and remeshing

Clearly the remeshing stage is a substantial bottleneck for simulations requiring it, such as machining in which it is essential due to the large plastic deformations which occur. Optimisation priority must therefore be given to reducing the dependency of the simulation on remeshing. Some widely used techniques include:

- modelling regions of a single body which could potentially undergo plastic deformation as a separate contact body to those regions which will only deform elastically. The two separate bodies are then 'glued' to one another and remeshing is only applied to the most deformed body. This saves some overhead as it reduces the volume through which the remesher is required to generate mesh. Si, 2015, show that for three different 3D tetrahedral mesh generators, as geometric complexity increases so does the time required to generate a 3D mesh. By removing subsections of a mesh that aren't involved in the phenomenon under test, the geometric complexity can be reduced and therefore so can computation time;
- sacrifice element aspect ratio and mesh quality (accuracy) for improved performance, by meshing less frequently;
- split parts of the model into domains with fixed mesh interfaces which can be remeshed in parallel processes. Yagawa and Shioya, 1993, show that this technique (also known as 'domain decomposition') can be used to dramatically reduce computation time at the cost of slightly increased memory demand. In their example a problem executing in 99,684 seconds was reduced to just 4,496 seconds using 105 subdomains and 26 cores. Noor, 1988, review other ways in which parallel processing can be used to greatly accelerate finite element structural analysis; and
- only remeshing a sub mesh of the parent mesh based on its proximity to a cutter. This technique is commonly known as 'local remeshing' (as opposed to 'global remeshing') and is routinely employed to remove global remeshing steps which are costly, unnecessary and vulnerable to introducing interpolation error (Zheng *et al.*, 2016).

#### **4.4 Chapter Summary**

The objectives of this work call for the creation of a numerical model capable of simulating multi-angle valve seat cutting over many increments to a high degree of accuracy and with a minimum of computational and experimental effort. Chapter Two revealed the complexity of the cutting process zone and Chapter Three set out the fundamental numerical modelling techniques necessary for modelling non-linear systems.

This chapter has presented a review of various techniques developed in literature aimed at characterising and modelling cutting systems. It has also shown how others have integrated models designed to simulate machining in to finite element studies with varying degrees of success.

Although many of the methods presented in this chapter have been proven in literature to deliver reasonable results, many require complex experiments to capture the necessary input data. There are few examples in literature of models that incorporate all of the physics of cutting. It is clear from the findings of this chapter that attempting to build a composite model that incorporates every necessary process zone model presented in literature, to the high degree of accuracy necessary to support multiple cutting passes, would be practically impossible. Furthermore, the experimental burden required to support each model would be prohibitive.

Despite these issues, this chapter has provided good insight into the types of phenomena which must be modelled, albeit through some other more simplified means.

# Chapter Five – Experimental Analysis of pcBN Cutting Inserts

---

The literature review for this work presented a number of mechanisms that could potentially damage polycrystalline cubic boron nitride (pcBN) cutting inserts during multi-angle valve seat machining, leaving them vulnerable to chipping. This chapter aims to analyse a selection of cutting inserts to determine if any manufacturer defects or any of the damage mechanisms reported in literature were present.

The pcBN inserts studied were a selection of new, chipped and worn inserts collected from Ford's Sigma 1.6 Engine production line in Craiova, Romania. The inserts were supplied without history and with varying degrees of wear and damage ranging from visually perfect, to worn and chipped along some edges.

The inserts were categorised into three types of geometry, as shown in figure 5-1, referred to from left-to-right as, *hex*, *tri* and *bar* respectively. These specific types of inserts all have a negative rake face.



*Figure 5-1 – Cutting inserts*

Only tri and hex inserts are used during valve seat machining on the production line referred to in the case study for this work. The case study suggests that hex inserts are generally more likely to fail. For these reasons, hex inserts form the main area of interest for this investigation.

Hex inserts are approximately 2.49 mm in height and measure 6 mm between parallel edges; similarly, tri inserts are approximately 2.42 mm in height and measure 7.93 mm from corner to edge as shown in figures 5-2 and 5-3 respectively. Both inserts have identical cutting edge dimensions.

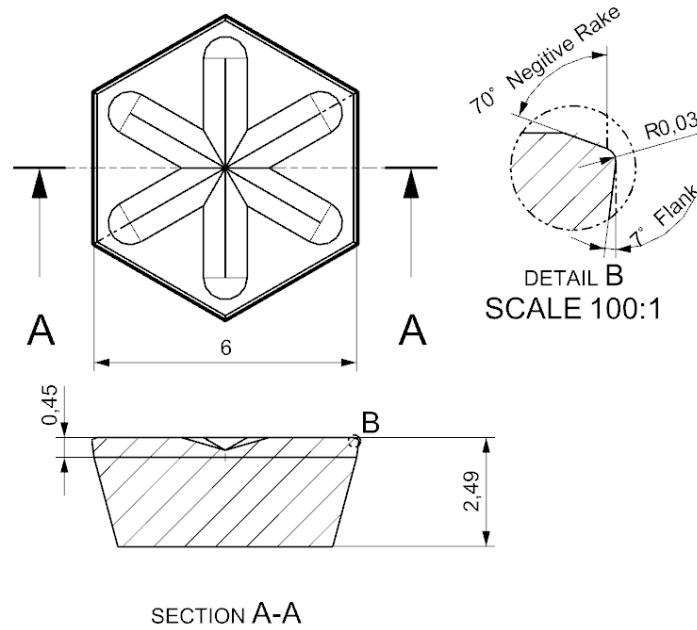


Figure 5-2 – Hex insert dimensions

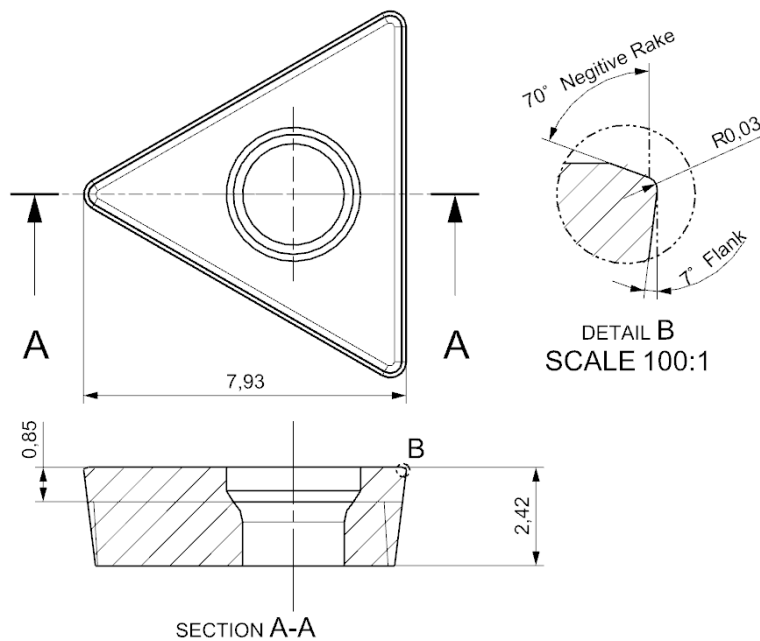


Figure 5-3 – Tri insert dimensions

## 5.1 Schedule of Specimens

Table 5-1 gives a schedule of the specimens inspected, with a description of their visual condition listed in the final column. For the hex inserts, the code found on the part refers to the last three digits of the supplier part code.

Ref	Type	Code on Part	Visual condition
1	Hex	410	Slight chip on one edge
2	Hex	485	Slight wear visible on one edge, chip on one edge
3	Hex	485	Slight wear visible on one edge
4	Hex	485	Slight wear visible on one edge
5	Hex	485	Chip visible on two edges, wear visible on two edges
6	Hex	485	Perfect
7	Hex	485	Perfect
8	Hex	485	Perfect
9	Hex	485	Perfect
10	Hex	485	Slight wear visible on one edge
11	Hex	485	Perfect
12	Hex	485	Slight chip on one edge
13	Hex	485	Perfect
14	Hex	485	Large chip on one edge, wear on three edges
15	Hex	485	Chip visible on one edge
16	Hex	810	Slight wear visible on three edges, Chip on point of corner
17	Hex	810	Heavy wear on one edge, heavy wear and chipping on one edge, slight wear on three edges
18	Hex	810	Wear on four edges, very large chip on one edge
19	Hex	810	Wear on five edges
20	Hex	810	Slight wear on one edge, chipping on one edge
21	Hex	835	Slight wear on all six edges
22	Tri		Perfect
23	Tri		Perfect
24	Tri		Perfect
25	Tri		Wear in one corner
26	Tri		Slight wear on one edge
27	Tri		Perfect
28	Tri		Wear at two corners
29	Tri		Wear on one edge
30	Tri		Perfect
31	Tri		Wear on one edge
32	Bar	30185065 L2	Perfect
33	Bar	30185065 L6 L5 L4 L3 L2 L1	Perfect
34	Bar	30185065 L3 L2 L1	Perfect
35	Bar	30185066 L4 L3 L2 L1	Perfect
36	Bar	30185066 L4 L3 L2 L1	Slight wear on edge
37	Bar	30185065 L5 L4 L3 L2 L1	Perfect
38	Bar	30185065 L1	Perfect
39	Bar	30185073 L2	Perfect
40	Bar	30185066	Perfect
41	Bar	30033462 L2 L1	Slight chip on edge
42	Bar	30266906 L1	Missing tool tip
43	Bar	30185068 L1	Perfect
44	Bar	30185068	Perfect
45	Bar	30185068	Perfect

*Table 5-1 – Cutting insert defects found on examination of samples*

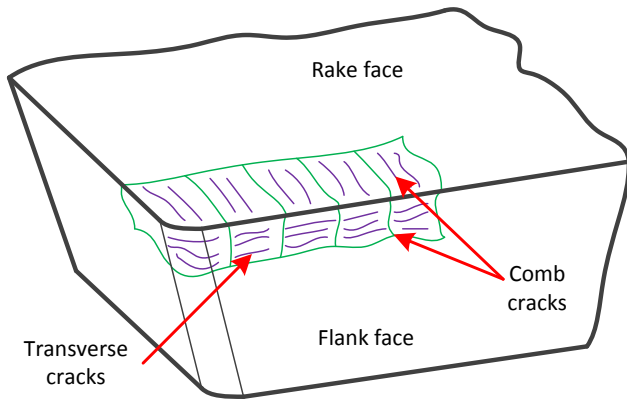


In previous chapters, the weight of literature reviewed suggested that the premature failure of pcBN tools when used in valve seat cutting operations occurs as a result of pcBN's vulnerability to vibration, primarily due to its poor fracture toughness. Dynamic instability is inherent in the valve seat machining process as multiple independent surfaces at different angles are cut simultaneously.

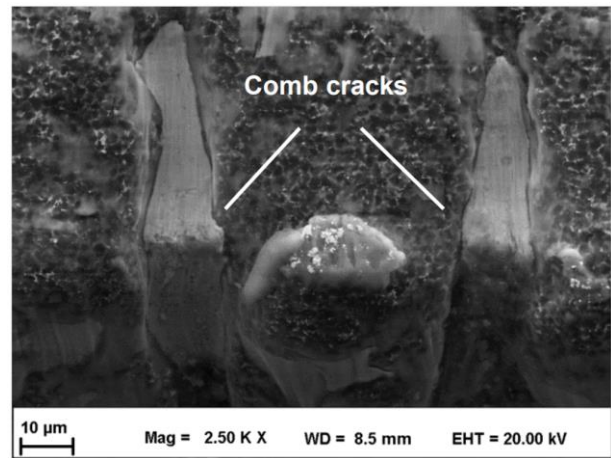
However, the literature also suggests a number of other possible mechanisms that could explain the failure of pcBN tools during high speed machining. In order to have confidence in the dynamic imbalance explanation, these other potential explanations must be ruled out.

- **Tool geometry (cutting radius):** It is known that sharp cutting radii on pcBN inserts increases the amplitude of vibration and thus the probability of chipping (Lacerda and Siqueira, 2012). It is sometimes desirable therefore to modify the cutting radius and chamfer using precision grinding techniques, brushing or magnetic field assisted finishing (Ventura, Köhler and Denkena, 2013). pcBN tools are generally supplied with a negative rake face and can be used with a cutting radius anywhere between 0 $\mu$ m (sharp) through 30 $\mu$ m (sharp, with edge preparation) to 200 $\mu$ m (worn). The cutting radius is defined as the radius between the rake and flank faces of the tool. It can be measured using optical microscopy combined with calibrated digital image measurement software.
- **Flank and crater wear:** Some researchers have identified an erratic relationship between cutting speed and wear rate when using pcBN tools that results in flank and crater wear (Rocha *et al.*, 2004).  
The flank of a tool is the edge leading away from the cutting edge radius which is not in contact with the chip. Flank wear causes a thinning of the tool between the rake and flank faces thus weakening the cutting edge and increasing the probability that volume will be chipped away. Crater wear is characterised by pitting and chipping that builds up to form a crater on the rake face. It is typically visible on images obtained under optical microscopy or scanning electron microscopy (SEM). Physical mechanisms responsible for flank and crater wear include chemical, adhesion, solubility and melt between the tool material (pcBN) and chip (sintered steel workpiece material). The rate of flank wear can be measured by analysing and comparing the surface roughness of the flank face at intervals before and during machining. Evidence of chemical interaction, diffusion and melt can be obtained by Energy-dispersive X-ray spectroscopy (EDS).
- **Manufacturing Defects:** Inhomogeneous material, poor fusion, crystal clumping, voids, cracks and chips are several types of defect that can occur in sintered materials such as pcBN cutting inserts. Sufficiently large defects (>1 $\mu$ m) within the material may be detected using X-Ray tomography. Smaller defects within the material may only be detected by slicing the cutting insert at intervals and inspecting using SEM. Small pre-existing chips on the cutting edges of the inserts that result from the manufacturing process e.g. flash or dirt in the mould, can lead to premature chipping during machining. Such chips can be detected on new inserts using optical microscopy or SEM.
- **Comb-cracking due to high temperature:** Cemented carbides such as pcBN with a tungsten carbide binder material are particularly vulnerable to a phenomenon referred to as comb cracking. Comb cracks are not to be confused with thermal shock cracks. Thermal shock cracks develop spontaneously by a sudden and rapid change in temperature, for example, after sudden exposure to coolant during turning. Comb cracks however develop after exposure to several periodic temperature induced stress cycles and thus develop more slowly (Klocke, 2011). Figure 5-4, A) shows the orientation of comb cracks (in green) with respect to the rake and flank faces of the tool.

The lines in purple show the orientation of transverse cracking which is a similar phenomenon that forms as a result of pressure-induced stress cycles from the cutting force. B) shows an SEM micrograph of comb cracks found on a pcBN cutting insert (Malakizadi, Sadik and Nyborg, 2013).



A) Comb crack diagram



B) SEM micrograph of comb cracks  
(Malakizadi, Sadik and Nyborg, 2013)

Figure 5-4 – Comb and transverse cracking diagram

## 5.2 Visual Inspection using Optical Microscopy

Visual inspection of the inserts was performed, primarily using an Olympus BX60M microscope. Further visual inspection and digital measurement was performed using a Nikon Optiphot microscope using 5 X, 10 X and 20 X optical zoom. Images were captured using a GXCAM-5 ISH500 5.0MP camera directly attached to the microscope. GT Vision GXCapture version 8.0 was used to process the captured images.

### 5.2.1 Measurement of Corner Radius

Both hex and tri inserts were inspected and found to have undergone edge preparation. In almost all cases they had unworn corner radius of approximately 30μm. Figure 5-5 shows a corner radius measurement.

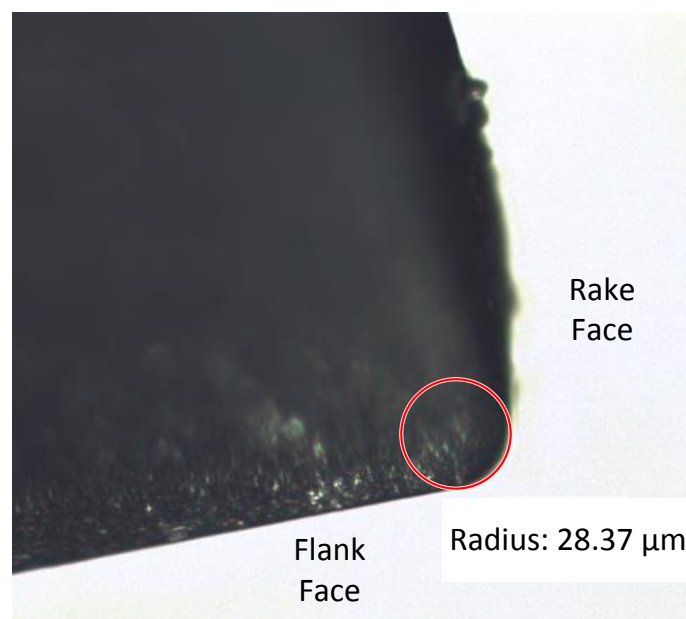


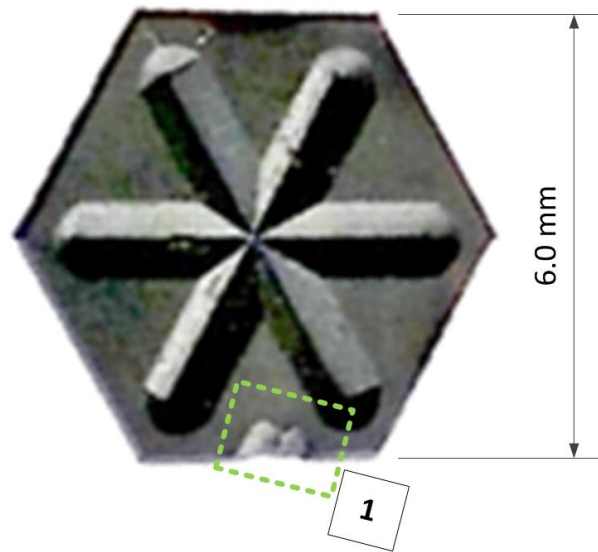
Figure 5-5 – Corner radius measurement

Six measurements were made across four hex inserts yielding a mean radius of  $29.77\mu\text{m}$  and a further two measurements were made on two tri tools giving a mean radius of  $28.12\mu\text{m}$ .

A radius of  $60\mu\text{m}$  was suggested in literature to be the optimum radius for the minimisation of vibration when cutting valve seats with pcBN tools when compared against radii of  $0\mu\text{m}$ ,  $30\mu\text{m}$  and  $200\mu\text{m}$  (Lacerda and Siqueira, 2012).

### 5.2.2 Inspection of Large Chip on Insert 14

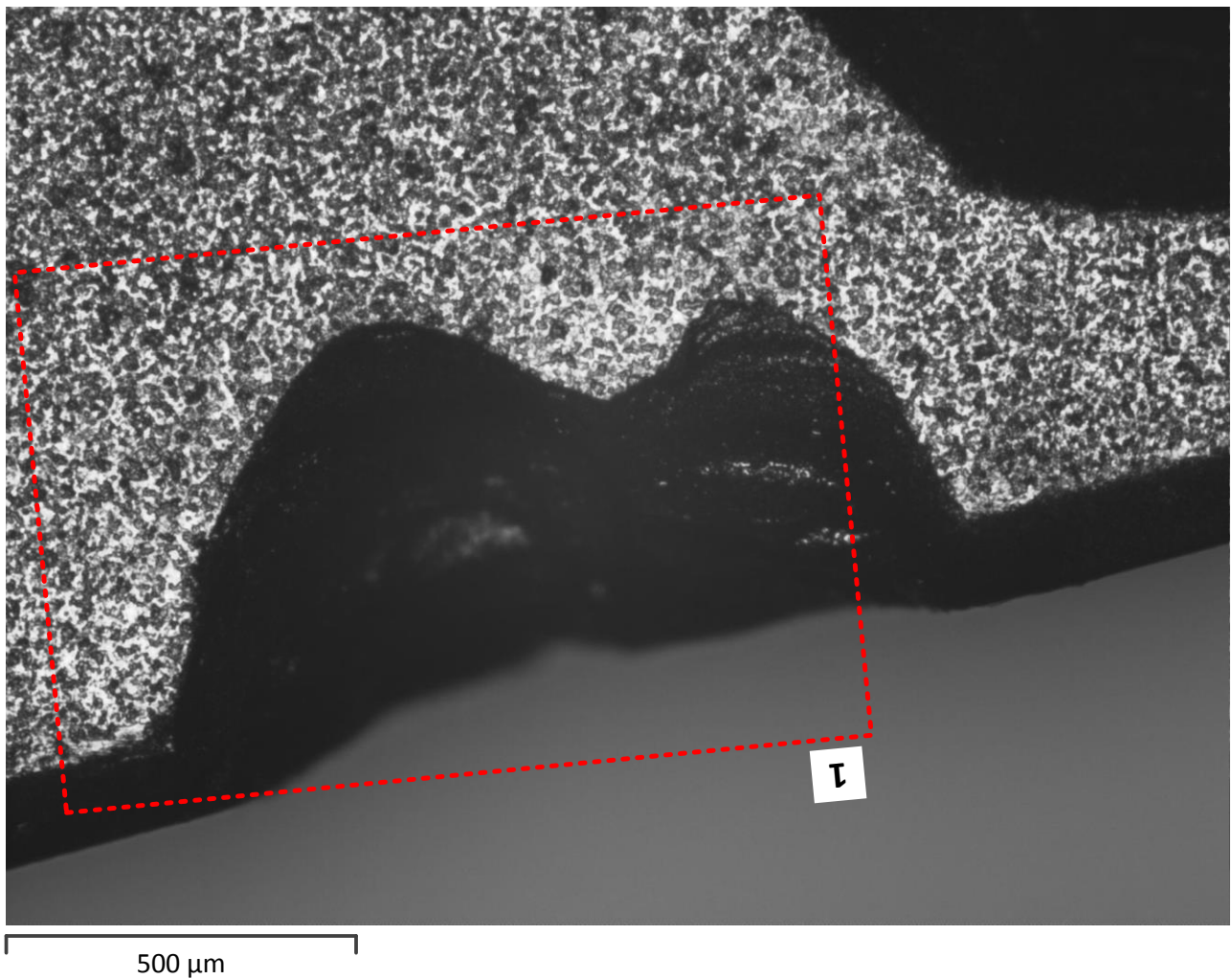
Insert 14 had one of the largest chips observed. Figure 5-6 shows the top-down view of insert 14 with the position and relative size of the chip visible on the South edge. The North, NE and SE edges were also heavily worn.



*Figure 5-6 – Insert 14*

Figure 5-7 shows a large chip on the edge of insert 14 from region 1 in figure 5-6. The focal plane was set to the top surface of the insert. The chip boundary on the top surface appeared to be clean and there are no cracks leading away from the broken edge.

The dark region in the upper right hand corner of the figure is the chip breaker cavity. Six such cavities exist and radiate from the centre of the insert to the corners between edges.

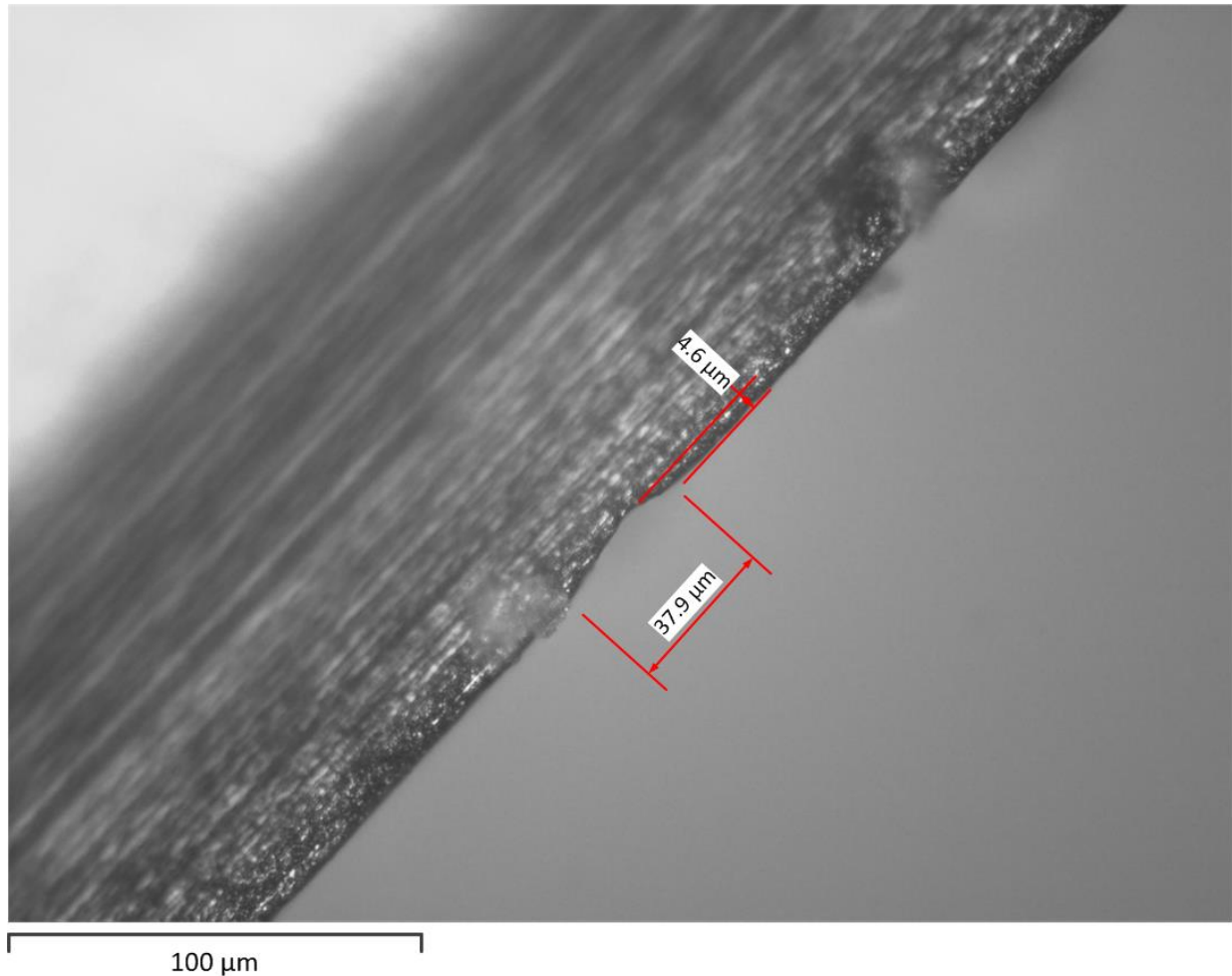


*Figure 5-7 – Large chip on insert 14 viewed at 5 X optical zoom*

No evidence of voids within the material on the exposed interior surface was observed and there was no obvious nucleation point for the crack. The cutting edge and radius were unworn, suggesting that the edge failed early during its duty cycle. However, several other edges were heavily worn, but showed no evidence of chipping. It would be reasonable to assume that before this chip occurred, the bulk material of the insert would have been exposed to multiple heating cycles during use on other edges.

### 5.2.3 Inspection of Small Chip on Insert 25

Figure 5-8 shows a chipped edge of insert 24 which was initially indexed as visually perfect. For this image, magnification was set to 10 X optical zoom and the focal plane was aligned with the corner radius between the rake and flank faces.

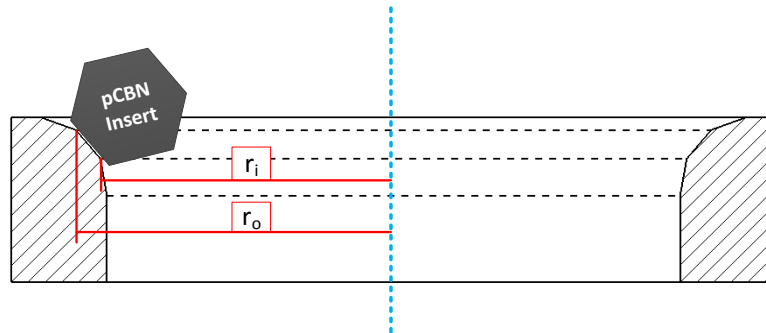


*Figure 5-8 – Small chip on insert 24 viewed at 10 X optical zoom*

In this image, the light out-of-focus region in the top left is the top surface of the insert. The dark band with parallel striations is the negative rake face of the insert. A small chip is visible at approximately image centre. The depth of the chip at its deepest point is approximately 4.6 μm.

The chip depth is sufficiently large to leave a raised band on the finished part which would cause the valve seat to fail during a cylinder leak test. The surface finish roughness specification for valve contact faces on the seat is 3.2 μm.

The chip character and dimensions initially appeared consistent with flank and crater wear. However, the chip was the only one of its kind along the edge. If the edge was exposed to even cutting conditions along its length, flank and crater wear would be evident across the entire edge. However, during valve seat machining, wear is not even across the edge. Figure 5-9 shows a section view of the cutting operation, where the blue line is the centre of rotation, and the radii of the inside and outside edges of one of the seat's angles are marked in red.



*Figure 5-9 – Cutting radius cross section diagram*

From the diagram, it is clear to see that since  $\omega$  is constant at all points on the insert,  $V_o > V_i$  and therefore the cutting conditions cannot be said to be constant along the length of the cutting insert edge. As discussed earlier, an erratic relationship exists between cutting velocity and the rate of flank and crater wear when using pcBN tools.

Since the area around the chip appeared unworn, it is also possible that the chip may have resulted from a manufacturing defect, a common cause of premature chipping. DeVries, 1992, suggests that once chipping on this scale occurs, catastrophic failure of the insert is inevitable.

Optical microscopy inspection has allowed measurement of the cutting radius of the hex and tri inserts, both of which were found to have radii inconsistent with the optimum edge preparation radii for the minimisation of vibration as suggested in literature. Optical microscopy provided a better visualisation of large scale chipping along the edges of some inserts and also revealed the presence of small-scale damage that was previously invisible to the naked eye.

The large-scale chipping observed on insert 14 may or may not have been nucleated by manufacturing defects within the insert, but was too large to be the result of flank and crater wear alone. Closer inspection using SEM would be required to look for further evidence on and around the exposed chip surface.

The small-scale damage on insert 24 was likely to be flank and crater wear (although the physical mechanism responsible was not clear), but may also be due to a manufacturing defect.



### 5.3 X-Ray Computed Tomography

pcBN is known to have a very poor fracture toughness, and therefore when exposed to excessive vibrational loads, pcBN can be expected to fail. In the previous section, it was suggested that the large scale chip shown in figure 5-7 may be consistent with manufacturing defects within the insert. The defects may take the form of internal voids or cracks.

X-Ray Computed Tomography can be used to inspect the cutting inserts for internal defects. A Metris XT H 160Xi X-Ray imager was used in conjunction with VGStudio MAX 2.2 software to produce a tomogram of insert 14. Figure 5-10 shows the position of the insert and polymer stage relative to the X-Ray source.

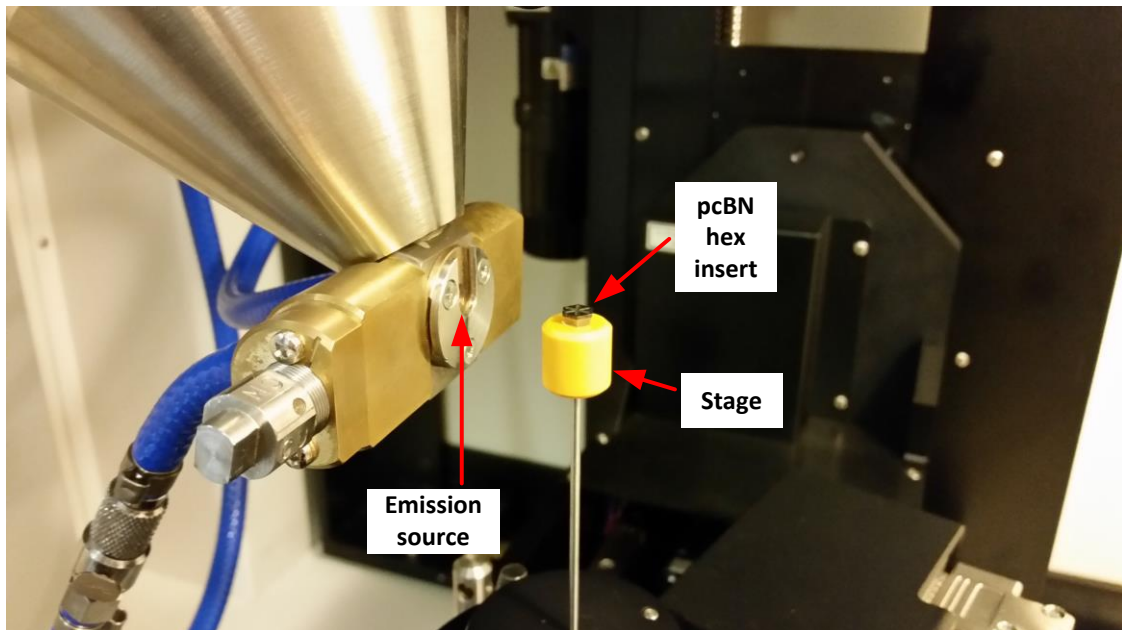


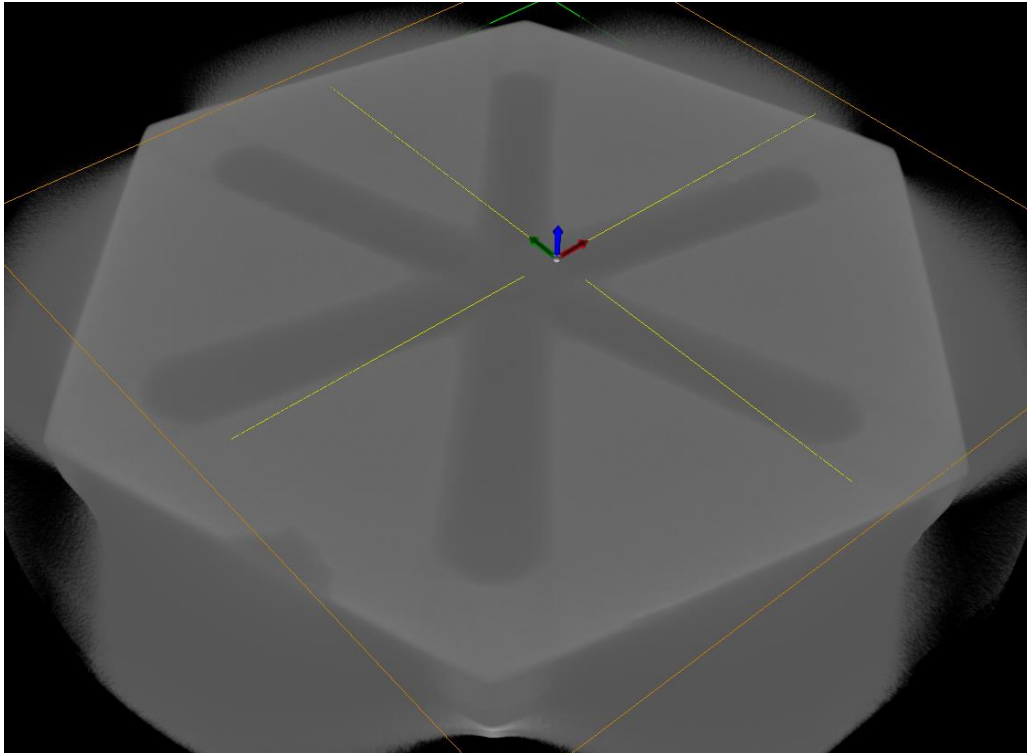
Figure 5-10 – X-Ray stage

Figure 5-11 shows an X-Ray image captured through the side-on view of insert 14. The dark tapered layer in the image centre is the tungsten carbide substrate on which the lighter layer (the pcBN layer) is bonded. The contrast between the two layers was consistent with their densities at  $3.45\text{kg m}^{-3}$  and  $15.63\text{kg m}^{-3}$  for CBN and tungsten carbide respectively.



Figure 5-11 – X-Ray side view of insert 14

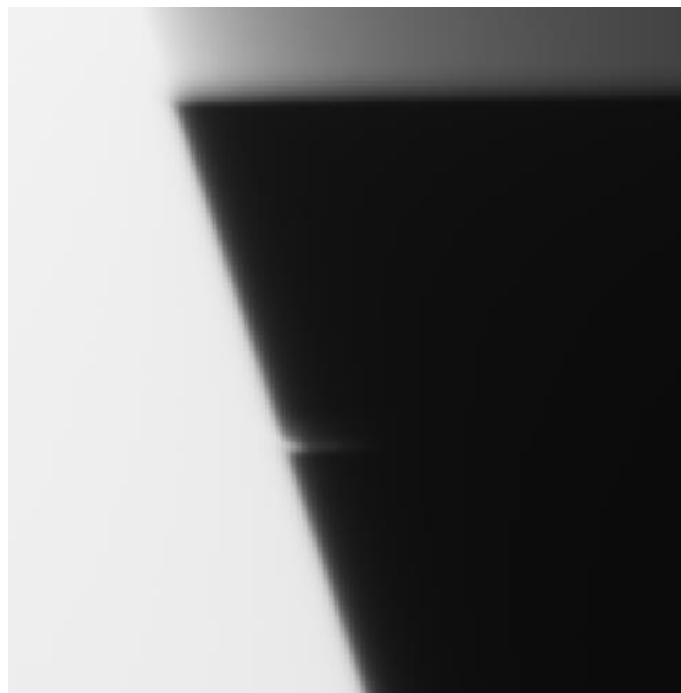
Figure 5-12 shows the computed tomogram of insert 14 as viewed in VG studio.



*Figure 5-12 – Computed tomogram of insert 14*

Thorough inspection of the body of the insert, in particular the region around the chip, revealed no evidence of internal defects such as voids and cracks larger than 1  $\mu\text{m}$ . However, unwanted noise (clearly visible around the outside of the insert in the figure) may have been hiding internal detail.

Figure 5-13 below shows a close up view of the left hand side of the insert first shown in figure 5-11 previously.



*Figure 5-13 – X-Ray side-on view of insert 14 (close up)*



The image shows what appeared to be a horizontal crack extending into the insert. It is possible that this was evidence of a crack or a separation of two bonded layers of the tungsten carbide substrate. It may also be flair from the sharp edge. In either case, the anomaly was located far away from the pcBN layer and is therefore not significant. Furthermore, the anomaly was not visible in the 3D tomogram, nor was it visible under optical microscopy, suggesting it most likely just an artefact in this single image.

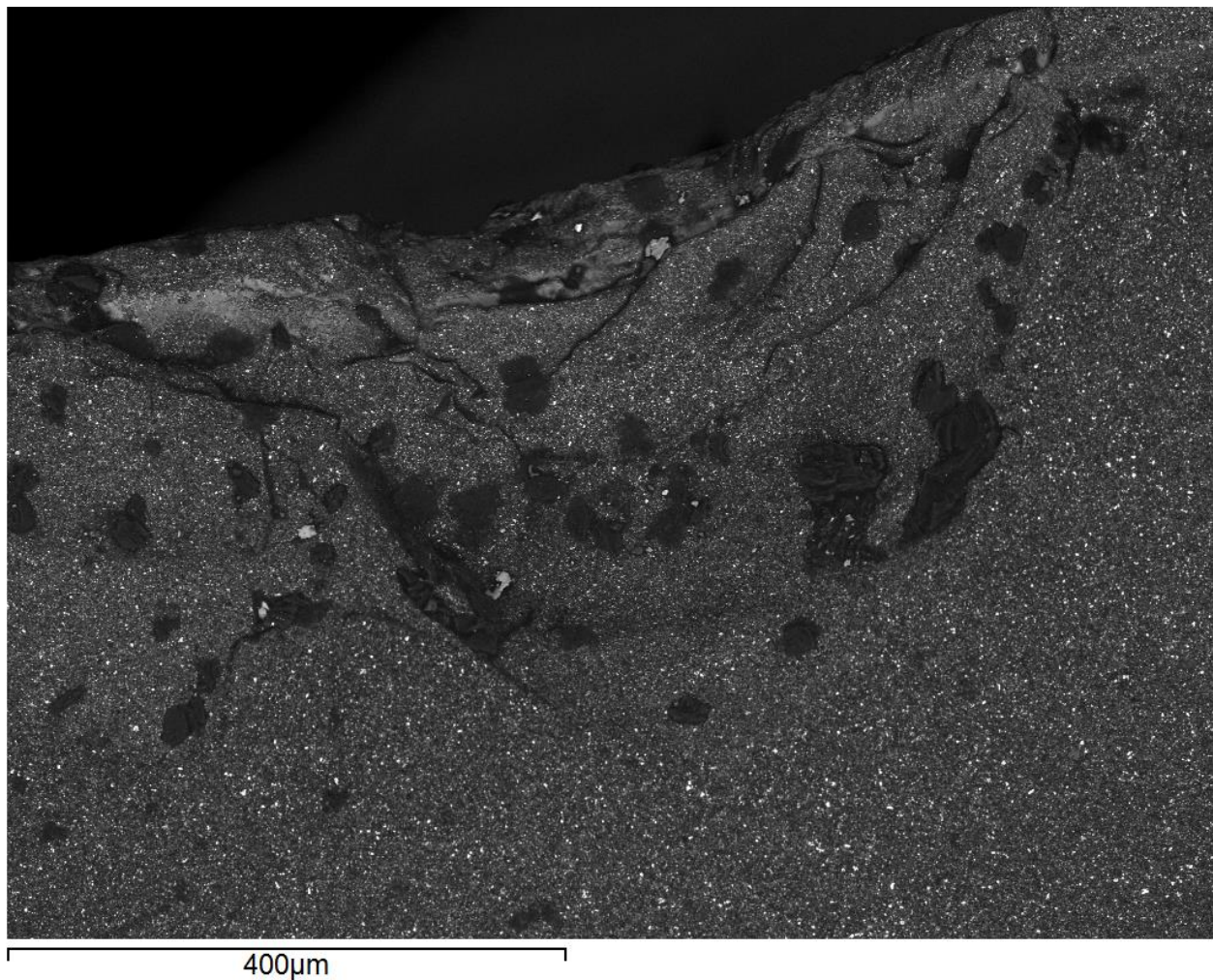
## 5.4 Scanning Electron Microscopy (SEM)

Scanning Electron Microscopy (SEM) can offer superior magnification compared to optical microscopy. This can be combined with Energy-dispersive X-ray spectroscopy (EDS) to perform chemical analysis on parts of the cutting insert surface.

All images in this section were obtained using secondary electron detection in a Hitachi TM3030 table top scanning electron microscope.

### 5.4.1 Insert 14 – Chipped edge

Figure 5-14 shows a SEM image of insert 14 focused on region 1 defined previously in figure 5-7. The image width is 862.5 $\mu$ m and was captured using an acceleration voltage of 15 kV.



*Figure 5-14 – SEM image of large chip on insert 14*

The image shows what appeared to be cracks or steps that radiate away from the edge (indicated in red in figure 5-15). The speckle pattern was consistent with pcBN crystal dispersion in a tungsten carbide binder.

The black spots were most likely to be oil or loosely bonded carbon deposits that diminished the conductivity of the exposed surface accessible to the electron beam. Whilst every effort was made to keep the samples clean for SEM analysis, they were retrieved from a working production line and have been exposed to coolants, lubricants and other contaminants. The inserts were not cleaned prior to SEM analysis as cleaning solvents and chemicals can mask the presence of cracks.

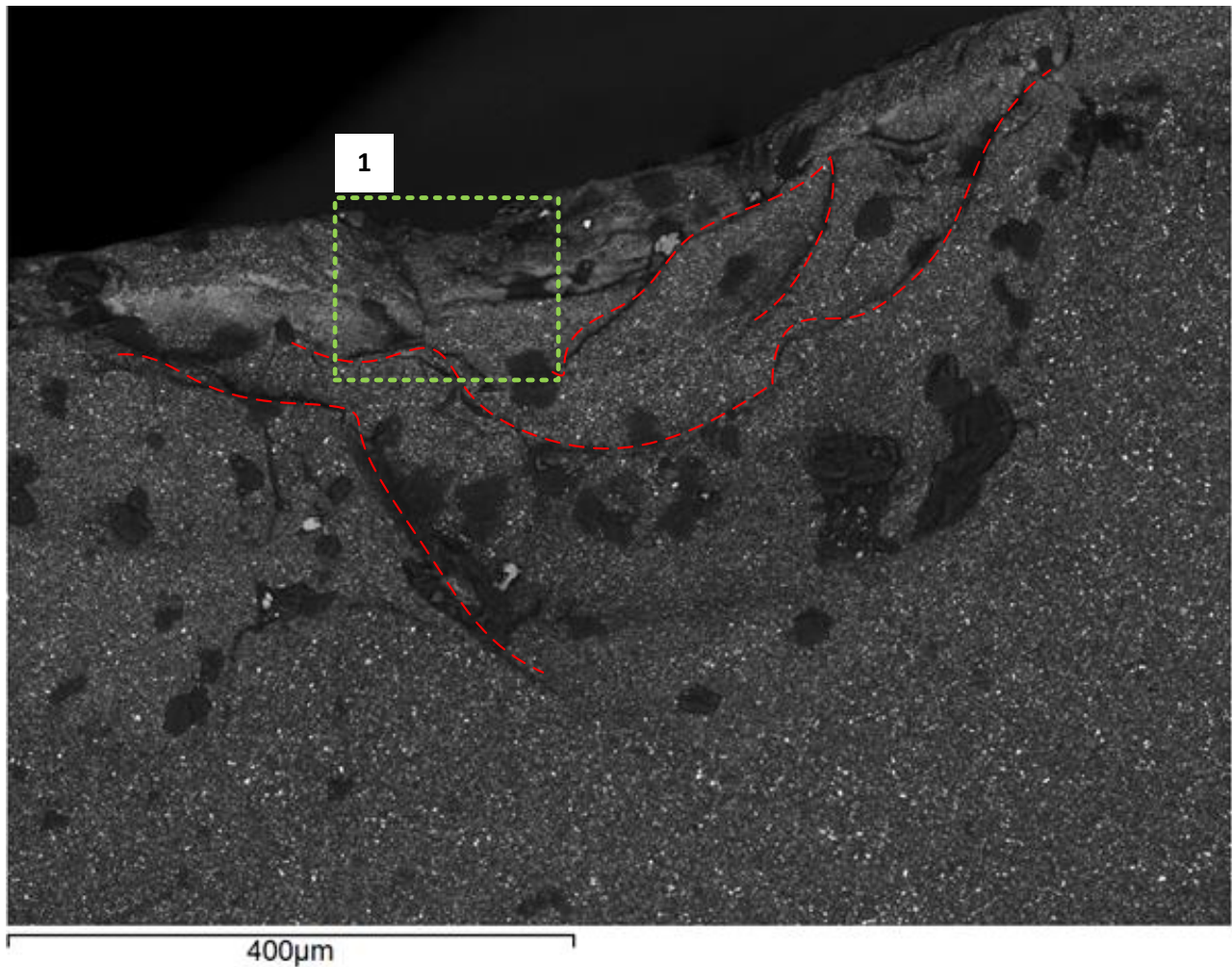
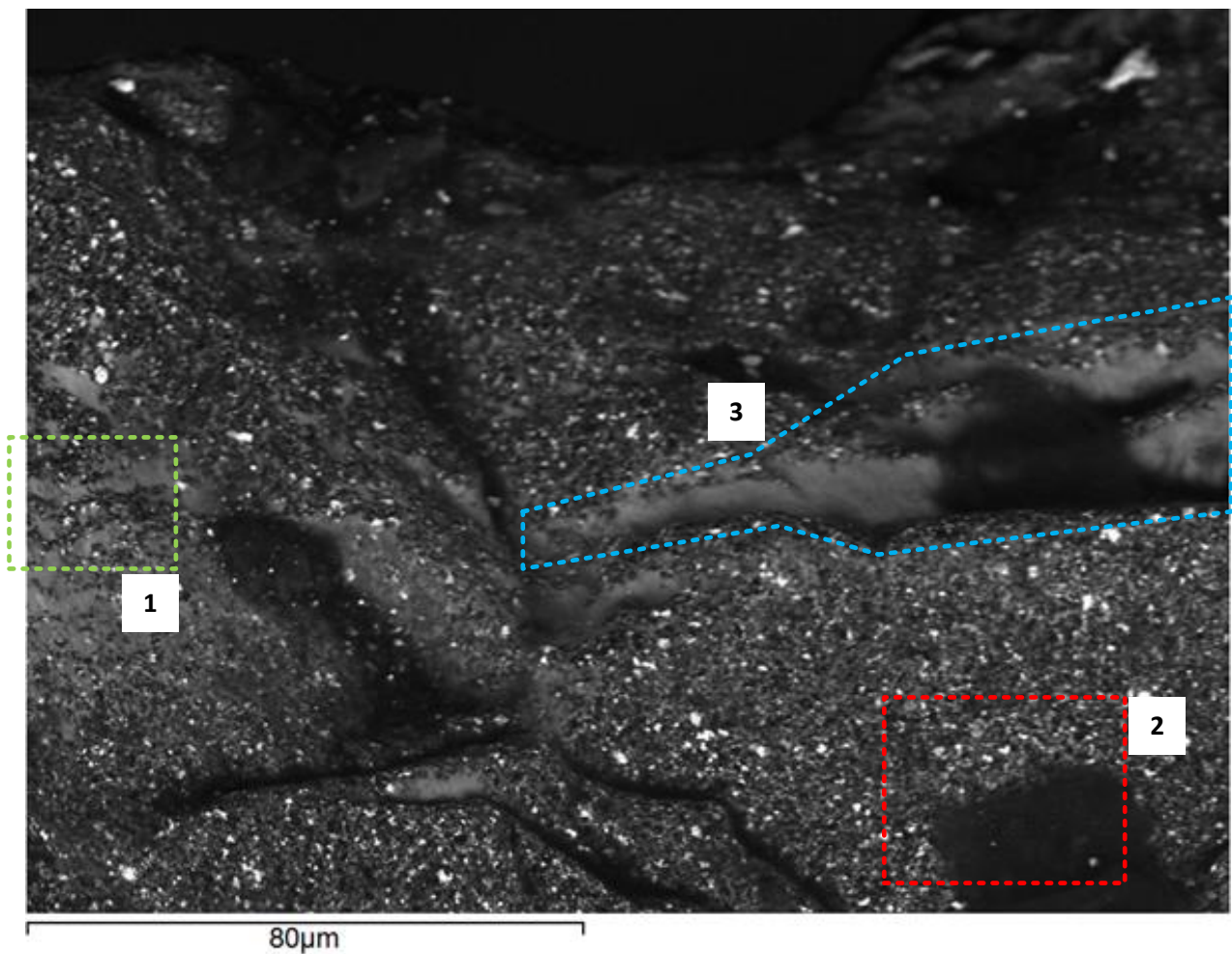


Figure 5-15 – SEM image of large chip on insert 14 showing crack boundaries

The nature of the cracking pattern indicated in red appears characteristic of tungsten carbide failure following exposure to vibrational and cyclic thermal loading (Dogra *et al.*, 2012).



Figure 5-16 shows a higher magnification of region 1 (green) in figure 5-15. At this level of magnification a distinctly different surface becomes visible as indicated by region 3 (blue).



*Figure 5-16 – SEM image of large chip on insert 14 (high magnification)*

The texture of this region was smooth compared to the surrounding speckle pattern of the pcBN. Its appearance was not dissimilar to that of comb and parallel cracking found by Malakizadi, Sadik and Nyborg, 2013, suggesting thermal and stress related cracking.

#### 5.4.2 Insert 14 – Chemical Analysis of Speckle and Black Patterns

Insert 14 was analysed using EDS to determine the chemical makeup of the speckle textured surface and the black spots. Figure 5-17 shows a higher magnification of region 2 (red) in figure 5-16. An average chemical spectrum was acquired from region 1 (green) and point 2 (red). The dominant elements found in each zone are given in tables 5-2 and 5-3 respectively.

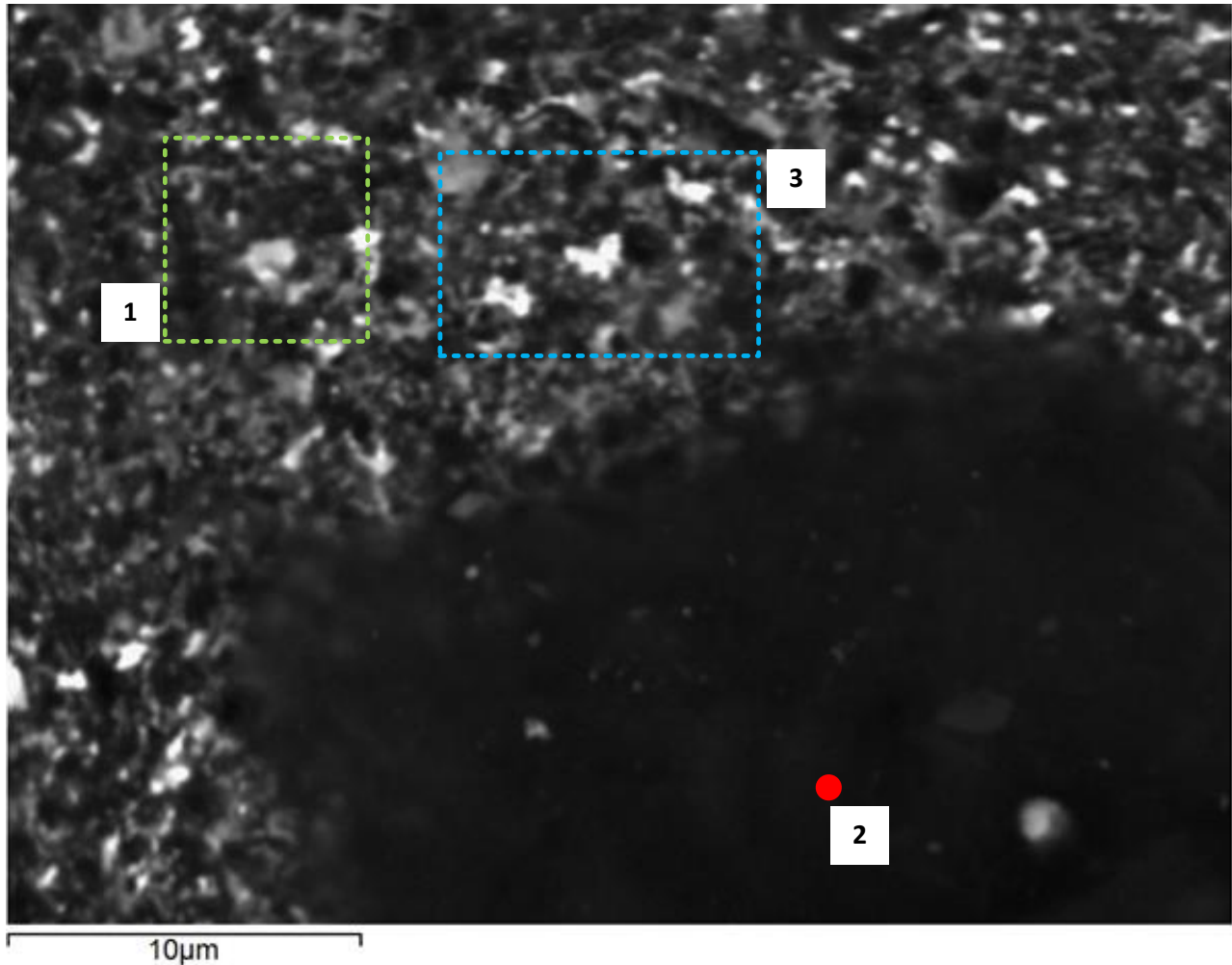


Figure 5-17 – EDS chemical spectrum sample regions

Element	Weight %	Weight % $\sigma$	Atomic %
Carbon	28.613	0.554	36.631
Nitrogen	18.845	0.607	20.687
Boron	16.819	1.035	23.922
Oxygen	14.113	0.321	13.564
Tungsten	8.358	0.167	0.699

Table 5-2 – Dominant chemical elements found in speckle zone indicated by region 1 (green)

Element	Weight %	Weight % $\sigma$	Atomic %
Carbon	56.094	6.812	55.310
Boron	37.908	7.532	41.527
Oxygen	3.505	0.444	2.594
Cobalt	0.682	0.093	0.137
Tungsten	0.419	0.059	0.027

Table 5-3 – Dominant chemical elements found in black spot indicated by point 2 (red)

Region 1 (green) in figure 5-17 shows approximately balanced parts boron and nitrogen as was to be expected from cBN (having the chemical composition BN). The region also had high concentrations of carbon which was consistent with the tungsten carbide binder material and contamination from oils.

Point 2 (red) in figure 5-17 showed very high concentrations of carbon, consistent with carbon deposits or dirt. Interestingly, this region showed disproportionately high concentrations of boron which was balanced by nitrogen.

B<sub>2</sub>O<sub>3</sub> can be used to accelerate the synthesis of hexagonal boron nitride (hBN). hBN is used in the synthesis of cBN, which again, can be accelerated by the addition of B<sub>2</sub>O<sub>3</sub> (Choi *et al.*, 1993). B<sub>2</sub>O<sub>3</sub> will bond with carbon to form boron carbide (B<sub>4</sub>C) at sufficiently high temperatures (1350°C). It is not known as to whether or not B<sub>2</sub>O<sub>3</sub> had been used in the manufacture of this insert, but if it had, it may support a hypothesis that clumps of boron carbide exist in the pcBN substrate. Boron carbide has a strength comparable to cBN, nonetheless, it is still regarded as an undesirable impurity.

The chemical analysis from the speckle zone was used to estimate the chemical composition of the insert. Cubic boron nitride has the chemical formula BN, thus consists of equal parts boron and nitrogen. Similarly tungsten carbide has the chemical formula WC and consists of equal parts tungsten and carbon. In each case, only tungsten and boron are unique to their respective materials as carbon and nitrogen may be introduced from other sources, thus the ratio of CBN to tungsten carbide can be approximated as shown in equation 5-1.

$$100\% \times \frac{\%wt\ BN}{\%wt\ BN + \%wt\ WC} = 100\% \times \frac{16.819}{16.819 + 8.358} = 66\% \quad (5-1)$$

However this equation assumes that the chemical makeup of the surface spectrum represents the average of the interior which is unlikely due to the skin effects of the cBN binding process, wear, cleaning solvents and oxidisation of WC (occurring above temperatures of 500°C).

Figure 5-18 shows a high magnification scan of region 1 (green) in figure 5-16. In this image the very dark and very light speckle pattern is interrupted by a grey substance. This substance was found predominantly near failed or damaged zones across all inserts inspected.

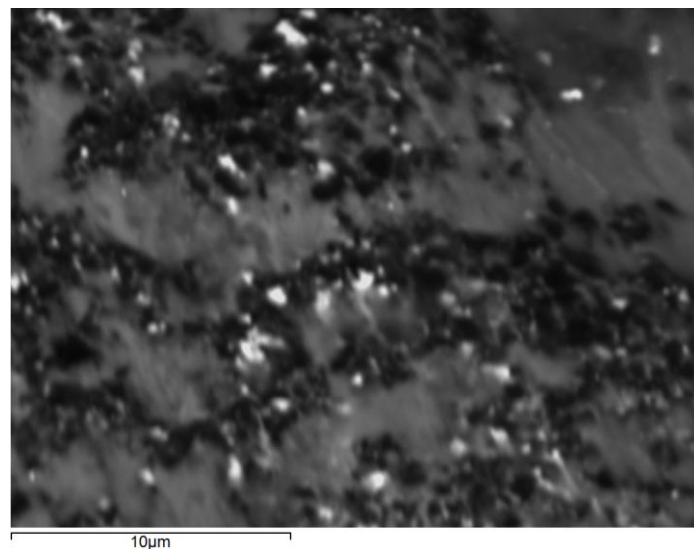


Figure 5-18 – Region 1 (green) from figure 5-16

Two possible theories are proposed. The first is that the grey material revealed itself as a result of surface stretching, wherein cBN crystals clump and were not uniformly distributed as the material stretched. The second theory is that the grey material was composed mainly of workpiece material that had been driven into troughs on the materials surface. A more detailed analysis of this material is given later in section 5.4.6.

#### 5.4.3 Insert 16 – Tungsten Carbide Chemical Analysis

An EDS chemical analysis was performed on the bright spot indicated in figure 5-19 (from region 3 (blue) of figure 5-17) and the weights of tungsten and carbon are given in table 5-4 below.

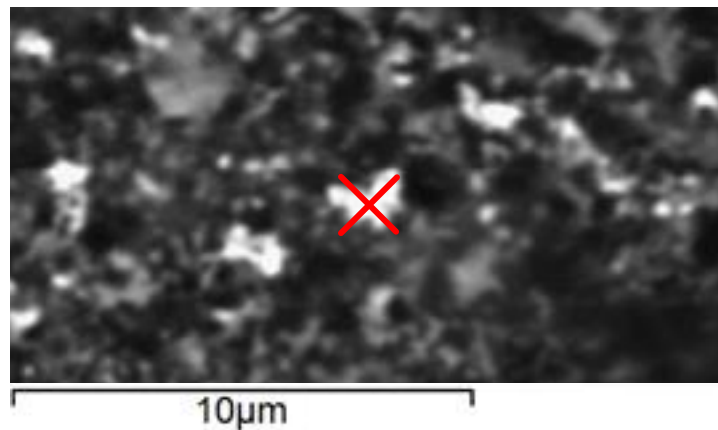


Figure 5-19 – Region 3 (blue) from figure 5-17

Element	Weight %	Weight % $\sigma$	Atomic %
Tungsten	52.498	1.093	7.609
Carbon	23.323	0.932	51.744

Table 5-4 – Chemical analysis of bright spot

The spectrum showed that the bright spot consisted mainly of tungsten. The ratio of tungsten to carbon was higher than expected for WC tungsten carbide, but matched that of  $W_2C$  which is commonly found in powdered tungsten carbide used during the sintering process. Nonetheless, the presence of  $W_2C$  suggests poor homogeneity of the binder material.

#### 5.4.4 Insert 16 – Wear Profile

Figure 5-19 shows the wear profile from a worn edge on insert 16. Several dark deposits were visible along the edge, these were most likely carbon rich deposits such as oil or dirt.

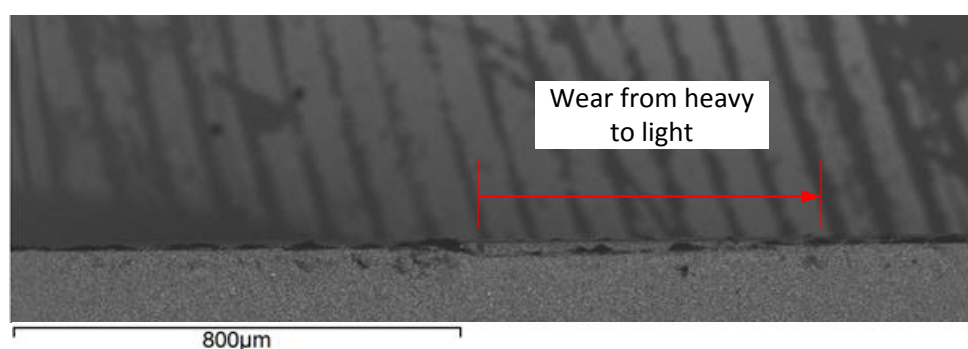
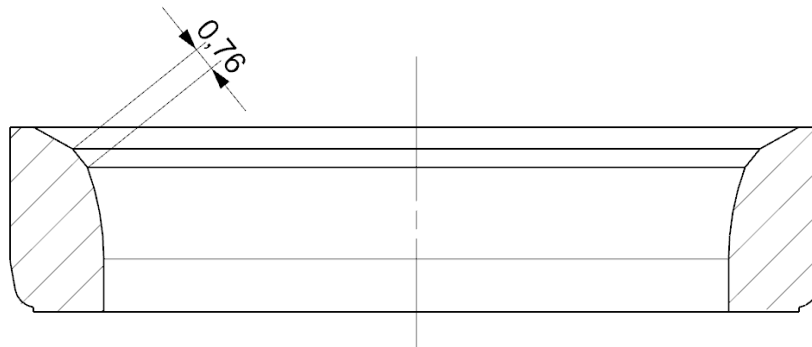


Figure 5-20 – SEM image of worn edge on insert 16

The figure shows that the wear was not linear across the cutting edge, transitioning from heavy at the leftmost edge to light at the rightmost edge. This pattern was expected as discussed earlier and shown in the diagram in figure 5-9.

The width of the wear pattern was approximately 0.7 mm in length and matched the width of the cut indicated in figure 5-21.



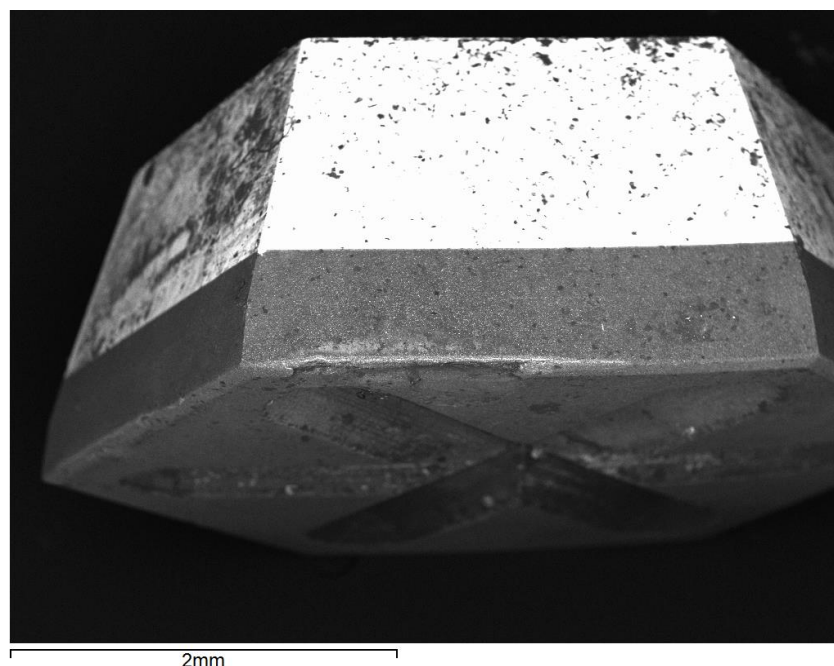
*Figure 5-21 – Width of inside angle*

The radius at the leftmost point of the face is 11.00mm and at the rightmost point on the face is 10.52mm. At 1646 RPM, the velocities at the right and left points are  $1.90 \text{ ms}^{-1}$  and  $1.81 \text{ ms}^{-1}$  respectively, making the velocity at outermost extreme of the wear profile 5% higher than the innermost.

#### **5.4.5 Insert 2 – Rake face deposits**

Insert 2 shows signs of normal wear on one edge and chipping on another, thus inspection of this insert may yield clues as to the processes in action on the run-up to failure.

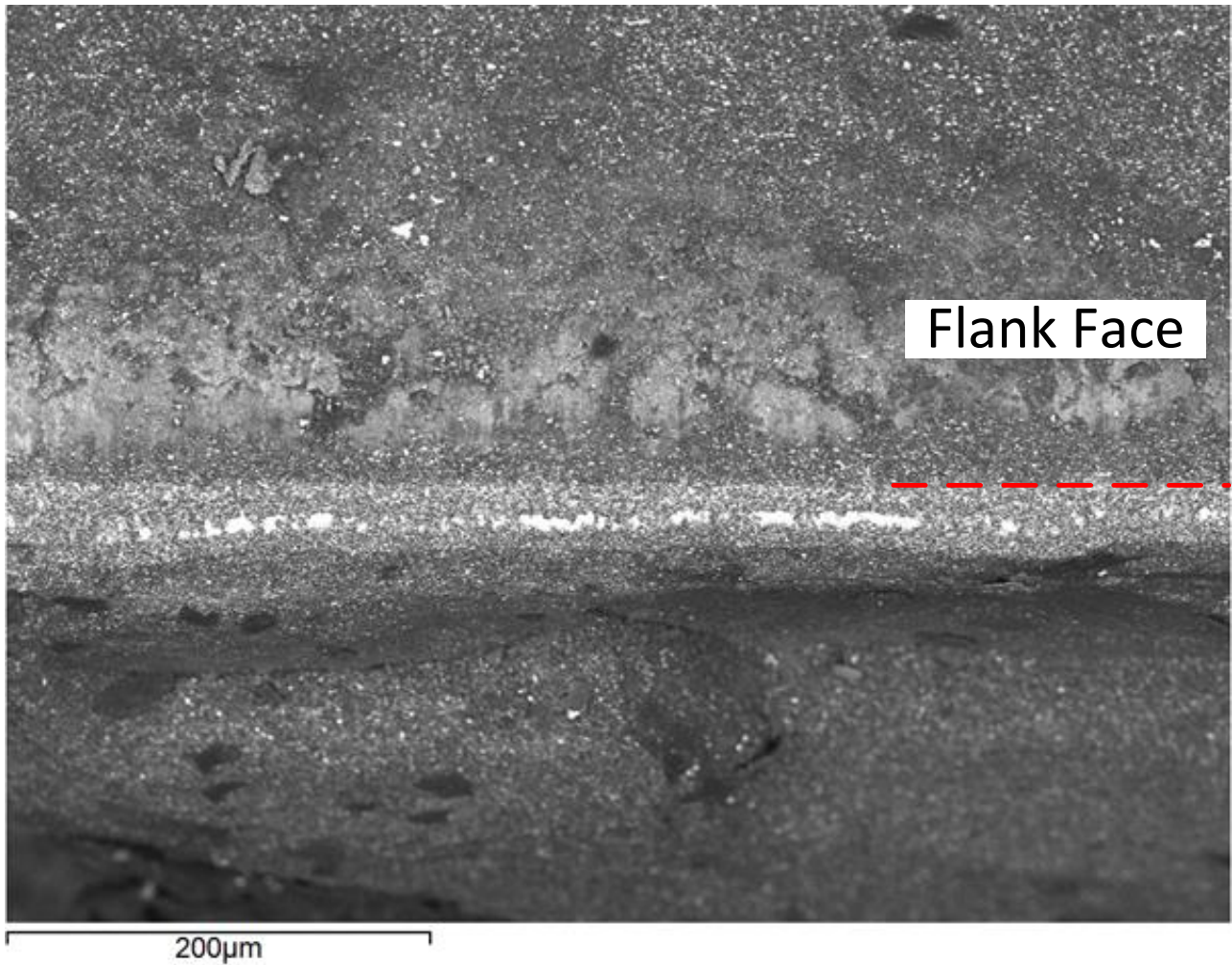
Figure 5-22 below shows insert 2, loaded on its side, at 40x magnification. From this view, it was possible to look down at the wear zone and flank face (where comb cracks were most likely to be visible).



*Figure 5-22 – SEM image of damaged edge on insert 2*



Figure 5-23 shows the damaged zone at higher magnification. The location of the red line indicates the approximate location of the cutting radius (heavily worn). The area below the cutting radius was material that had been exposed after the rake face had been chipped away.



*Figure 5-23 – SEM image of damaged edge on insert 2 (high magnification)*

The figure shows several different aberrations across what remained of the cutting radius. It is broken by a series of bright deposits scattered in a horizontal line from left to right. The flank face shows obvious scarring, and perhaps metallurgical changes, including evidence of diffusion.



Figure 5-24 shows a close-up view of one of the bright deposits found near the cutting radius. The deposit did not appear to be raised above the surface of the rake face but instead seemed to be flush with it.

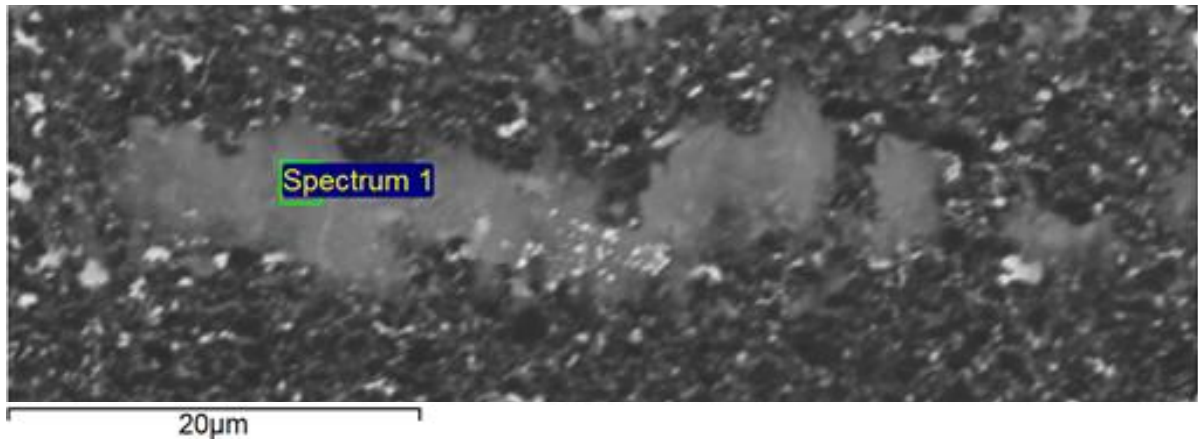


Figure 5-24 – SEM image of deposits on insert 2

Table 5-5 shows the chemical spectrum from the area indicated in figure 5-24, as well as the quoted chemical composition of the Novofr AR20 (the copper infiltrated sintered high speed steel workpiece material).

Element	Weight %	Weight % $\sigma$	Atomic %	Weight % in Novofr AR20
Oxygen	30.799	0.560	53.792	0.00 - 0.00
Iron	16.692	0.378	8.352	23.15 - 56.95
Chromium	12.661	0.286	6.804	3.50 - 5.50
Tungsten	11.237	0.409	1.708	2.50 - 4.50
Carbon	7.317	0.511	17.023	0.80 - 1.30
Manganese	6.986	0.272	3.553	0.30 - 1.50
Copper	6.475	0.423	2.848	10.00 - 20.00
Cobalt	2.338	0.283	1.109	15.00 - 22.00
Silicon	2.279	0.143	2.268	0.50 - 2.00
Sulphur	1.790	0.109	1.560	0.15 - 0.75
Vanadium	1.009	0.128	0.553	1.00 - 2.30

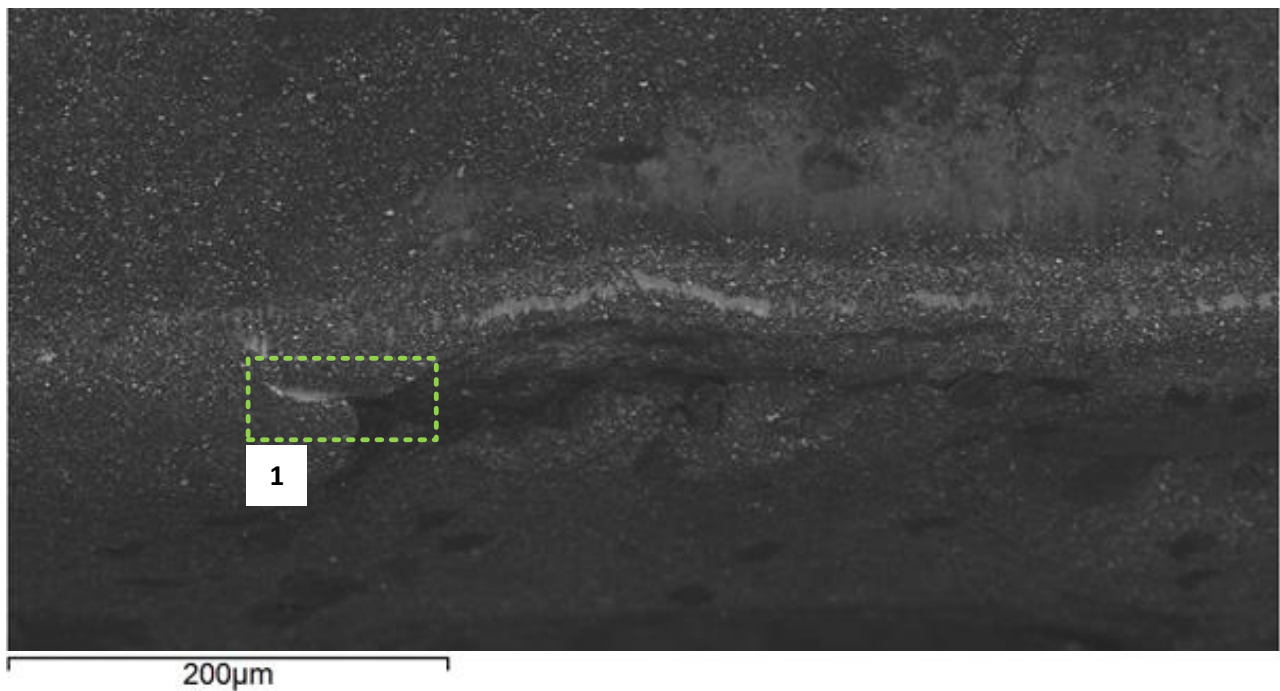
Table 5-5 – EDS chemical spectrum of deposit found on insert 2

The presence of chemical elements correlates with the constituent elements of Novofr AR20, but not at their original concentrations. The elements were also supplemented by disproportionately large amounts of tungsten, suggesting that diffusion and other chemical processes had occurred in this region.

The ratio of iron to oxygen in this region was 1.63:3.00 and very similar to that of iron oxide which is 2.00:3.00. Oxidisation of iron in the deposit is not likely to have taken place after machining as the concentration of chromium was sufficiently high in the sample to protect the iron. If oxidisation had occurred, it must have done so at elevated temperatures.

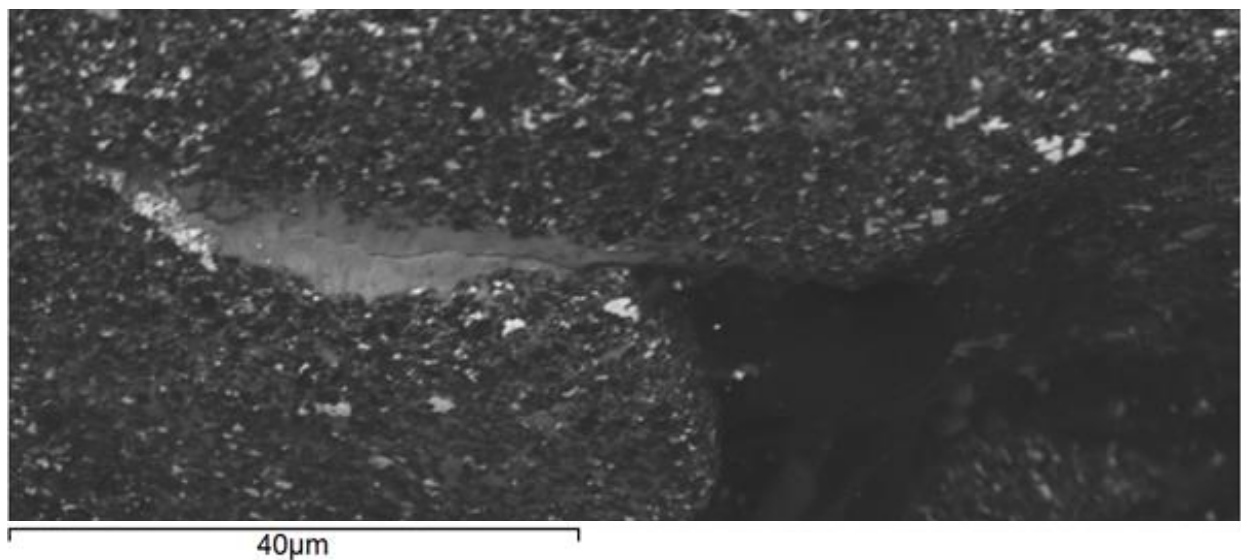
#### 5.4.6 Insert 2 – Crack

A small crack was present at the far left of the damaged area running nearly parallel with the cutting edge as shown in region 1 (green) of figure 5-25. The crack was approximately 38 $\mu$ m in length.



*Figure 5-25 – SEM image of crack found on insert 2*

The material to the left of the crack was unworn rake face material, whereas the material to the right was what remained after the tool material had been chipped and worn away. Figure 5-26 shows a higher magnification view of region 1 (green) indicated in figure 5-25.



*Figure 5-26 – SEM image of crack found on insert 2 (high magnification)*

In this figure, it is clear to see the growth of a primary crack through the cutter material. This crack appeared to be filled with a secondary material which itself was cracked. The chemical composition of the region of secondary material shown in figure 5-27 is given in table 5-6.

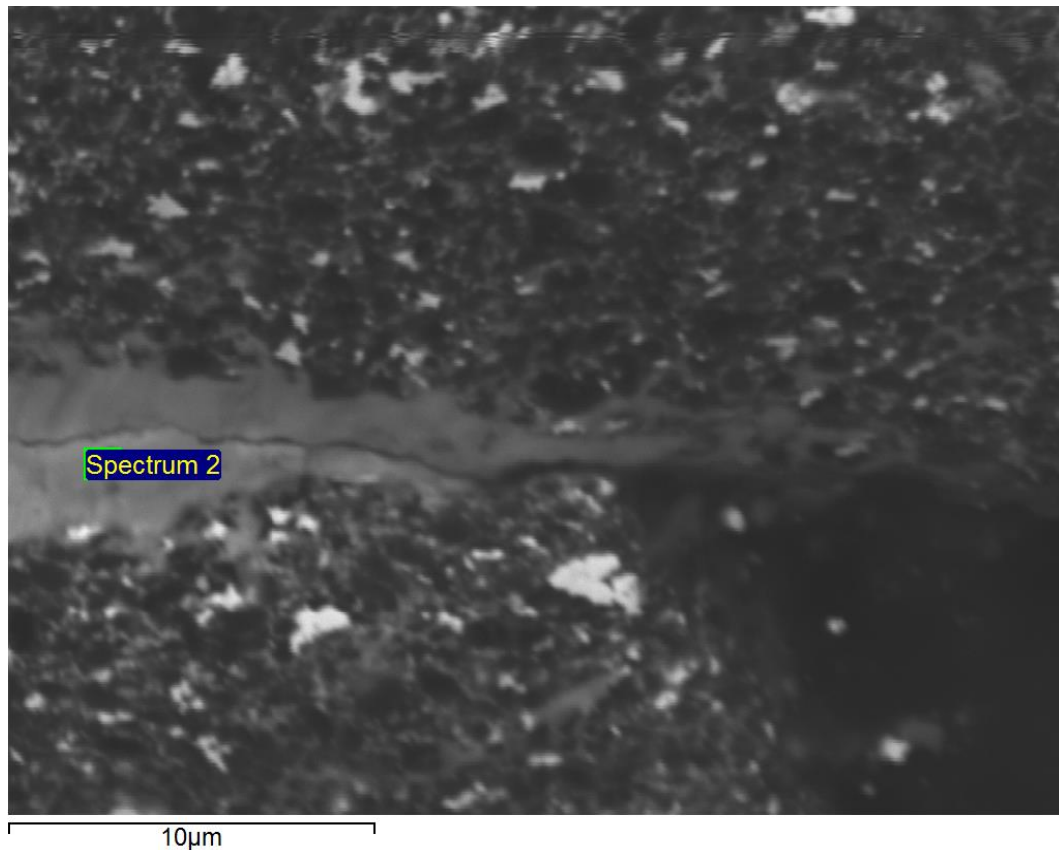


Figure 5-27 – EDS chemical spectrum sample zone

Element	Weight %	Weight % $\sigma$	Atomic %
Cobalt	36.066	3.084	33.359
Iron	31.157	2.784	30.411
Manganese	17.611	2.102	17.473
Chromium	7.364	1.360	7.720
Tungsten	4.013	1.184	1.190

Table 5-6 – EDS chemical spectrum of secondary crack material found on insert 2

The secondary material contained large amounts of iron, manganese and chromium and was likely to be workpiece material that was forced into the primary crack due to the high cutting pressure. It was similar in appearance to other lightly colour veins of material found on insert 14 and again on insert 2 in regions that were exposed to the workpiece.

Large amounts of cobalt were detected which was unexpected. Despite official data for the tungsten carbide substrate being unavailable, cobalt was not expected to be a component of the WC substrate since it was only found in trace amounts elsewhere on the sample in undamaged areas. Cobalt was present in large quantities in the AR20 sintered valve seat material, although not in concentrations as high as 36%. This could be an indicator that valve seat material had undergone melt or diffusion at this location which had increased the concentration of cobalt.

It is possible that the secondary material seen here was accelerating the growth rate of the primary crack. If secondary material (from the workpiece) with a higher rate of thermal expansion was forced into primary cracks in the tool material by high cutting pressures, then when that material expanded due to elevated cutting temperatures, it may have forced open the primary crack. When the cutters retracted from the

workpiece, the secondary material cooled and cracked centrally, thus providing a void for more secondary material to fill.

Table 5-7 shows coefficients of linear thermal expansion of the cutter materials and elements found in the crack shown in figure 5-27.

Compound / Element	Coefficient of linear thermal expansion, $\alpha$ ( $\mu m m^{-1} K^{-1}$ )	Reference
CBN	1.2	Monteiro <i>et al.</i> , 2013
Tungsten carbide	5.9	Hidnert, 1937
<b>Weighted group average</b> (Based on 66% cBN content)	4.3	
Cobalt	11.8	White, 1965
Iron	12.7	
Manganese	24.5	
Chromium	5.2	Roberts, White and Fawcett, 1983
<b>Weighted group average</b> (Based on % weights given in table 5-6)	14.0	

Table 5-7 – Weighted coefficients of thermal expansion for cBN and tungsten carbide

A weighted average is given for the cutter material, based on 66% CBN content (found earlier using equation 5-1) and likewise for the workpiece material found in the crack as shown in table 5-6. The weighted group average for the tool material was found according to equation 5-2, and likewise for the workpiece material according to equation 5-3.

$$\alpha_{CBN}\%wt_{CBN} + \alpha_{WC}\%wt_{WC} = 4.3 \mu m m^{-1} K^{-1} \quad (5-2)$$

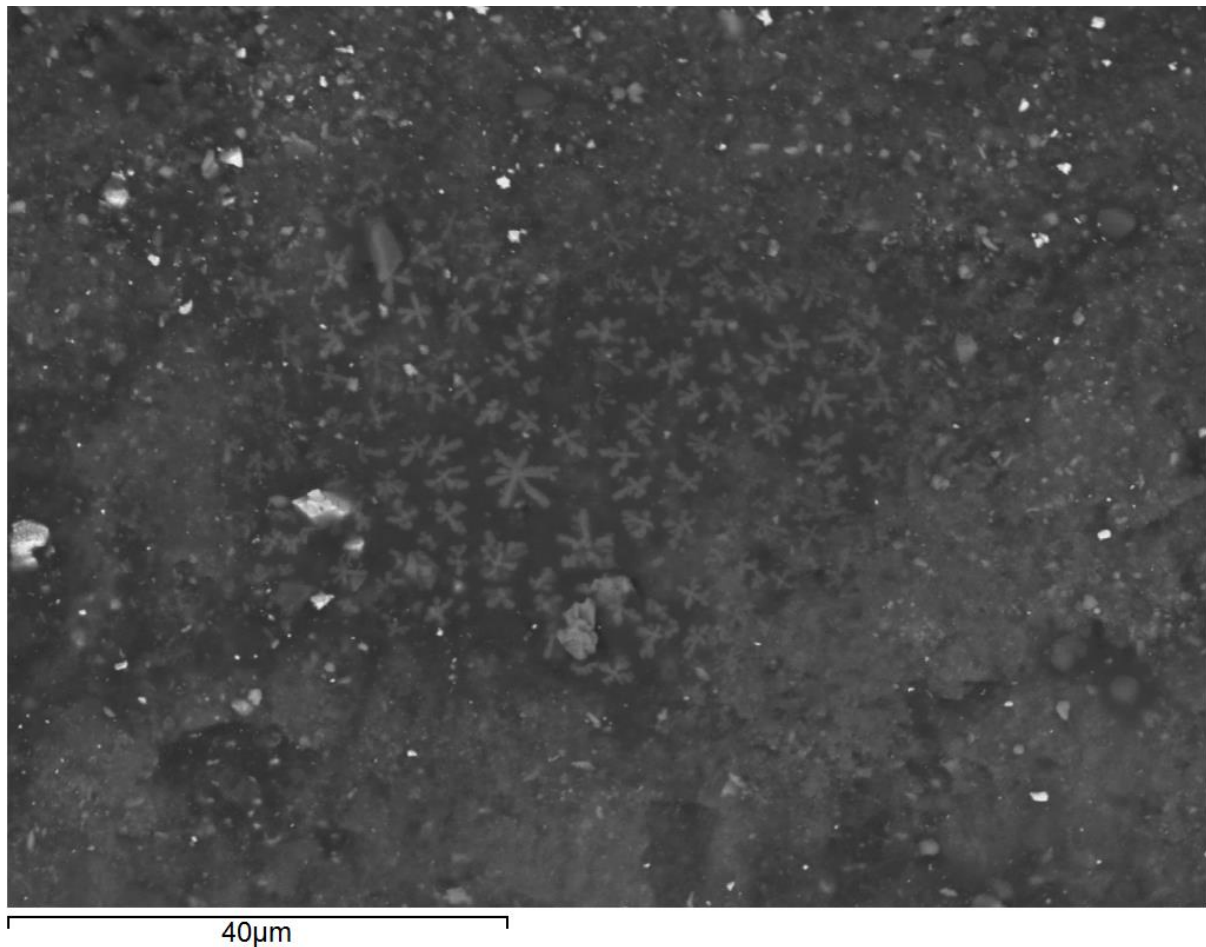
$$\sum_{Element=Co,Fe,Mn,Cr} \alpha_{Element} \frac{\%wt Element}{\%wt_{Co} + \%wt_{Fe} + \%wt_{Mn} + \%wt_{Cr}} = 14.0 \mu m m^{-1} K^{-1} \quad (5-3)$$

The weighted coefficients of linear thermal expansion show that the composite material found in the crack had a coefficient of thermal expansion more than three times greater than that of the pcBN insert. This supports the hypothesis that the mismatch in workpiece and tool material coefficients of thermal expansion, coupled with the ability of AR20 to melt and diffuse into the pcBN substrate, may have accelerated the growth rate of cracks running parallel along the cutting insert rake face.

#### 5.4.7 Insert 2 – Crystal Pattern

Figure 5-28 shows what appears to be a crystal growth pattern found on insert 2, approximately 100 $\mu$ m from the boundary line between the top surface of the insert and the rake face, near a worn section of the rake face.

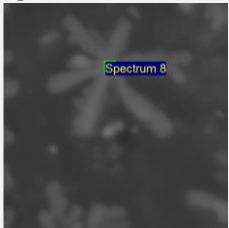
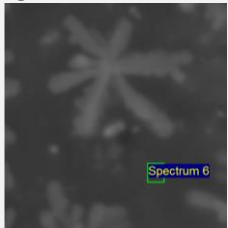
This section of the chip was exposed to high temperature and is unlikely to have come into contact with the workpiece. Survival of crystals of this size and shape that existed before sintering is extremely unlikely due to the mixing and compression that would have taken place during the sintering process. The surface topology of the crystal pattern was flat and none of the crystals appeared to be growing out of the material. These two observations suggested that the crystals must have grown during manufacturing under the high temperature and pressure of the sintering process.



*Figure 5-28 – SEM image of crystal pattern found on insert 2*

Chemical analysis of the crystals and surrounding area indicated high concentrations of boron with low concentrations of nitrogen as shown below in table 5-8. Where boron was found on the insert, it was expected to be balanced by nitrogen, since both elements are equal components of boron nitride (BN). This was approximately true for all previous chemical spectra taken across unworn surfaces of the insert. The imbalance in this region suggested that a chemical reaction had taken place which had broken down the BN crystals in favour of boron crystal growth. The dark region engulfing the crystals was similar in chemistry to the crystals themselves.

Table 5-8 shows the chemical spectra taken from one of the crystals compared to the spectra of the material surrounding the crystal.

Element	Weight % 	Weight % 	Difference %
Boron	52.115	55.829	+3.714
Carbon	39.121	36.882	-2.239
Oxygen	3.684	2.747	-0.937
Nitrogen	2.494	1.896	-0.598
Potassium	0.554	0.491	-0.063
Silicon	0.498	0.346	-0.152
Iron	0.404	0.535	+0.131

*Table 5-8 – EDS chemical spectrum of crystal pattern found on insert 2*

As the table shows, the spectra for the two regions is very similar, but the imbalance between boron and nitrogen shows that BN had broken down in this region. The crystals were rich in boron and carbon suggesting that they could be crystals of boron carbide.

There was almost no trace of tungsten in this region which was unexpected. This could be due to a manufacturing process defect that had led to poor homogeneity of the powdered sintering material mixture.

## 5.5 Chapter Summary

Chapter Two presented a literature review that documented various possible damage mechanisms that could explain the random chipping, observed in the case study for this work, when pcBN cutting inserts are used to cut multi-angle valve seats.

This chapter presented experimental attempts to identify damage and manufacturer defects in a series of cutting inserts, recovered from the valve seat cutting process referred to in the case study.

Evidence was presented that shows the presence of evolving damage conditions such as flank and crater wear, diffusion and chemical changes brought about as a result of heat exposure. Without complete tool history however, it has not been possible to determine the rates of wear or damage. This chapter also produced evidence to support a hypothesis that the relatively large difference in coefficients of thermal expansion between pcBN and steel may be accelerating crack growth.

No evidence of manufacturing defects such as voids or pre-existing cracks or faults in the inserts was found. All of the damage observed was regarded as typical for cutting inserts towards the end of their rated life. The evidence presented in this chapter cannot explain, however, the sudden and random nature of pcBN cutting insert failure when machining multi-angle valve seats.

Failure to identify clear manufacturing or material defects in the pcBN cutting inserts adds further support to the argument presented in this work. Specifically, that: in conjunction with the low fracture toughness of pcBN, random pcBN cutting insert failure is due to vibration caused by the radial imbalance that develops during multi-angle valve seat cutting.



# Chapter Six – Dynamic Analysis of the Valve Seat Cutting Operation

---

## 6.1 Introduction

Ford's own investigations raise concerns about the valve seat machining process. Some of these concerns could explain why Ford were seeing chipping when using pcBN cutting inserts. Specifically:

- the way the cylinder head is held during machining is suboptimal for reducing flex and may lead to the head flexing away from the tool if cutting thrust loads are sufficiently high;
- the quick release zero point locators (ZPL) that clamp to the head and are responsible for maintaining a known relative offset between the head and machine may be slipping relative to the head. Any subsequent misalignment may lead to tools engaging earlier or at different locations than expected; and
- the cylinder head and fixture system may be vulnerable to low and medium frequency resonance during machining.

The literature review for this work identified excessive vibration due to some imbalance in the system and vibration due to resonance as common causes of instability and damage within cutting systems (Lacerda and Siqueira, 2012; Moradi *et al.*, 2013; Fu and Zheng, 2014; Iglesias *et al.*, 2016). As stated in earlier chapters, the prevailing theory is that cutting inserts chip randomly due to the radial imbalance that arises due to each cutter machining a different angle on the valve seat. To have confidence in this theory, it is important to first rule out the possible causes of tool damage listed above.

This chapter looks at the cutting system for the Fox cylinder head, shown in figure 6-1, and investigates the stiffness of the cutting system and the structural stability of the ZPL support structures with reference to finite element studies. This chapter also presents an experiment performed to determine whether or not the system is affected by resonance.

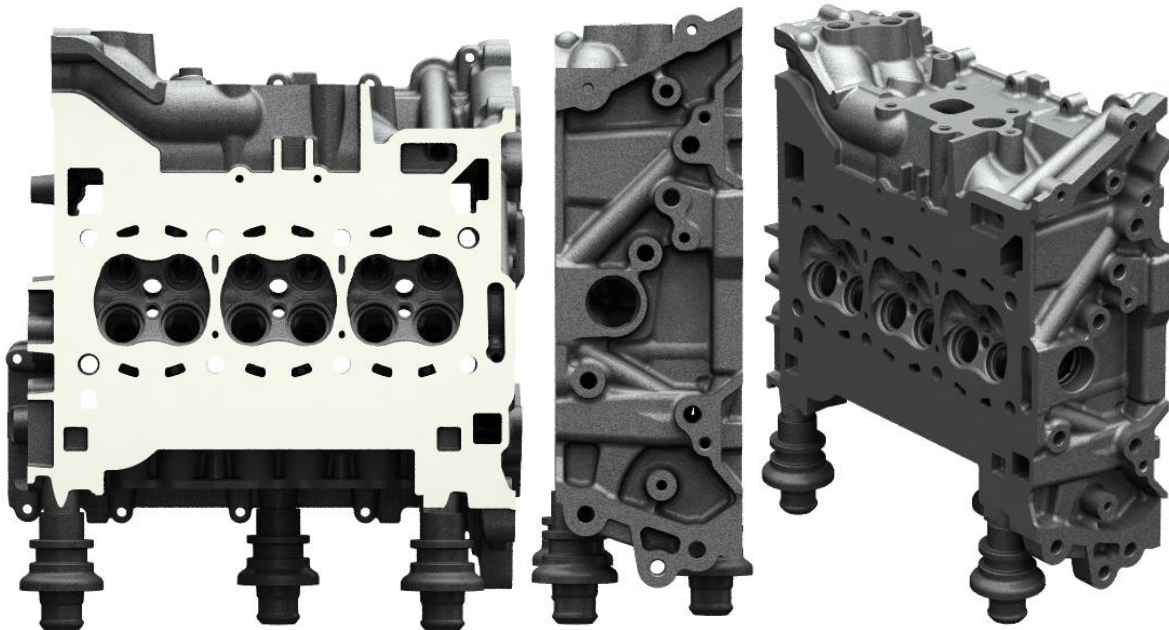


Figure 6-1 – Fox Upgrade 1.0L cylinder head rendering

Valve seat and valve guide geometric tolerances are amongst the most restrictive of anywhere in the engine. This is largely due to the high quality seal that must be achieved to reach engine emissions and performance targets and to ensure longevity of the engine. The seat and guide must maintain a tight coaxial tolerance, as well as meet tight deck, seat and throat angle location tolerances and surface finishes.

Simultaneous cutting of the three angles greatly improves the potential of the operation to maintain tight coaxial tolerances and reduces the time required to perform the operation since it involves fewer tool changes. However, these advantages come at the cost of greater cutting forces, more complicated dynamic behaviour and the risk that imbalanced radial loading will create displacements in the cutting system that lead to out-of-round error and diminished effectiveness of some cutters.

## **6.2 The Machining Process**

The process studied in this chapter is the valve seat and guide semi-finish and finishing operations performed during manufacture of the Fox 1.0L cylinder head. This process was selected because it was observed during production that certain cutting parameters were leading to tool breakages and yielding seats of poor surface finish, often displaying evidence of chatter.

The primary goal of studying this process is to determine if a lack of stiffness in the current fixture design could explain why cutting inserts appear to break randomly during production. Evidence of this may be large displacements of the cylinder head during machining or resonant vibration. This study also aims to determine the relationship between cutting thrust force and cylinder head displacement.

This study looks at the actual production line CNC machine, fixture and cylinder head. The boundary conditions of the system are imposed by the process design and cannot be changed.



The cylinder heads are loaded and unloaded from CNC machines using a quick release mechanism. Central to this mechanism is a structure called a zero point locator (ZPL), shown in figure 6-2 A). Before entering the production line, three ZPLs are bolted to the lower surface of each cylinder head using an M8 bolt as shown in figure 6-2 B). The lower portion of the ZPL mates with a hydraulic clamping system embedded in the CNC machine fixture. The compound structure (cylinder head and three ZPLs) can then be quickly clamped and released from CNC machines throughout the production line.

A) ZPL-bolt cross-section

B) ZPL bolted joint schematic

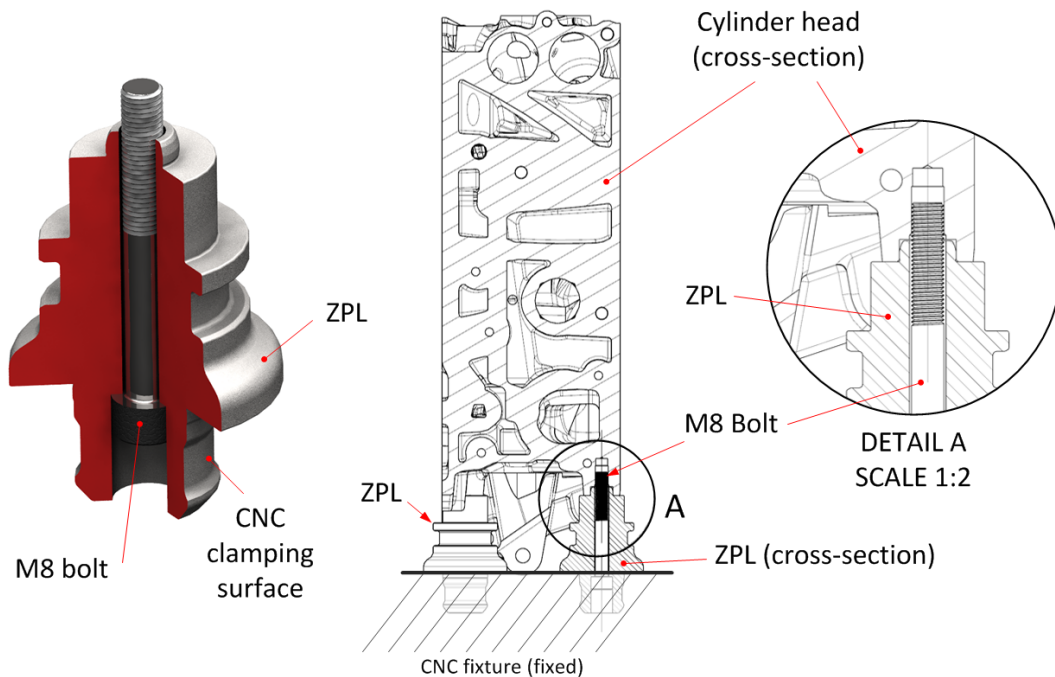


Figure 6-2 – A) ZPL-bolt cross section, B) ZPL bolted joint schematic

Figure 6-3 shows the layout of the cylinder head as it was fitted into the CNC machine. The figure also gives the port and bank definitions as used throughout this chapter. The ports are numbered in the order in which they were cut.

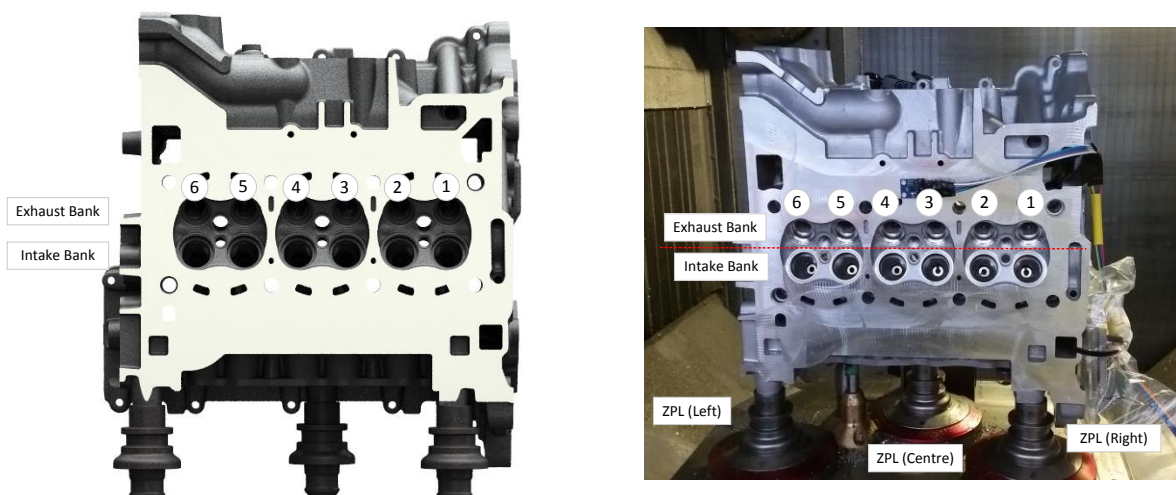


Figure 6-3 - Cylinder head port definitions

The ZPLs must resist a large moment that arises in response to the cutting load. The distance between the foot of the ZPL for the intake and exhaust bank is 163.4 mm and 197.6 mm respectively. For equal forces, the moment applied when machining the exhaust bank is more than 20% higher than the moment applied when machining the intake bank.

As can be seen from figure 6-3, the geometry of the support system is far from ideal for this particular cutting system. Figure 6-4 shows a side view of the cylinder head and ZPL fixture system. The blue and red arrows show the force directions for the intake and exhaust port cutter thrust forces respectively. A) shows the current configuration with the ZPLs on the bottom of the cylinder head. In a more ideal layout, the ZPLs would be placed on the rear of the cylinder head, similar to the hypothetical arrangement shown in B) so that the head would be supported by a compressive load. However such an arrangement was not possible due to the demands of other processes on the production line that require access to the rear of the head.

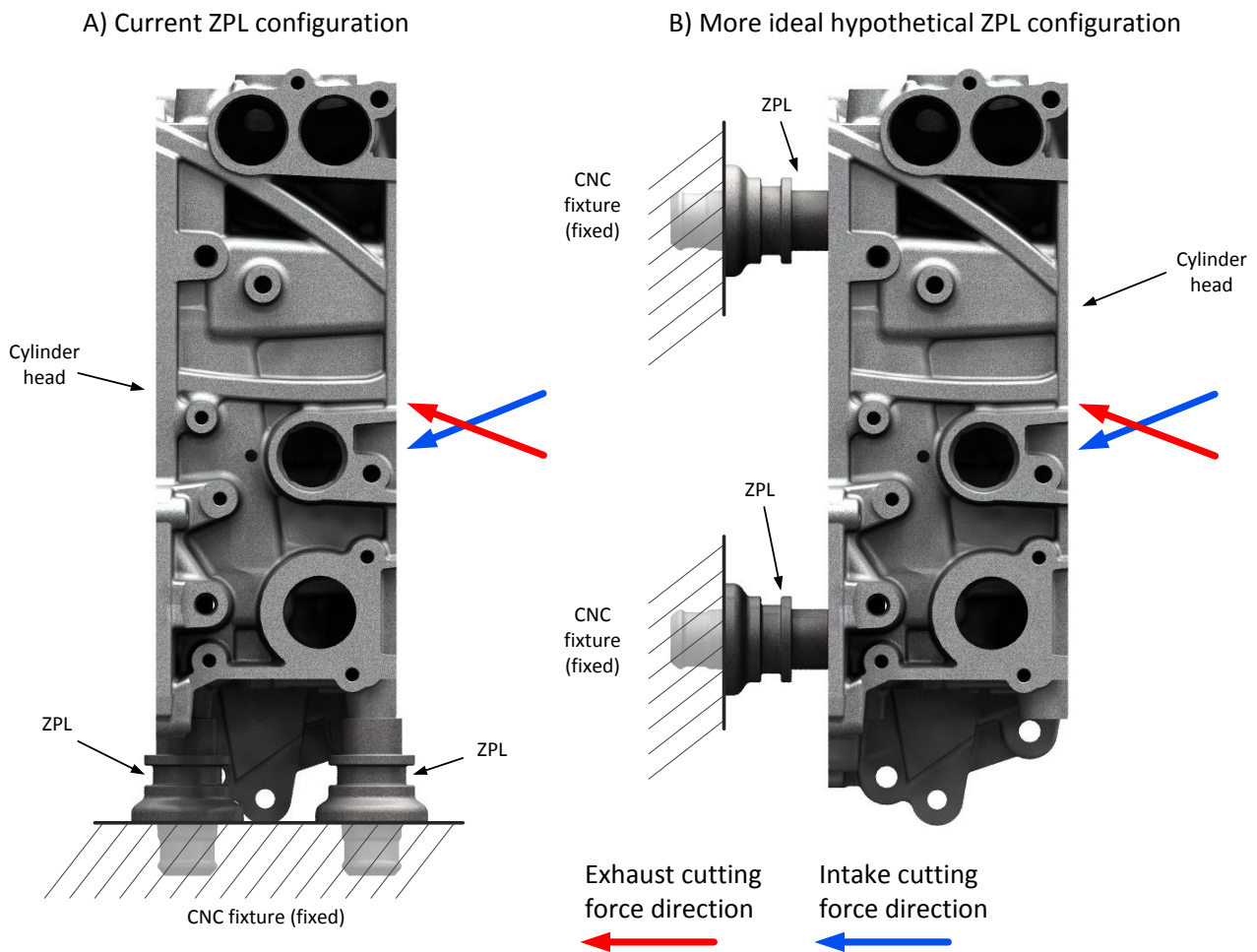


Figure 6-4 – A) Current ZPL configuration, B) more ideal hypothetical ZPL configuration

Figure 6-5 indicates some approximate locations around the head (again, these definitions are used throughout this chapter).

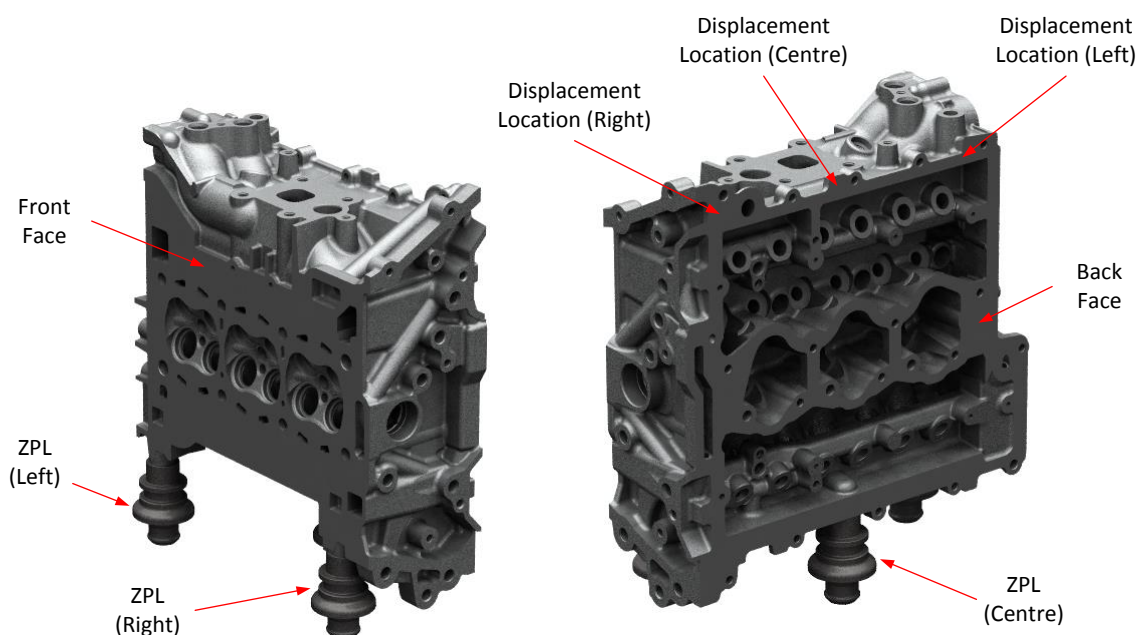
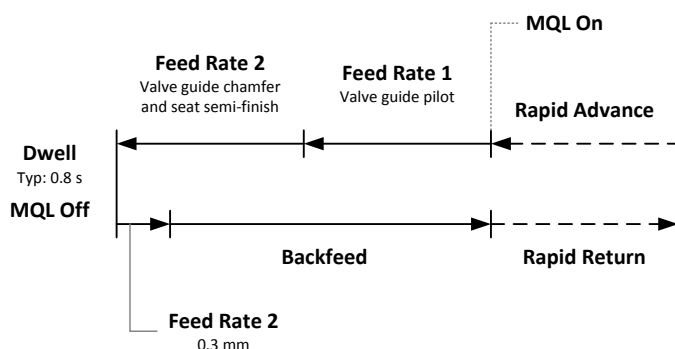
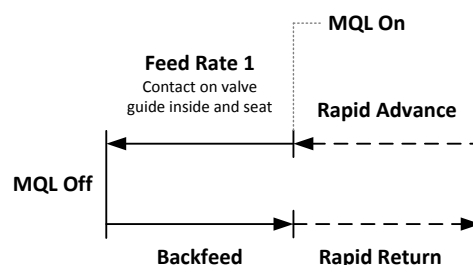


Figure 6-5 - Cylinder head location definitions

Figure 6-6 shows a typical cutting cycle diagram for a two-step process. In cycle 1 the valve guide is piloted and a semi-finish pass is applied to the seat. In cycle 2 the valve guide is reamed and a finishing pass is applied to the seat. During normal production, both cycles are required to fully finish both the intake and exhaust banks. Some cycles perform the valve guide finish ream in a separate third step. However, all cycles incorporate a critical seat and guide finishing step.



Cycle 1: Valve guide pilot and seat semi finish



Cycle 2: Valve guide ream and seat finish

Figure 6-6 – Cutting cycle diagrams

Typical cutting parameters for these cycles are given in table 7.

Parameter	Cycle 1	Cycle 2
Spindle Speed (RPM)	1600	4200 - 6000
Feed Rate 1 (mm rev <sup>-1</sup> )	0.06	0.06
Feed Rate 2 (mm rev <sup>-1</sup> )	0.08	

Table 6-1 – Typical cutting parameters

### 6.3 The Effect of ZPL Bolt Torque on Seat Positional Accuracy

The ZPL bolt torque was studied to better understand its influence on the cutting system. The default intake and exhaust seat finishing cycle was run on two cylinder heads according to cycle 1 shown earlier in figure 6-6, using a typical process spindle speed of 1600 RPM and feed rates of  $0.06 \text{ mm rev}^{-1}$ . Two bolt torques were trialled, 21 Nm and 27 Nm. Bolt torque was set using a computer controlled torque wrench to ensure consistency.

After cutting, both heads were analysed on a Carl Zeiss CenterMax Coordinate-Measuring Machine (CMM). The CMM was used to generate a representative cone by sampling a series of points on the deck, seat and throat angles according to the definitions given in figure 6-7.

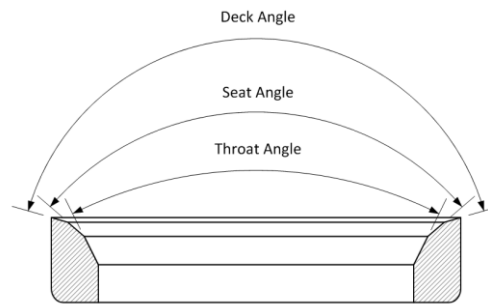


Figure 6-7 – Valve seat cross section showing deck, seat and throat angle definitions

For each cone, the cone angle, and two orientation angles as defined in figure 6-8 were recovered. All parameters were compared against a nominal target design figure for each cone and a deviation was calculated. The results are given in table 6-2.

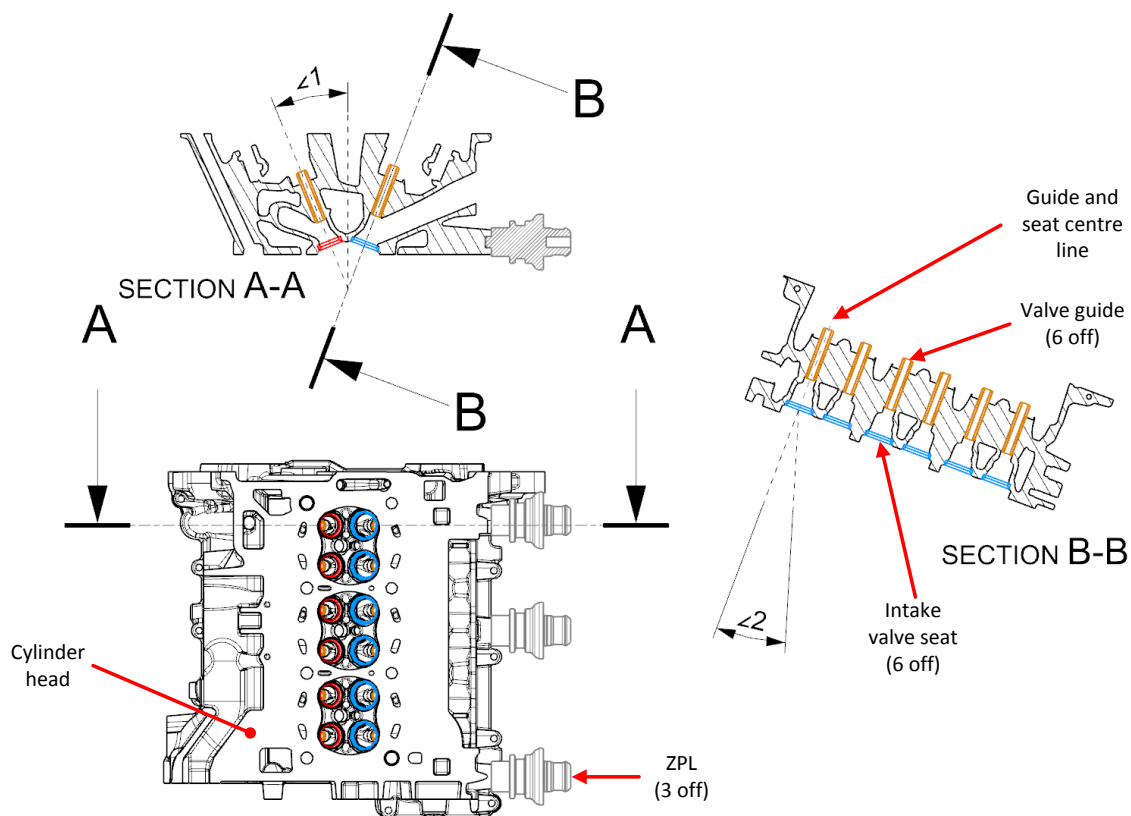


Figure 6-8 – CMM angle definitions  $\angle 1$  and  $\angle 2$

Cone	Reference	21 Nm						27 Nm					
		Exhaust			Intake			Exhaust			Intake		
		Actual (°)	Nominal (°)	Dev. (°)	Actual (°)	Nominal (°)	Dev. (°)	Actual (°)	Nominal (°)	Dev. (°)	Actual (°)	Nominal (°)	Dev. (°)
Deck	∠C	121.141	120.000	1.141	119.969	120.000	-0.031	121.189	120.000	1.189	119.940	120.000	-0.060
	∠1	-21.933	-21.900	-0.033	20.700	20.700	0.000	-21.925	-21.900	-0.025	20.706	20.700	0.006
	∠2	0.033	0.000	0.033	0.028	0.000	0.028	-0.013	0.000	-0.013	0.000	0.000	0.000
Seat	∠C	89.706	90.000	-0.294	89.814	90.000	-0.186	89.761	90.000	-0.239	89.728	90.000	-0.272
	∠1	-21.869	-21.900	0.031	0.021	0.000	0.021	-21.896	-21.900	0.004	0.012	0.000	0.012
	∠2	-0.022	0.000	-0.022	-0.003	0.000	-0.003	0.020	0.000	0.020	-0.040	0.000	-0.040
Throat	∠C	60.023	60.000	0.023	59.997	60.000	-0.003	59.990	60.000	-0.010	60.033	60.000	0.033
	∠1	-21.876	-21.900	0.024	0.015	0.000	0.015	-21.886	-21.900	0.014	0.019	0.000	0.019
	∠2	0.019	0.000	0.019	-0.029	0.000	-0.029	-0.023	0.000	-0.023	0.002	0.000	0.002

Table 6-2 - CMM results

Figure 6-9 shows the summary of position deviations for 27 Nm and 21 Nm for both the exhaust and intake seats.

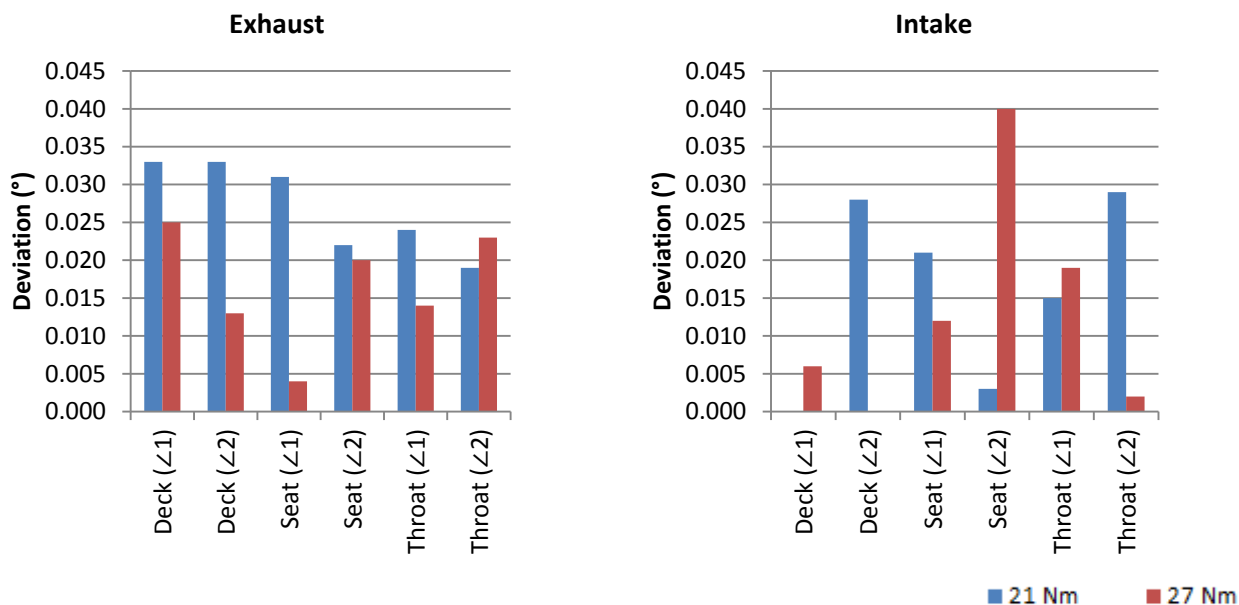


Figure 6-9 – Summary of deviations for position angles (lower is better)

As the figure shows, increasing the bolt torque to 27 Nm decreased the cone position deviation for the exhaust seat on all measures except for the lateral throat angle (∠2). For the intake bank, the results were unexpectedly much more varied, with the 27Nm configuration performing significantly worse on the critical seat lateral angle (∠2).

Figure 6-10 shows the deviations for the cone angle for both the exhaust and intake banks.

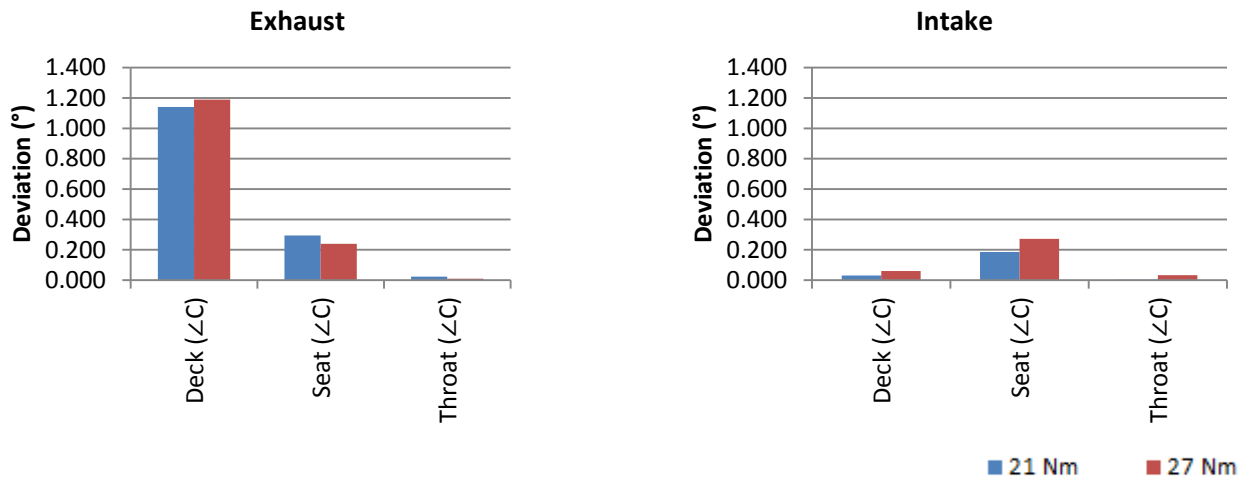


Figure 6-10 – Summary of deviations for cone angles (lower is better)

Overall, the cone angle measurements suggested that the increased bolt torque resulted in a slightly worse cone angle. Despite these results, the most significant observation was the comparatively very large deviation on the exhaust bank for the deck cone.

Cutting load can be reduced to normal and perpendicular components relative to the ZPL axial symmetry axis. For the exhaust bank, the cutting load is applied further away from the ZPLs than the intake bank, therefore the moment applied to the head and thus the displacement is greater for a given tool force. Furthermore, due to the orientation of the seats, the normal component of the intake cutting load on the ZPLs is compressive, whereas for the exhaust bank, the normal component is tensile. These factors combined make the exhaust bank significantly more sensitive to flexibility in the ZPL support system.

For this reason, the significantly higher deviation on the exhaust bank deck cone may have been due to a weakness in the ZPL support system which allowed the head to flex away from the tool as a result of tool force and therefore change the angle at which the seat was being machined.

## 6.4 Stiffness Analysis

A digital test indicator was used to measure the deflection of the cylinder head during machining. In rare cases it was observed that points on the top of the cylinder head were deflecting by around 40  $\mu\text{m}$ .

To better understand the flexibility of the cutting system, a representative finite element model was created in MSC Marc. The objectives of this model were to:

- determine the load required to reproduce the observed deflections of 40 $\mu\text{m}$ ;
- establish the relationship between cutting load and deflection;
- show the effect of 21 Nm and 27 Nm ZPL bolt torques on cylinder head deflection for a given cutting force; and
- characterise the twist of the head depending on the port and bank cut.

The model was verified using cutting force data from prior Ford tool force measurements and observed deflections during the real machining operation.

The following assumptions were made in order to simplify the model:

- the machine trunnion is rigid;
- there is only a touching contact between the tops of the ZPLs and the head (a touching contact resists penetration but allows the bodies to breakaway);
- the head is simulated without valves and guides to reduce the tool contact complexity;
- contact stress between the seats, guide and head are ignored as are all internal casting stresses within the head; and
- all materials are considered to be purely elastic.

Sixteen configurations were simulated as shown in table 6-3. The purpose of configurations 1 to 12 were to understand the twist of the head in response to every operation and the purpose of configurations 13 to 16 was to determine the difference between 27 Nm and 21 Nm ZPL bolt torques.

Configuration Ref	Bank	Ports	ZPL Torque (Nm)
1 – 6	Exhaust	1 to 6	27 Nm
7 – 12	Intake	1 to 6	27 Nm
13, 14	Exhaust	1 and 6	21 Nm
15, 16	Intake	1 and 6	21 Nm

*Table 6-3 – Simulation properties*

### 6.4.1 ZPL Preload

When there is no torque applied to the ZPL bolt, compressive forces on the support system are borne purely by the ZPL itself, whereas under tensile loads the stresses are borne purely by the M8 bolt. However, by introducing a preload to the bolt, the flexibility of the support system becomes a function of both the ZPL and the bolt stiffness as if they were one part. This is true until the tensile load on the system exceeds the bolt preload, at which point the cylinder head would break away from the ZPL and they would no longer be in contact.



The bolt preload was estimated using equation 6-1, where C is the coefficient of friction, D is the nominal bolt diameter (8 mm) and T is the applied torque.

$$F = \frac{T}{CD} \quad (6-1)$$

The coefficient of friction used was 0.3, based on Ford material specifications for the ZPL and bolt fixture assembly model (FORD, 2016c).

Table 6-4 gives the axial loading conditions for bolts under both torque configurations.

Bolt Torque	Estimated Axial Bolt Force (kN)	Axial Stress (MPa)
21 Nm	8.75	174.10
27 Nm	11.25	223.80

Table 6-4 – Table of bolt torques vs. estimated axial load and stress

The effect of bolt preload can be demonstrated by using a simple progressive loading finite element model to compare a bolted ZPL to a bolt-only variant. Figure 6-11 A) shows a section view of the ZPL – cylinder head bolted joint. B) shows the equivalent finite element model, indicating the boundary conditions and contact setup.

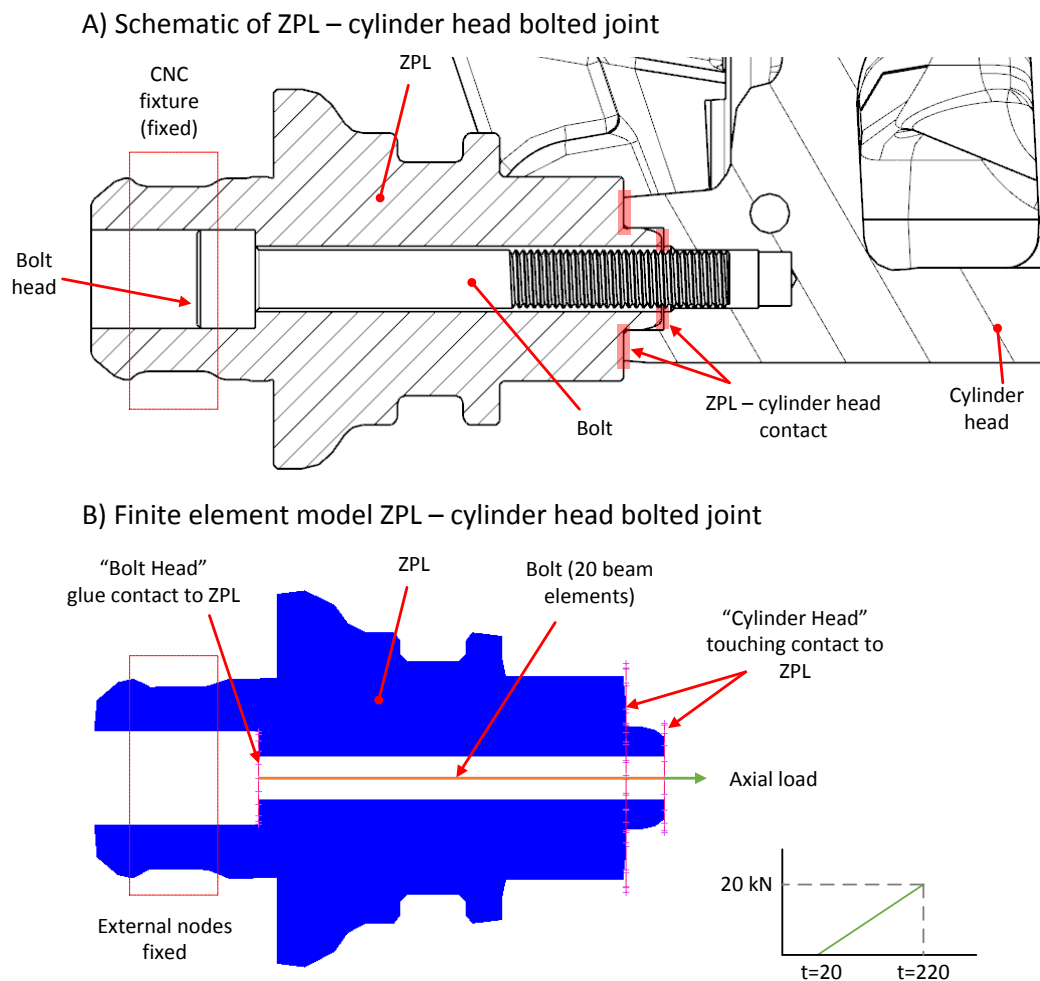


Figure 6-11 – ZPL-Bolt boundary conditions and contact



The system was loaded by a tensile load applied axially to the bolt gradually from 0 N at  $t = 20$  to 20 kN at  $t = 220$ . An initial stress condition (given in table 6-4) was applied axially to the bolt in order to include the preload effect.

Figure 6-12 shows a graph of the axial bolt strain vs. axial bolt load for the combined and bolt-only configurations. In each configuration, the final applied load on the bolt was greater than that of the bolt preload to ensure breakaway was observed.

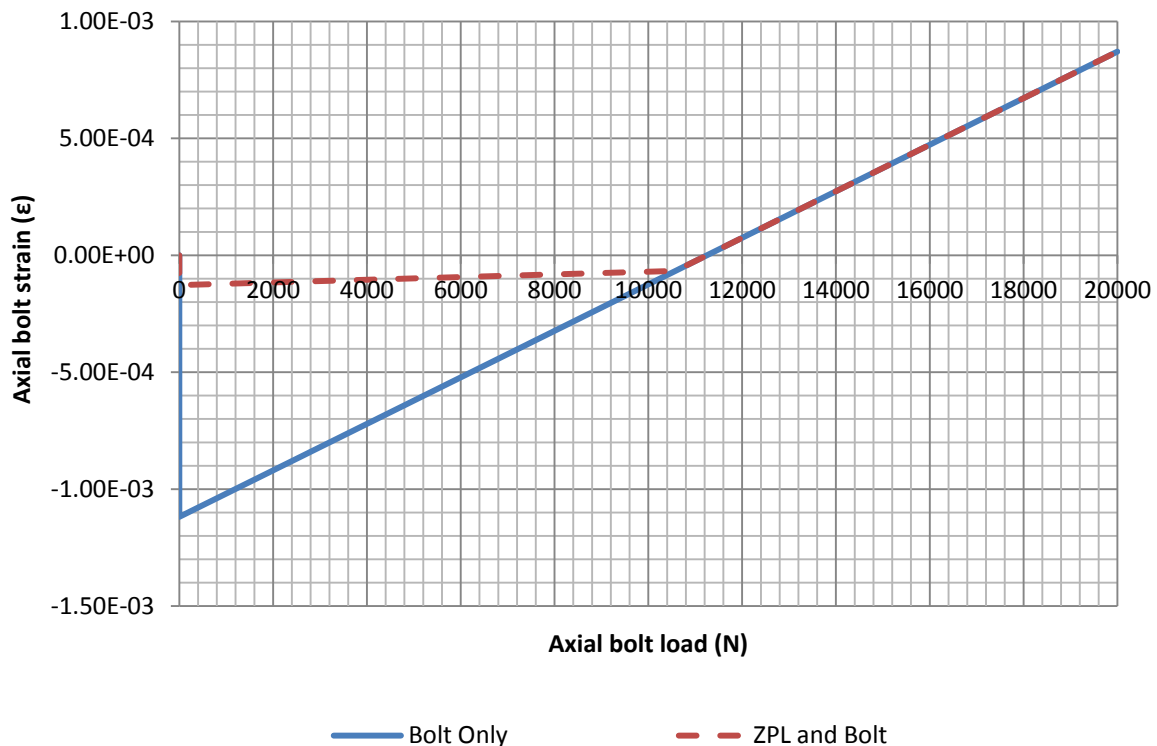


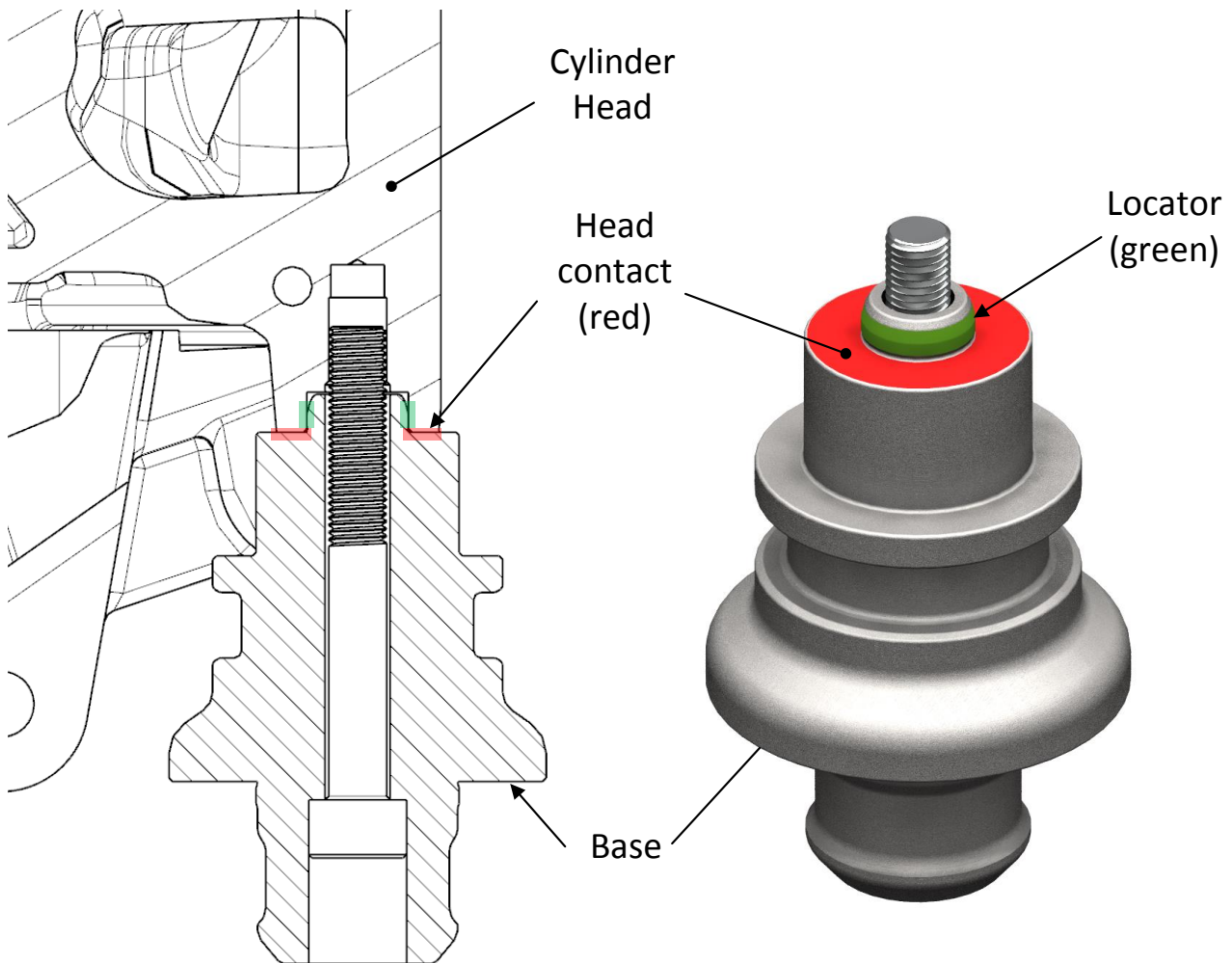
Figure 6-12 – 27 Nm axial bolt strain vs. axial bolt load for default and bolt-only configurations

As the figure shows, the final axial strain of the bolt was the same for both configurations. At the start of the simulation, the effect of the initial preload can be observed (applied in increment 1). The load increases until it reached 11.25 kN (the estimated bolt preload) at which point the strain rate increased. Up until the inflection point, the strain of the system was a function of both the ZPL and bolt. After the inflection point the strain rate was dictated purely by the properties of the bolt. In the bolt-only configuration (blue) the strain rate was constant throughout the simulation.

By preloading the bolt using a sufficient torque, it can be seen that axial strain (and therefore head deflection) can be kept to a minimum. However, if the preload selected is incorrect and the head breaks away from the ZPL, there will be a sudden change in strain rate during loading.

It would be extremely undesirable for break away to occur at any point during machining. Since the advance of the tool is position controlled as opposed to load controlled, the reduced stiffness that follows breakaway would create ideal conditions for chatter to develop.

Figure 6-13 shows a cross-section view of the ZPL – cylinder head bolted joint. As the tensile load on the bolt increases, the clamping force between the ZPL and head (red shaded area) decreases, therefore reducing the force required to make the head slip on the top of the ZPL.



*Figure 6-13 – ZPL Diagram*

#### **6.4.2 Geometry and Mesh**

A STEP format CAD file of the cylinder head was provided by Ford. The STEP file was imported into UGS NX10.0 as a collection of surfaces. Small gaps in the surface geometry were repaired manually using reasonable assumptions. The ZPL geometry was recreated in NX10.0 from an engineering drawing. Both the head and ZPL geometry were meshed in NX10.0 using a target mesh edge length of 13 mm. All seat and guide geometry was removed to simplify the model.

A mesh sensitivity analysis was performed to keep the element count low without compromising quality. Convergence against an equivalent 5 mm mesh was reached at a target edge length of 14 mm. Ultimately, a 13 mm target was used, as this gave better transitioning from the refined zones around the seat to the surrounding bulk of the cylinder head.

Figure 6-14 shows the cylinder head geometry and mesh at different element densities. The optimised mesh (left) consists of 239,268 elements (including ZPLs, bolts and head).

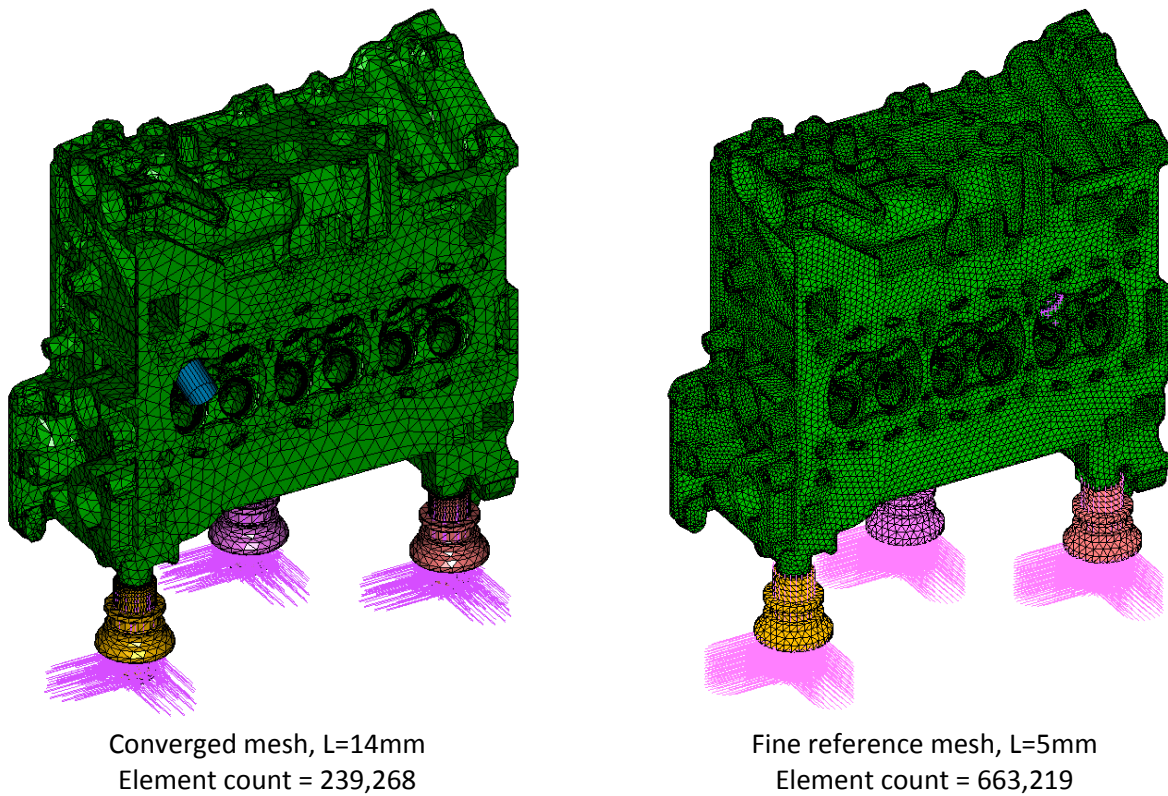


Figure 6-14 – Cylinder head and ZPL mesh showing ZPL boundary conditions

Figure 6-15 shows the bolt and ZPL contact geometry. All three ZPLs and bolts were modelled in the same way. The bolt was modelled as 20 beam elements (Marc type 98) with a diameter of 8 mm. There is no direct contact between the bolt and ZPL. The bolt had a glue contact condition to two rigid body surfaces at each end. One surface represents the bolt head which made a touching contact with the ZPL. The second represents the thread which made a glue contact with the cylinder head.

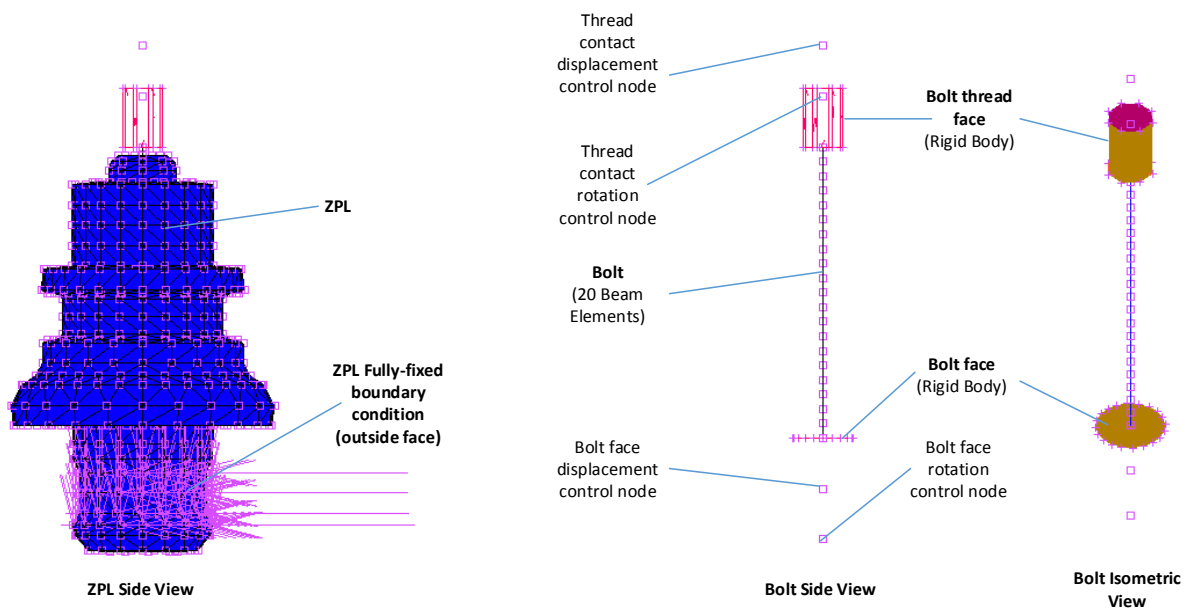
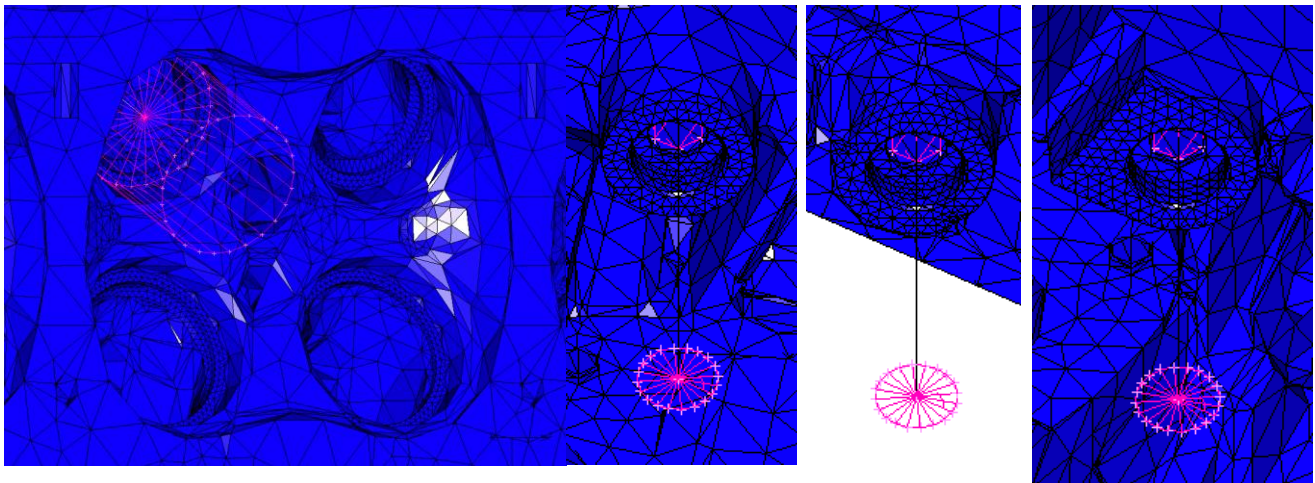


Figure 6-15 – ZPL and bolt geometry and contact

Mesh refinement was applied to all 12 seats to ensure a good contact performance in these areas as shown in figure 6-16.



Exhaust and Intake seat mesh refinement

ZPL contact (left)

ZPL contact  
(centre)

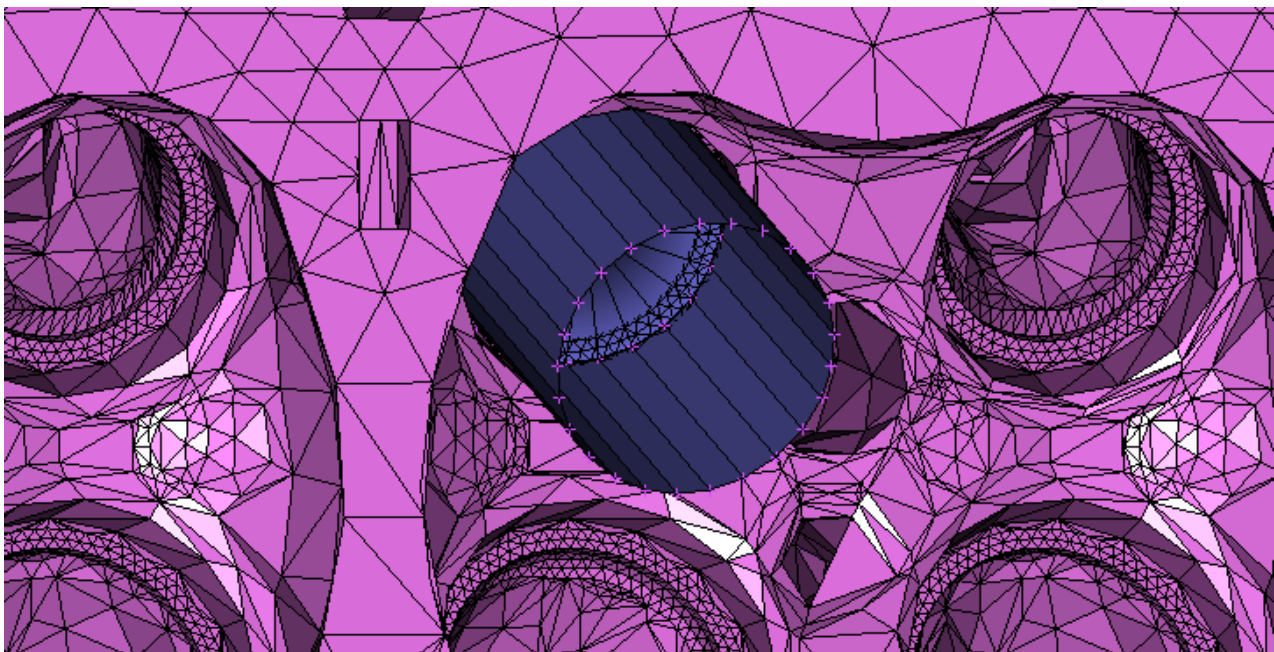
ZPL contact (right)

*Figure 6-16 – Head mesh refinement*

#### 6.4.3 Boundary Conditions

The ZPL was fully fixed on the lower surface to represent the interface with the CNC machine. An initial stress condition of 220 MPa was set for each bolt (calculated previously and given in table 6-4).

Only the thrust cutting force was considered for the simulation due to its dominant role in head deflection. The cutting load was applied to the head using a cylindrical position controlled rigid body which moved at fixed velocity as shown in figure 6-17. This method prevents the applied loads from following the head as it deflects, which is more representative of the real cutting system. Furthermore, this loading strategy allowed non-uniform tool pressure around the seat to be visualised wherever the head deflection was large enough to give rise to such an imbalance.



*Figure 6-17 – Tool force loading*

#### 6.4.4 Material and Physical Properties

Table 6-5 gives a summary of the mechanical properties used throughout all simulations. The physical behaviour of the bolt was modelled in accordance with ISO 898-1:2009.

Property	Cylinder Head (FORD, 2016b)	ZPL Body (FORD, 2016c)	M8 Bolt (FORD, 2016c)
Material	Aluminium (WSS-M2A178-A3)	Steel (16MnCr5)	Grade 12 bolt steel
Young's Modulus (GPa)	73.189	205.000	200
Poisson's Ratio	0.3	0.3	0.3
Coefficient of Friction	0.3 (Cylinder Head-ZPL contact)		
		0.3 (ZPL-Bolt contact)	

Table 6-5 – Simulation properties

Prior tooling calculations made by Ford estimated the total thrust force of the tool to be 392.8 N for the exhaust seat and 310.7 N for the intake seat, at a spindle speed of 1000 rpm and feed rate of 0.06 mm / rev (FORD, 2016a).

#### 6.4.5 Results

Table 6-6 shows a summary of the simulation results for the six exhaust machining operations modelled.

Ref	ZPL Torque	Bank	Port	Force required to deflect 40 $\mu\text{m}$ (N)		
				Left	Centre	Right
1	27 Nm	Exhaust	1	842	1109	1742
2			2	937	1114	1463
3			3	1060	1126	1269
4			4	1198	1139	1134
5			5	1399	1159	1015
6			6	1633	1181	934
7		Intake	1	1143	1642	3092
8			2	1328	1689	2487
9			3	1564	1734	2049
10			4	1850	1780	1779
11			5	2262	1812	1539
12			6	2865	1838	1362
13	21 Nm	Exhaust	1	805	1081	1759
14			6	1616	1162	916
15		Intake	1	1089	1595	3087
16			6	2858	1803	1332

Table 6-6 – Simulation results

As the cutter moved from left to right, the deflection increased from left to right suggesting that the head also twists on the ZPL support system. However it can be seen from the data that the tool forces required to induce such an extreme displacement in the head were generally high.

Figure 6-18 shows the cylinder head displacement at the top right of the head (using reference locations established earlier in figure 6-5). It can be seen that, the displacement was linear until the vertical blue line, after which the stick slip criteria allowed the head to slip on the contact surface of the ZPL (giving rise to the erratic force-displacement relationship beyond the blue line). The vertical red line shows the maximum realistic cutting load calculated by Ford (FORD, 2016a), therefore at no point within any realistic tool force did the head break away from the ZPLs, suggesting that the use of an M8 bolt is sufficient to prevent the head from breaking away under normal tool forces.

Displacements of 40  $\mu\text{m}$  were achieved at points on the top of the cylinder head at forces around 1000N. The physical response of the simulation showed good agreement with prior Ford simulations that show peak displacements of 19 to 23 $\mu\text{m}$  for forces of 400N (FORD, 2016a).

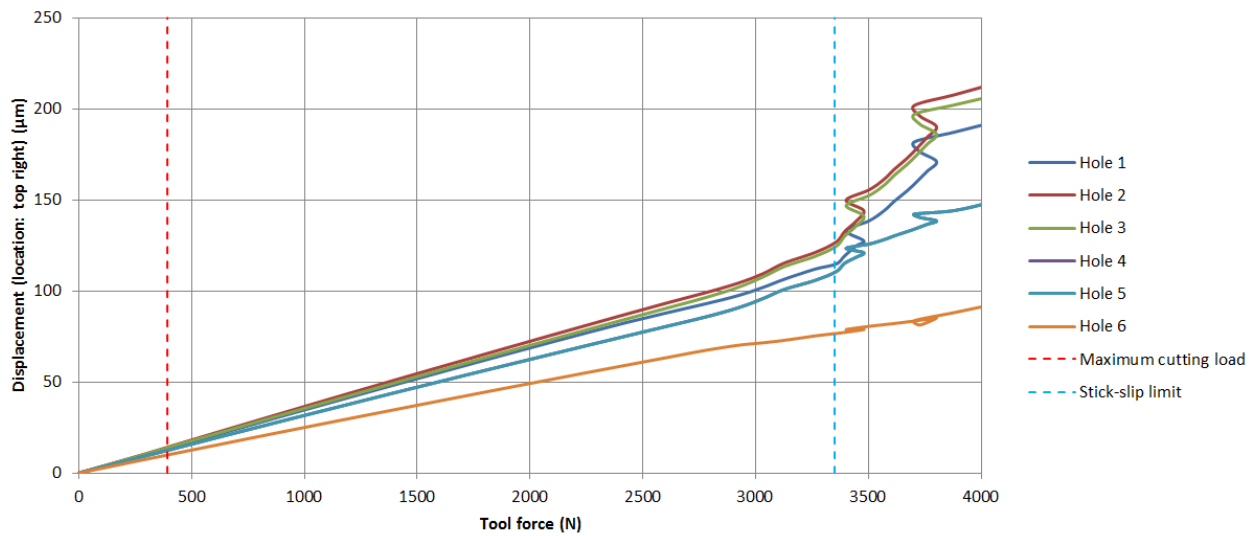


Figure 6-18 – Cylinder displacement at top right



Figures 6-19 and 6-20 show the head displacement simulations for all six ports on both the intake and exhaust banks respectively. Each image represents a different simulation, where the only variable is the port on which a load is applied. The images are given at 200 X deflection to help visualise twist in the head. The states are captured at the increment immediately after the tool force exceeds 2000 N (the maximum possible thrust force). The green arrow indicates the port and load direction for each simulation (unloaded ports are indicated by a white cross).

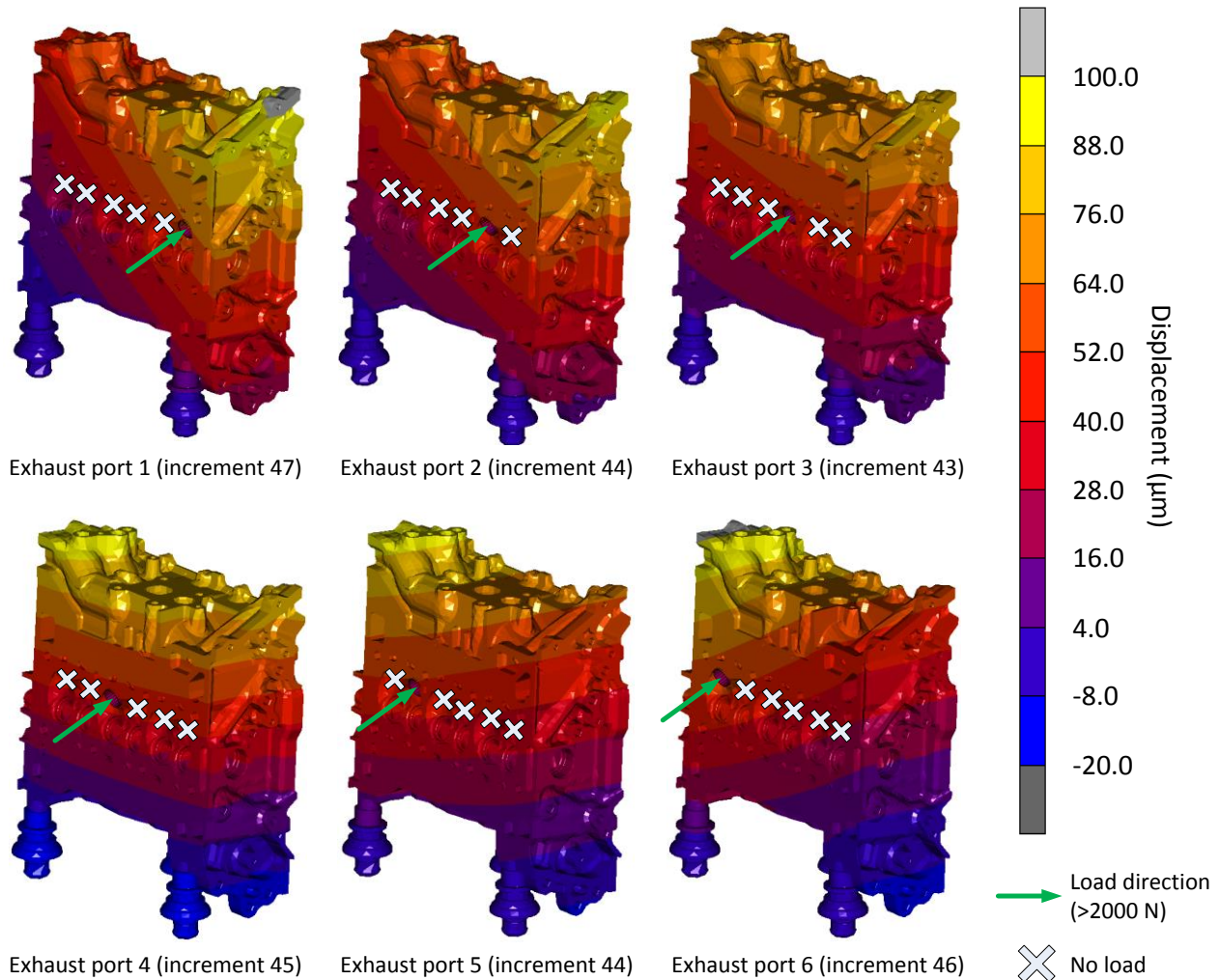


Figure 6-19 – Exhaust deflections (200X)

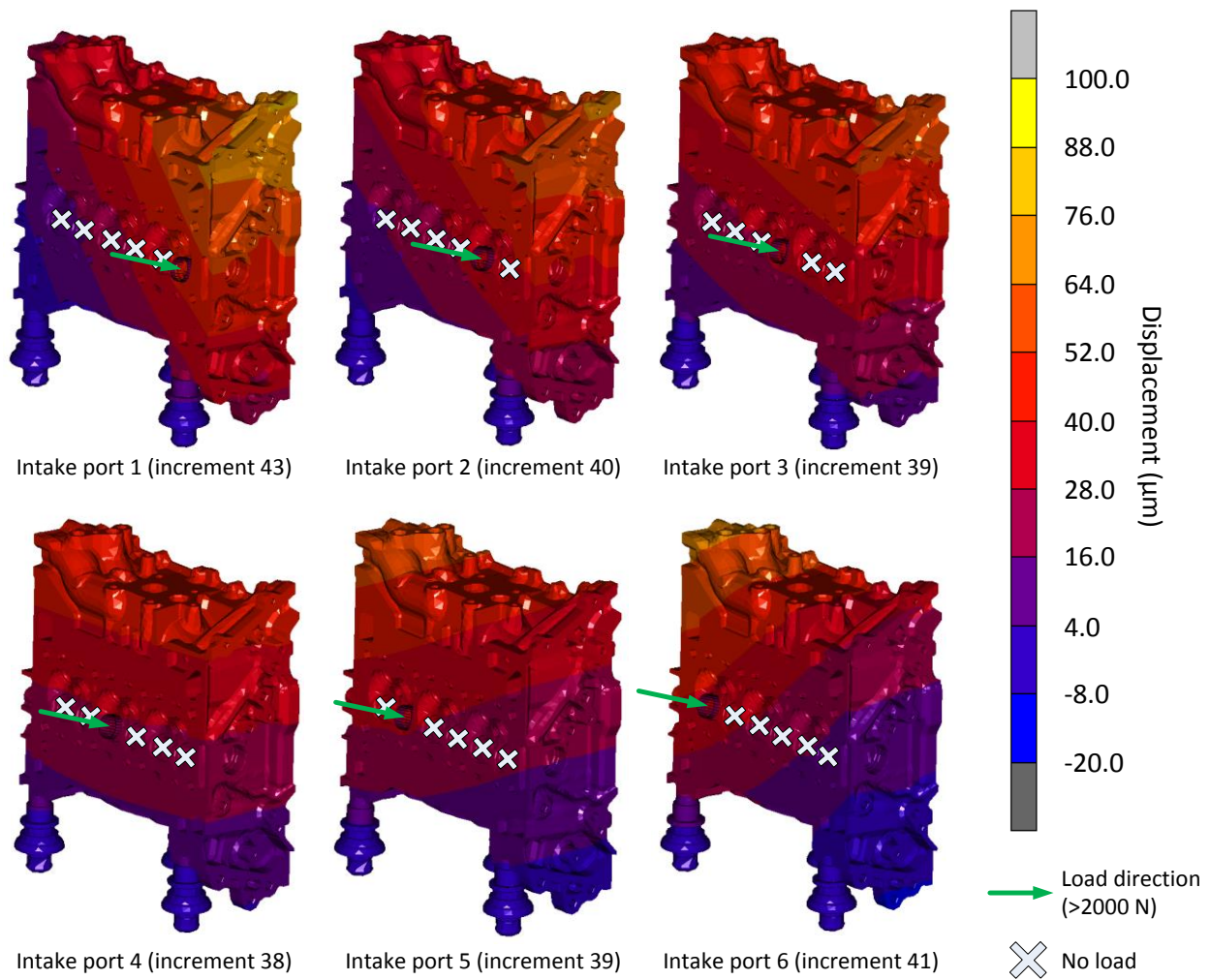


Figure 6-20 – Intake deflections (200X)

As the figures show, the twist of the head follows the loading position. Deflection when machining the intake ports was significantly lower than the deflection when machining the exhaust ports. Whilst both the exhaust and intake banks showed twist in the head, the effect is much more pronounced when machining the exhaust bank.



Figure 6-21 shows the exhaust deflections for ports 1 and 6 for both 27 Nm and 21 Nm, coloured according to the same scale colouring used in figures 6-19 and 6-20 previously. As the figure shows, there was no difference for port 6 and only minimal difference for port 1.

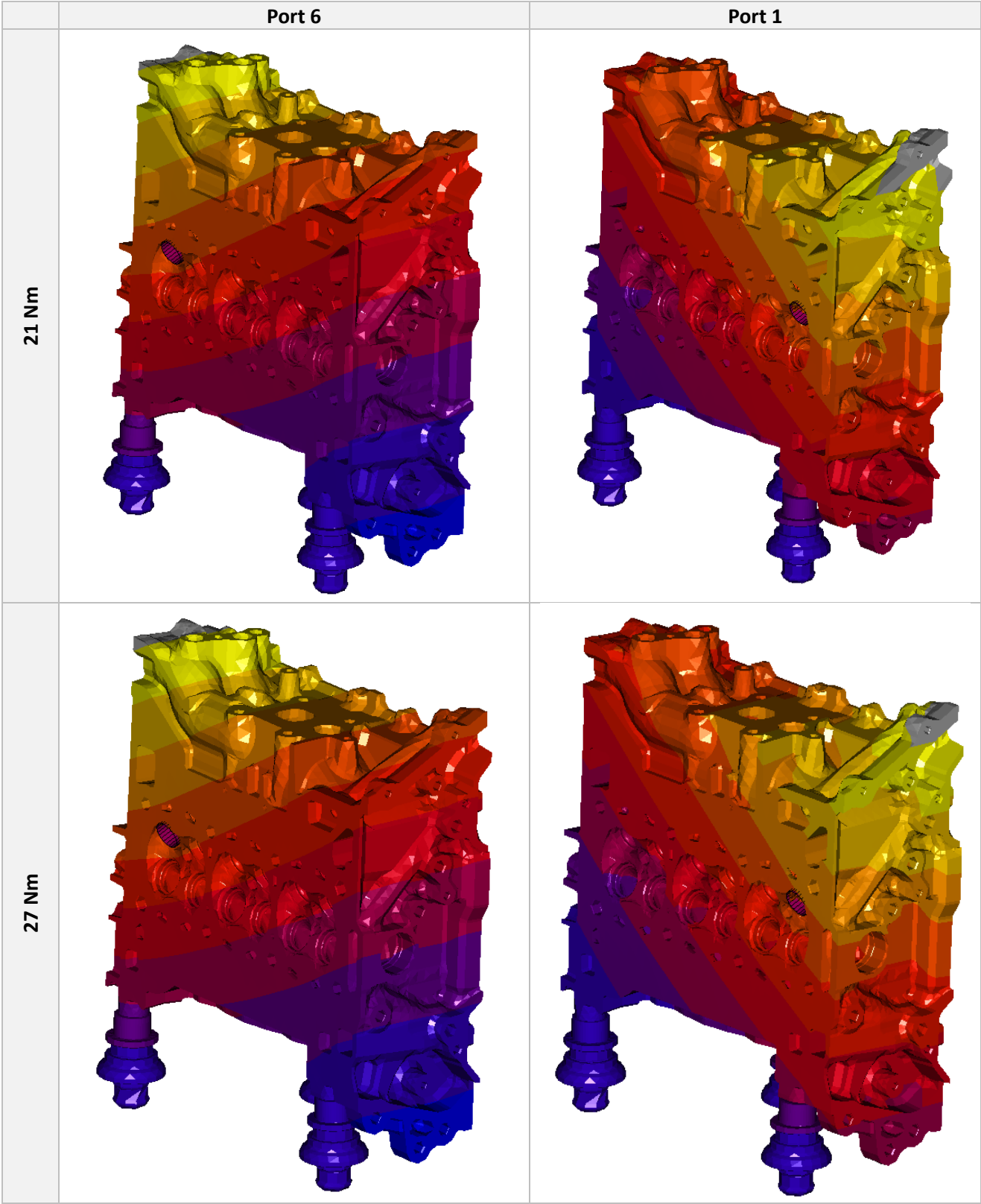


Figure 6-21 – Exhaust deflections, 27 Nm vs. 21 Nm

Likewise, figure 6-22 shows the same grid for the intake bank. Again, there was no significant difference between the different torque settings with regards to head deflection.

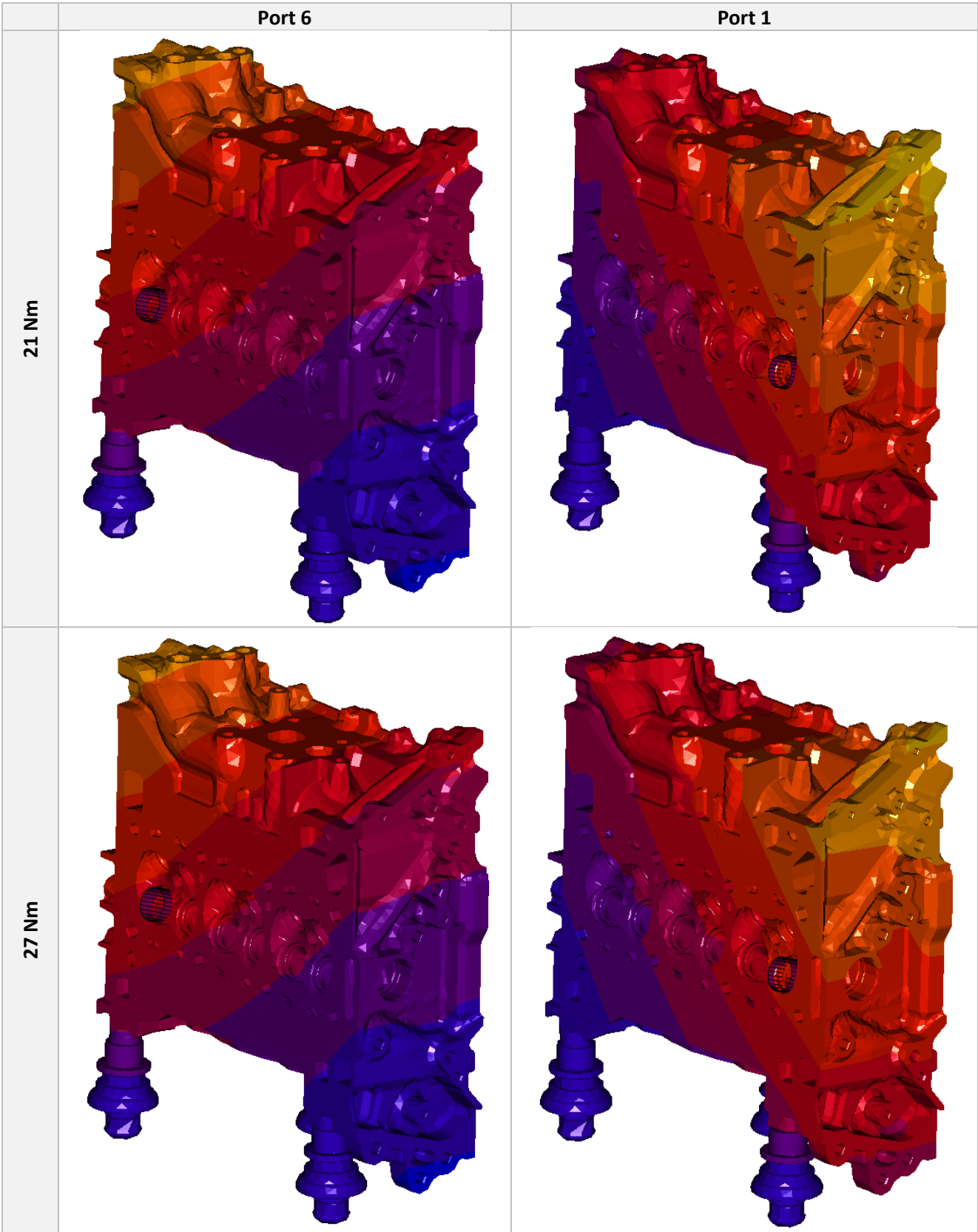


Figure 6-22 – Intake deflections, 27 Nm vs. 21 Nm

Overall the analysis shows that the ZPL torque made minimal difference to the magnitude of deflection within the possible range of cutting loads. The most significant factor in determining the deflection was the

cutting system geometry, specifically, the distance from the base of the ZPL to the region where cutting forces were applied.

No significant deformation was observed in the head structure itself and no part of the structure exceeded the true yield stress of the material. When the tool force was position controlled such that the head deflected 100 $\mu$ m, all points within the cylinder head were significantly below the yield strength of Ford standard WSS-M2A178-A3 aluminium. The simulation therefore indicated that the head is not likely to be undergoing any detectable plastic deformation during machining.

Figure 6-23 shows a close-up view of the head in un-deflected and deflected states. Deformation in the figure is pictured at 1000 X actual deflection. The colour bands on the plot represent equivalent Von Mises Stress in Pa. As the figure shows, the maximum deflection occurred around the rear ZPL. Stress peaks at the contact point between the rear ZPL and cylinder head. This ZPL is the only support structure on the rear of the head.

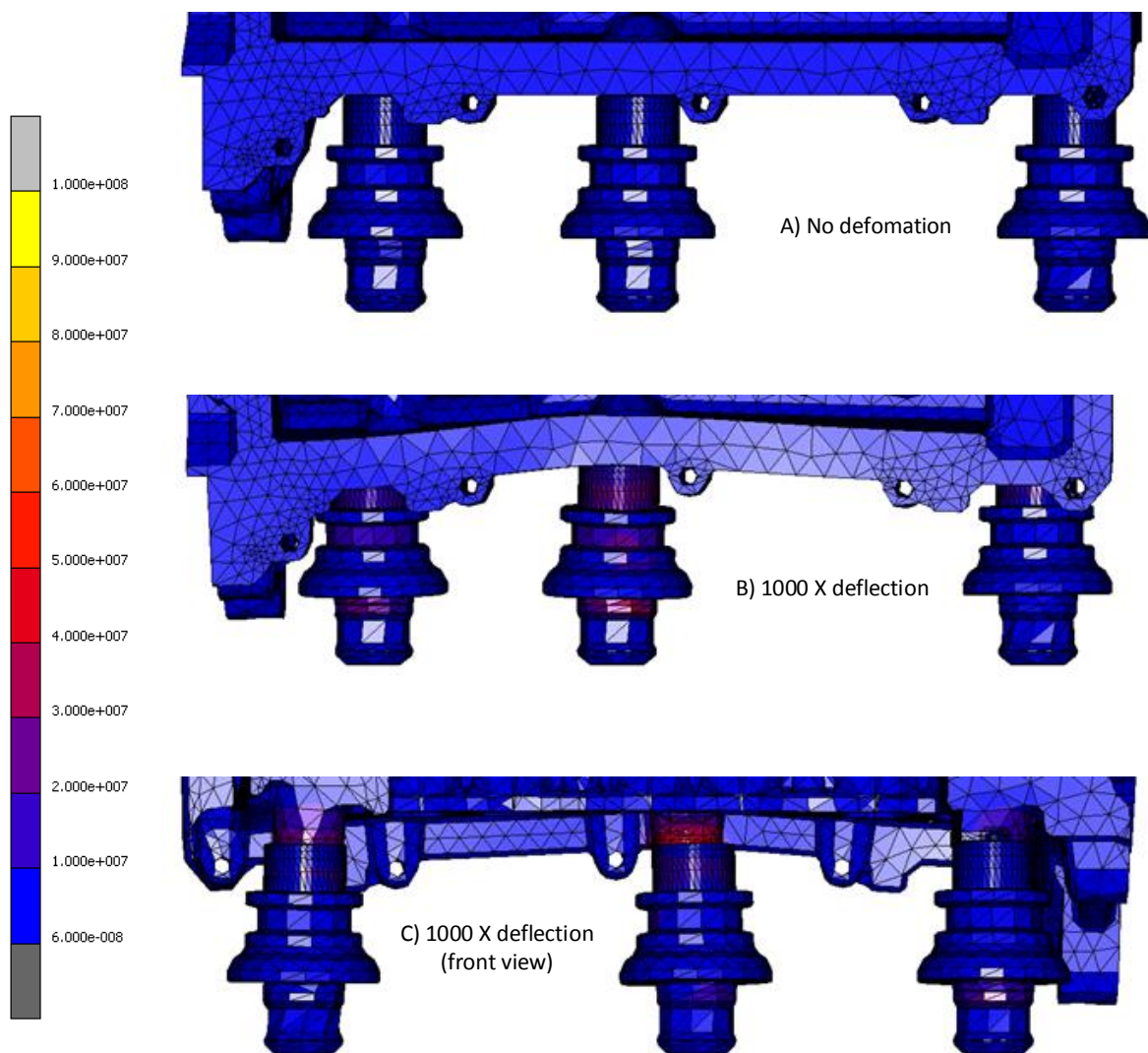
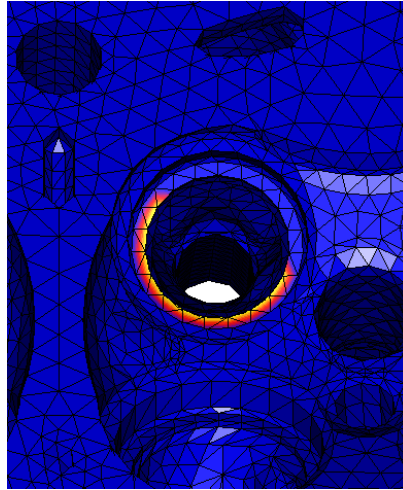


Figure 6-23 – ZPL support deformation and stress

The deflection in the head has helped to explain why some seats showed slight out-of-round error. Figure 6-24 shows the tool contact status for port 2 taken from the last increment (>2000N cutting load) of the 21

Nm exhaust bank simulation. As the figure shows, the contact was biased towards the lower left. This occurred as a result of the head flexing away from the tool due to the cutting load. Not only would this create an out-of-round error on seats, but it may also create an imbalanced radial cutting load which could lead to tool damage and chatter. However, the force required to produce this level of twist is significantly higher than the typical cutting forces observed.



*Figure 6-24 – Exhaust seat contact status on port 2*

Given the observed flexibility of the system in response to static thrust loading, it is concerning to consider that the radial cutting imbalance during valve seat machining can exceed 400N (Lacerda and Siqueira, 2012). This could create sufficient lateral displacement to initiate chatter or lateral vibration in the cylinder head which could damage the cutting inserts.

## 6.5 Cylinder Head Resonance Analysis

This section aims to look for and measure (if present) any resonance in the cylinder head, that arises as a result of inadequate stiffness of the cylinder head, fixture system or trunnion. If resonance can be identified at typical valve seat cutting speeds and feed rates, then tool breakages could possibly be reduced by altering feed rates and speeds. This section draws reference to a modal analysis simulation performed by Ford Motor Company that aimed to simulate flexibilities in these structures and calculate a series of natural modes in order to determine safe cutting speeds and feed rates. The first 10 natural frequency modes from that study are summarised in table 6-7. (FORD [Praveen. T], 2016).

Mode	Frequency (Hz)
1	261
2	306
3	354
4	393
5	453

Mode (cont.)	Frequency (Hz)
6	735
7	808
8	864
9	1101
10	1493

*Table 6-7 – Modal analysis of Fox GTDI*

### 6.5.1 Experimental Design

The frequency range of interest, 0 Hz to 1500 Hz intersects with what is considered as the low frequency range - 0.1 to 10 Hz. When structures resonate at low frequencies, they typically do so at high amplitudes (Wilcoxon, 2018). High amplitude oscillation of a structure like a cylinder head can cause severe damage

including warping of the parent metal. High amplitude oscillation within this range that causes relative motion between the tool and workpiece can also lead to significant out of round error, significant tool loading and misalignment of tool and workpiece that can create a bending load on tools such as drills and reamers.

A commercially available ADXL326 acceleration transducer was used to measure the surface acceleration on the cylinder head. The ADXL326 was selected due to its ideal bandwidth intersection with the range of frequencies, including low frequencies, anticipated in this experiment (0.5 to 1600 Hz) (Analog Devices Inc., 2009). The transducer was queried using a 12-bit analogue to digital converter (ADC) on an Arduino DUE. The data was sent via a serial connection to a Raspberry Pi where it was recorded to a comma separated values (CSV) file. The system was powered by an external battery to decouple it from factory mains noise. Figure 6-25 shows the accelerometer circuit diagram.

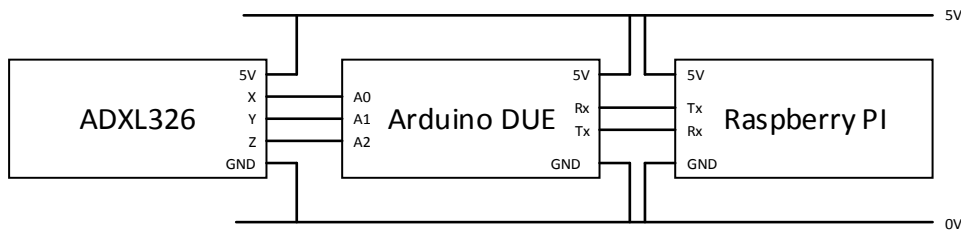


Figure 6-25 - Accelerometer circuit diagram

Bespoke code was written in Python to process and display the acceleration data. This code was used to isolate the cutting events from the overall acceleration data, apply calibration factors, calculate wave power and perform Fourier transform (FFT) analysis to extract the key vibrational frequencies and their magnitudes.

A validation step was performed to improve confidence in the timing of both the logging hardware and software. The accelerometer was attached to the cone of a loud speaker. A tone generator attached to the loud speaker was used to generate sine waves of various frequencies from 5 Hz to 1600 Hz. This range was selected to completely encompass the range of resonant modes shown previously in table 6-7. For each frequency, 10 seconds of data was logged and converted to the frequency domain using FFT. Figure 6-26 shows a typical FFT spectrum from this validation, in this case from the 200 Hz excitation frequency test. As the figure shows, the main response is sharply focused on 200 Hz, with decaying modes at 200 Hz intervals thereafter. There are no significant artefacts or unexplained responses in this spectrum.

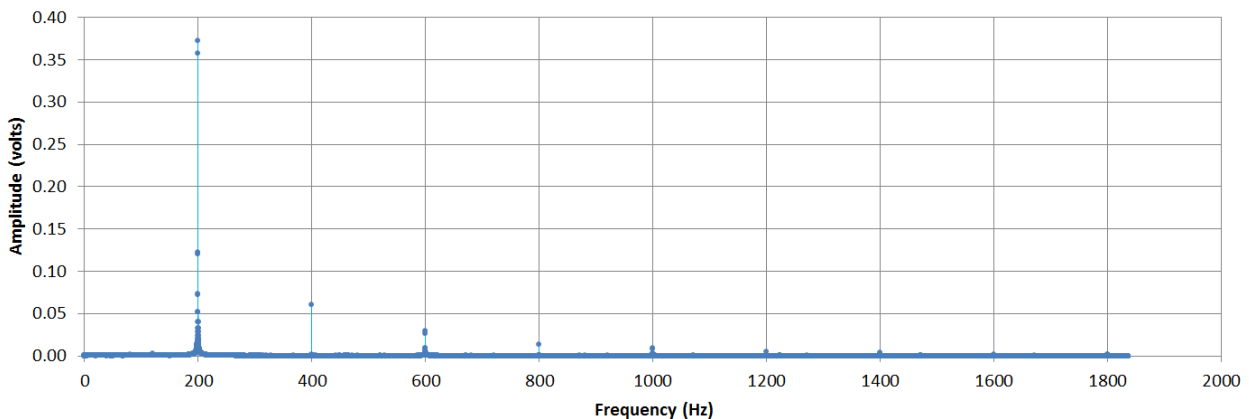
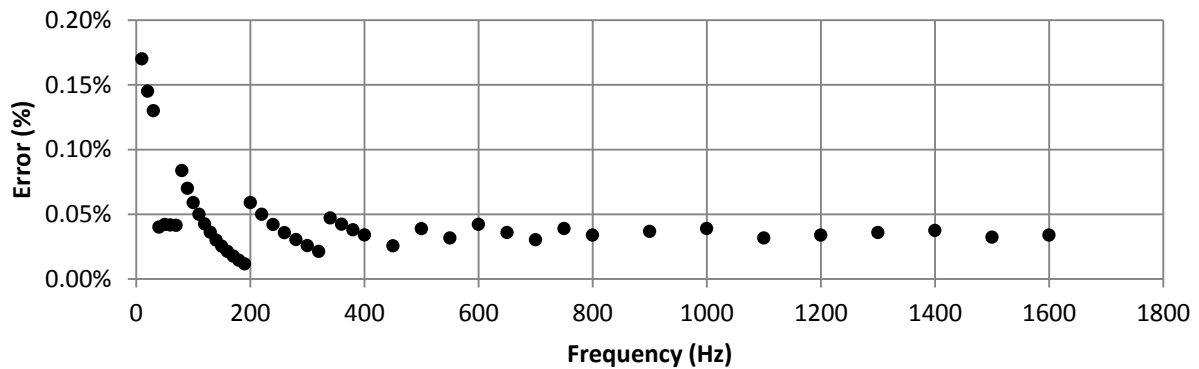


Figure 6-26 – 200 Hz FFT Spectrum



The calculated frequency was compared to the excitation frequency and the percentage error was calculated and plotted for each validation test frequency as shown in figure 6-27. As the figure shows, the maximum frequency error was 0.17% but generally below 0.05%.



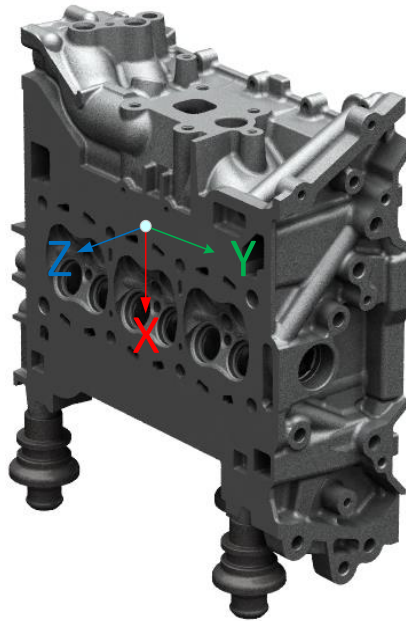
*Figure 6-27 – Accelerometer data logger frequency response error*

The accelerometer was attached to the cylinder head using a cyanoacrylate based adhesive at a location just above the intake ports as shown in figure 6-28.



*Figure 6-28 - Accelerometer attachment location*

The accelerometer data coordinate system is given in figure 6-29. This coordinate system and colour coding is used for all vibration data presented in the remainder of this chapter.



*Figure 6-29 - Accelerometer axis orientation*

The CNC machine used to perform the vibration study was a five axis MAG SPECHT 600 A/B. The machine used was a live production line machine and therefore represents the real production operation perfectly. The five axes of the machine are divided between the workpiece fixture and the tool spindle. The workpiece fixture can move in translations through Z and rotations about X and Y. The tool spindle can move in translations through Y and X. This division of axes helps to improve the overall stiffness of both the fixture and spindle. Furthermore, inertial excitations from translations through Z are clearly visible on the accelerometer data, but translations in X and Y are not picked up. This shows excellent isolation between the workpiece fixture and spindle.

Figure 6-30 shows the raw data collected from a typical valve guide and seat semi finish cycle. The graph shows the acceleration measured in each axis according to the coordinate system given in figure 6-29. Cycle 1 given in figure 6-6 is overlaid on the acceleration data for reference.

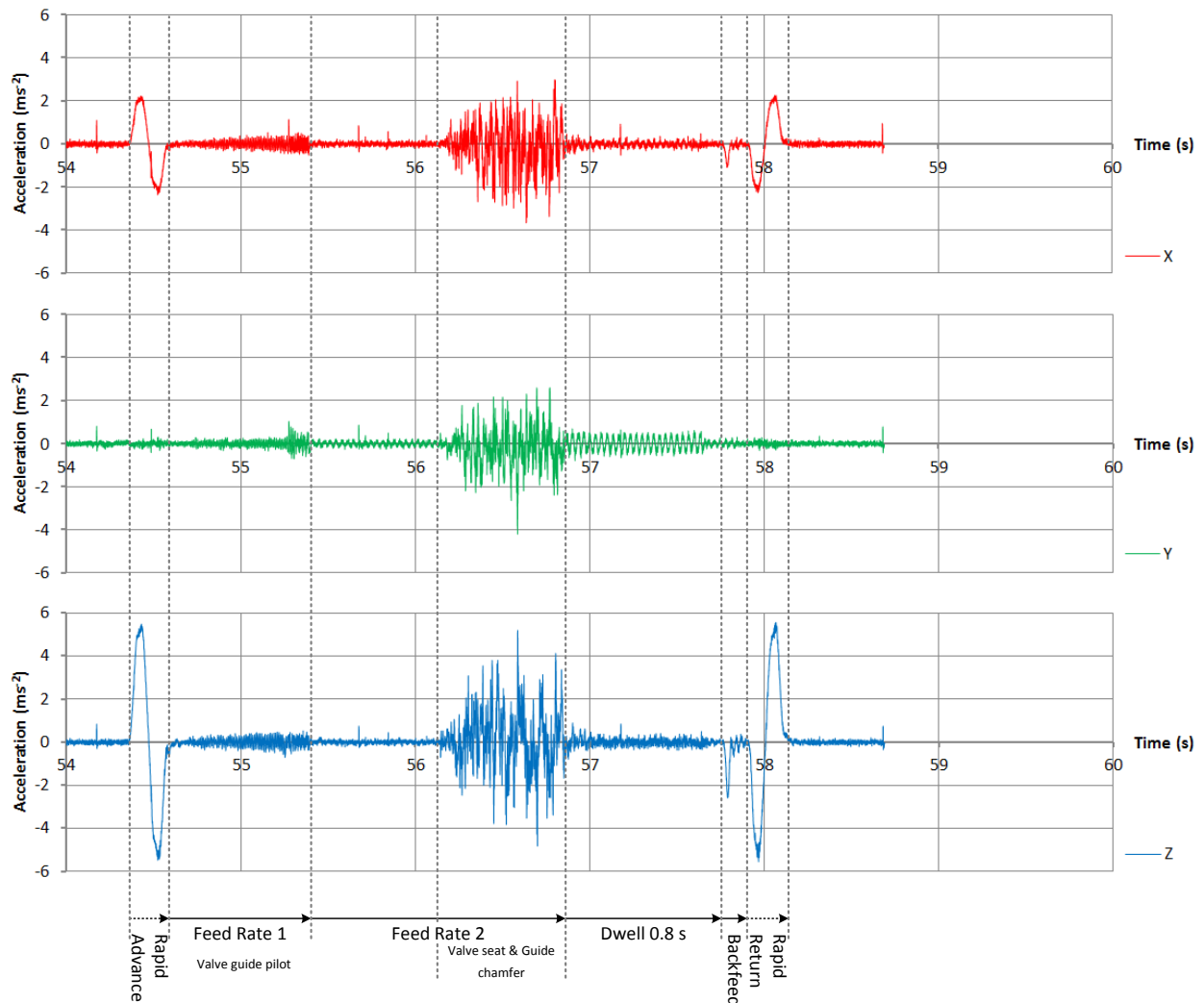


Figure 6-30 – Raw data from valve guide and seat semi finish cycle



Similarly for cycle 2, the valve guide reaming operation, a typical plot of the acceleration data is given in figure 6-31.

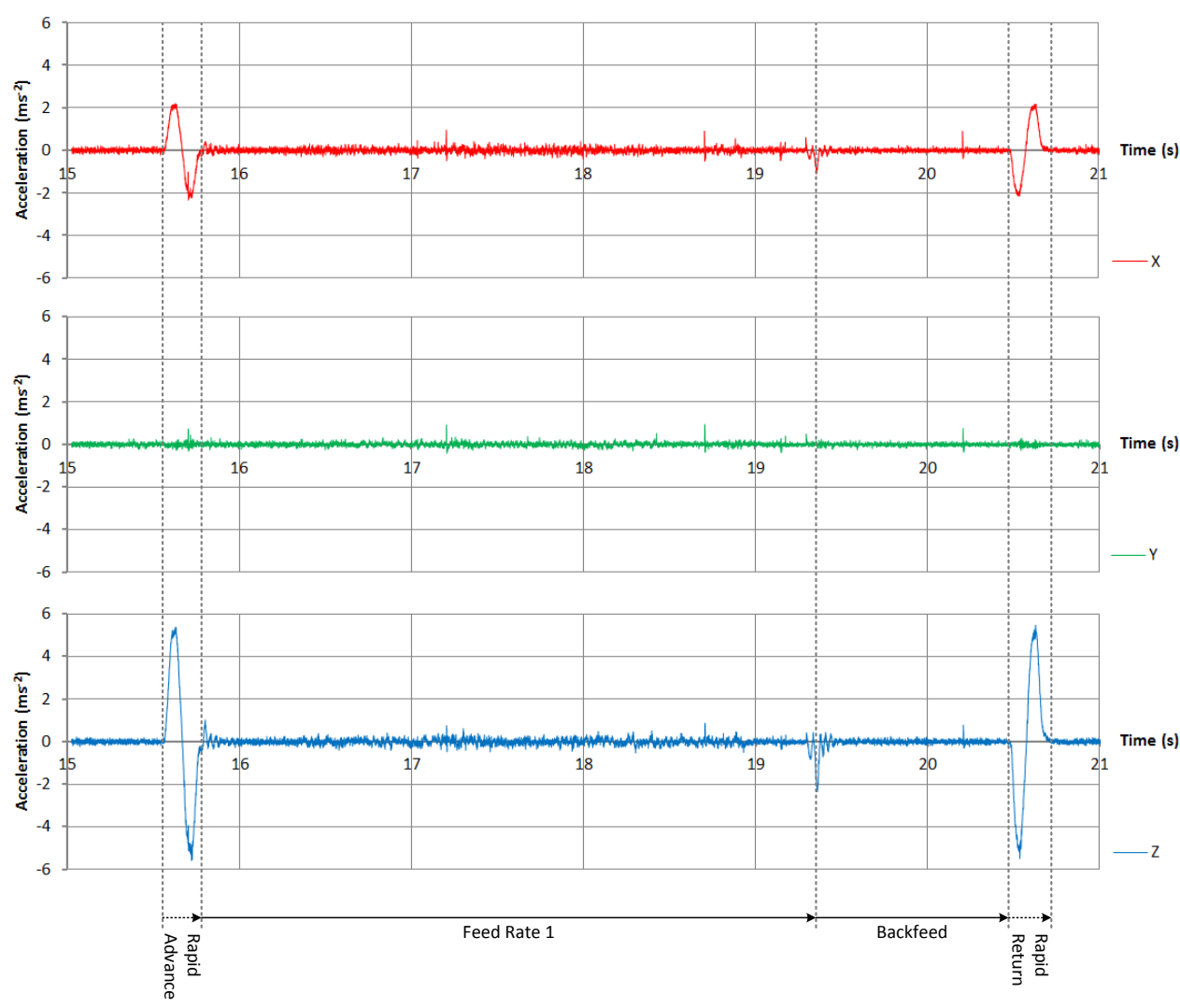


Figure 6-31 – Raw data from finish ream

### 6.5.2 Calibration

A calibration step was performed in order to determine a first degree polynomial relationship in the form of equation 6-2, that relates a measured ADC quantity,  $q$  (an integer division between 0 and  $2^{12}$ ), to acceleration,  $a$ , with units  $\text{ms}^{-2}$ .

$$a = mq + c \quad (6-2)$$

Calibration factors,  $m$  and  $c$ , were found through experimentation by adjusting the location of gravity (taken to be  $9.81 \text{ ms}^{-1}$ ) relative to the sensor, such that for each axis, two readings were taken, one with the relevant axis pointing towards the ground and the other with the axis pointing away from the ground. Minimum and maximum values of  $q$  recorded over a 0.2 second settling period corresponded to an acceleration of plus and minus  $9.81 \text{ ms}^{-1}$  respectively. The calculated calibration factors  $m$  and  $c$  are given in table 6-8 for each axis.

Axis	$m (\text{ms}^{-2} \text{ division}^{-1})$	$c (\text{ms}^{-2})$
X	0.0246	-50.1931
Y	0.0241	-49.0360
Z	0.0243	-51.0539

Table 6-8 - Acceleration calibration factors

### 6.5.3 Results

During the trials run, three failed reamers were observed. Figure 6-33 shows that last of these reamers embedded in a valve guide. This reamer failed on a 6000 RPM experiment according to cycle 2 shown earlier in figure 6-6.



Figure 6-32 – Reamer 3 failure

Figure 6-33 A) shows the typical acceleration data observed during a successful reaming operation at 4200 RPM. B) shows the same cycle, but at 6000 RPM. The feed period in B) was of noticeably higher energy than the same period in A). Before the end of the feed phase was reached the amplitude of vibration started to build, resulting in failure of the reamer, after which the reamer underwent a violent disintegration.

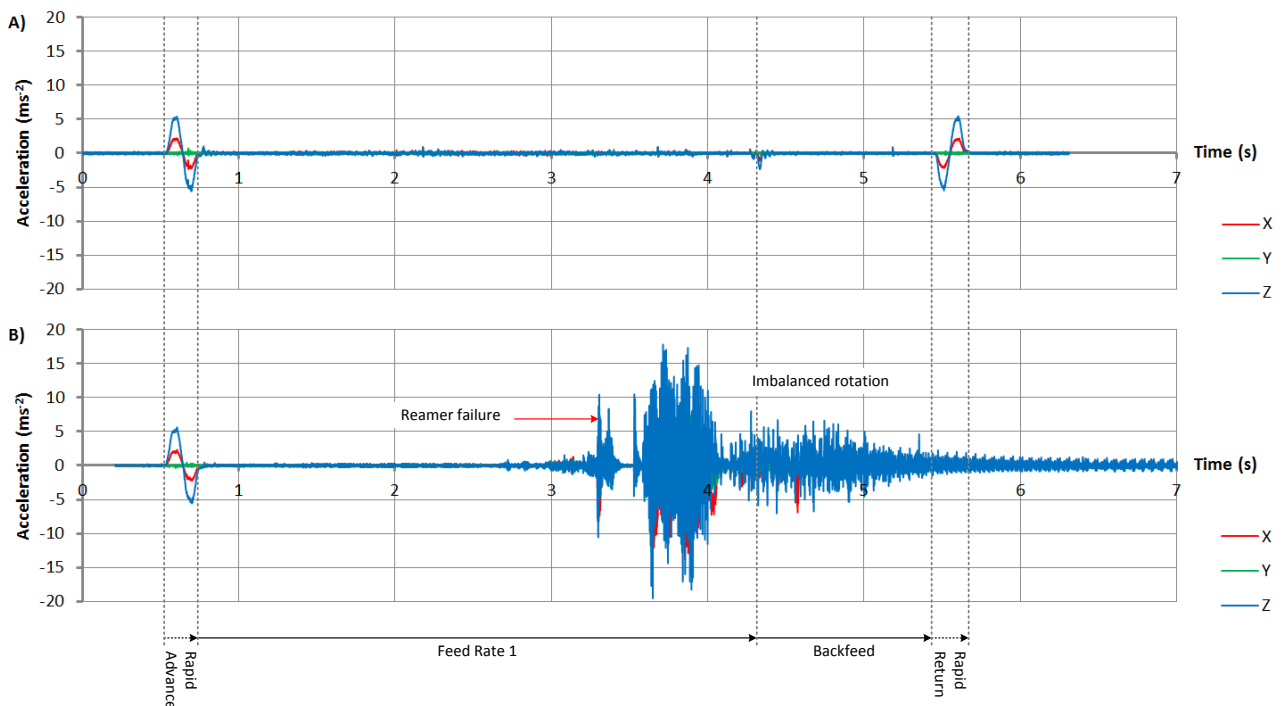


Figure 6-33 – Charts to show A) ideal reamer vibration data and B) failed reamer vibration data

Figures 6-34 and 6-35 show FFT spectra for a series of data immediately before two reamer failures set with the same cutting parameters. As the figures show, there are regular resonant peaks at multiples of 100Hz. In both experiments, the RPM was 6000, which corresponds to an excitation frequency of 100Hz. The strong peak at 200 Hz strongly suggests that the head underwent resonance at this frequency.

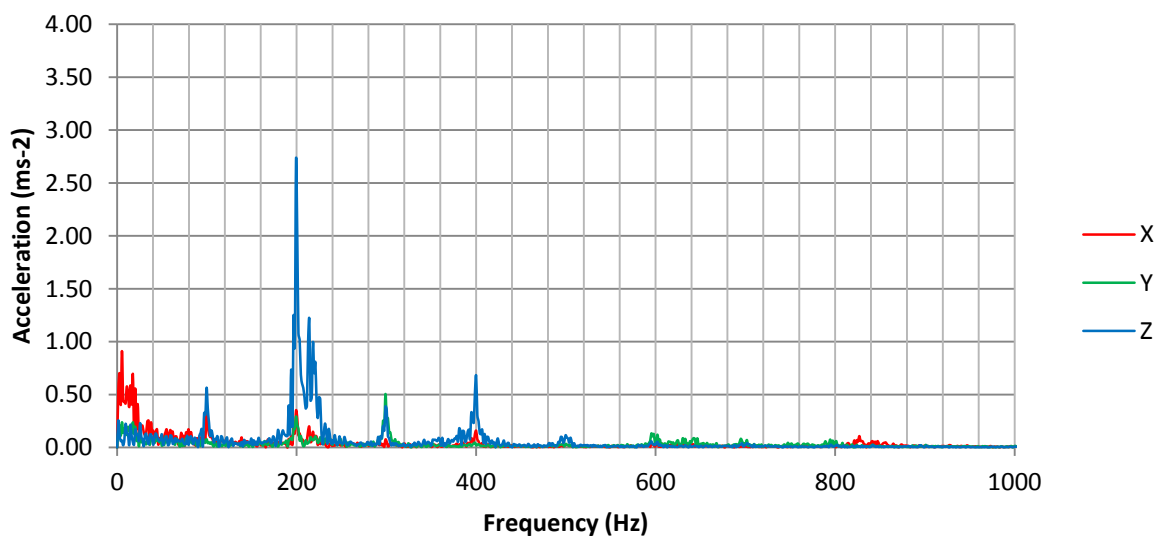


Figure 6-34 – Reamer 1 failure (real-positive FFT)

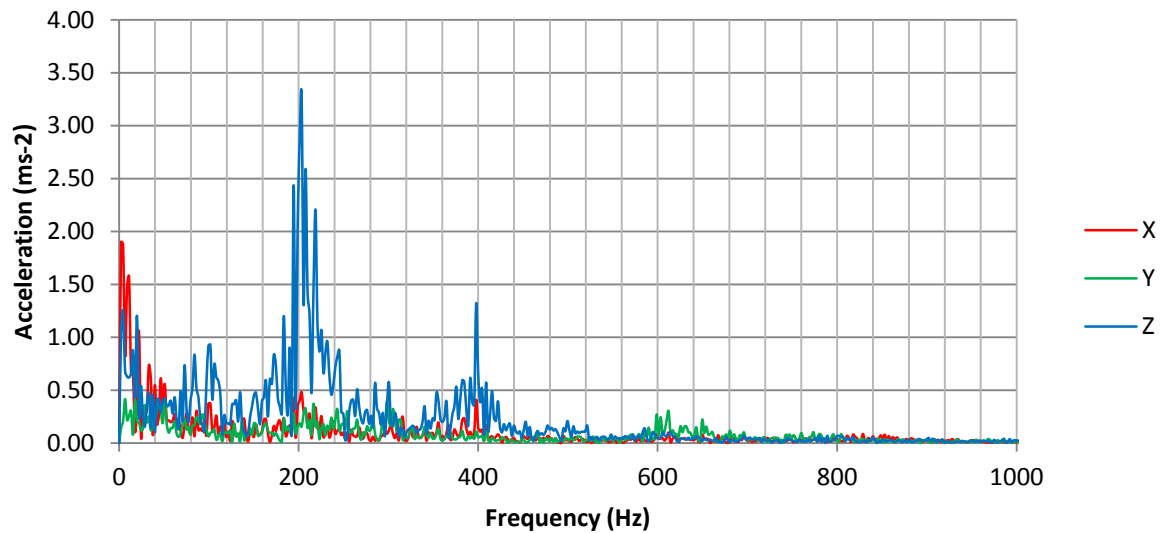


Figure 6-35 – Reamer 2 failure (real-positive FFT)

Both spectra exhibit a smaller peak just to the right of the major peak at 200 Hz, which may indicate the exact location of the resonant frequency in this range. The smaller peak is located at approximately 215 Hz, within 46 Hz of the first resonant mode of the head given earlier in table 6-7 on page 94 as 261 Hz. The 46 Hz discrepancy may be due to inaccuracies in the Ford modal analysis. Figure 6-36 shows the exploded geometry used in the Ford modal analysis. As the figure shows, fixed boundary conditions were used at the fixture trunnion, as opposed to the machine base (or some static component in-between). Furthermore the modal analysis did not take into account the mass of fluids (MQL and hydraulic fluid) and sub-assemblies (such as motors and hydraulic actuators) within the complex machine trunnion.

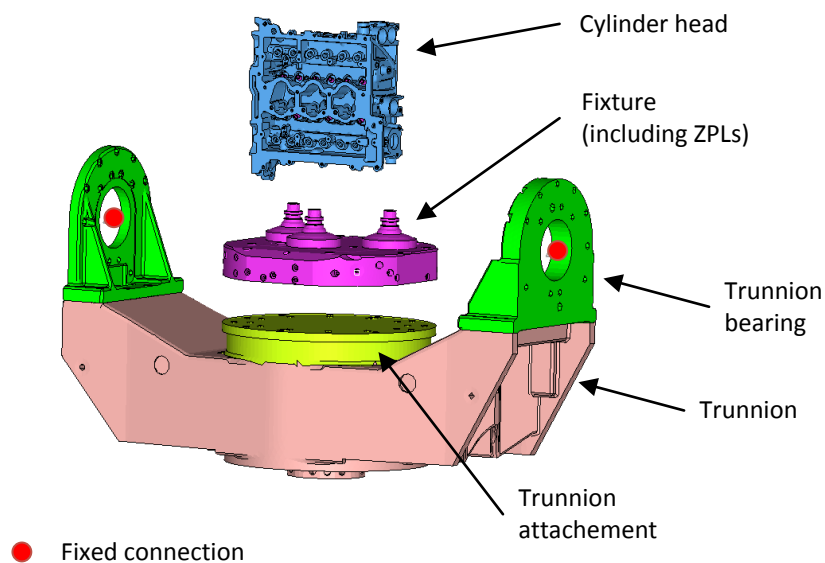


Figure 6-36 – Ford modal analysis, exploded geometry (FORD [Praveen. T], 2016)

## 6.6 Chapter Summary

Approximately 100 operations were observed containing a mix of valve seat finishing and valve guide reaming at various spindle speeds, feed rates and ZPL torque settings. No cutting inserts were observed to fail, however this was not unexpected since the pcBN cutting inserts will often survive hundreds of passes before randomly failing.

A stiffness analysis was used to show that in the worst case scenario, cutting thrust loads in excess of 800 N were required to deflect the head by 40  $\mu\text{m}$ . In most cases actual cutting loads were significantly below this at around 390 N. Deflections of 40  $\mu\text{m}$  were only observed in the most extreme cases and changes in feed rate and spindle speed were able to significantly reduce this.

Several reamers were observed to fail only when using spindle speeds of 6000 RPM. Resonant vibration was evident in the periods leading up to reamer failure. A strong resonant mode was observed at around 215 Hz, which agreed with a prior modal analysis which suggested a resonant frequency of approximately 261 Hz. Excessive high amplitude resonance in the head will cause the reamer and valve guide centrelines to deviate from one another, placing a bending load on the reamer.

Stiffness, structural and resonance issues were identified as possible causes of tool failure when machining valve seats. This chapter looked at the Fox cylinder head valve seat machining process to verify whether or not any of these phenomena are present and could explain cutting insert chipping. This chapter presented evidence to support the following conclusions:

- a stiffness analysis showed that for typical cutting forces, deflection of the head was minimal and unlikely to affect the cutting inserts in all but extreme cases. This theory is substantiated by the fact that insert chipping was also observed on the Sigma cylinder head which uses a rear mounted fixture plate;
- slipping of the head relative to the ZPLs was extremely unlikely given the exceptionally high tool force required to initiate a slip ( $> 3000\text{ N}$ ) compared to the relatively low maximum observed tool force (approximately 400 N); and
- no evidence of resonance was found during the valve seat cutting cycles. However, resonant vibration did occur in some finishing cycles leading to failure of valve guide reamers. This vibration is likely to have arisen due to the radial cutting imbalance in conjunction with the relatively flexible cylinder head.

# Chapter Seven – Experimental Characterisation of Cutting Force

---

$W(d)$	$m$	Width of cut as function of depth of cut
$w$	$m$	Width of cut
$U_f(f, \theta)$	$m$	Feed displacement of the cutter as a function of feed rate and absolute spindle angle, where $U_f(f, 0) = 0$ , is the point at which the cutter first touches on to the workpiece.
$u_f$	$m$	Feed displacement of the cutter where $u_f = 0$ is the point where the cutter first touches on to the workpiece
$V(d, f, d\theta)$	$m^3$	Volume of cut as a function of depth of cut, height of cut and a change in spindle angle
$D(u_f)$	$m$	Depth of cut as a function of the feed displacement of the cutter
$d$	$m$	Depth of cut
$t$	$s$	Time
$t_0$	$s$	Time zero, the time in force data representing the start of cutting
$t_{s1}$	$s$	Selection start time for calculating K values
$t_{s2}$	$s$	Selection end time for calculating K values
$f$	$m \text{ rev}^{-1}$	Feed rate
$\omega$	$\text{rad } s^{-1}$	Spindle angular velocity
$A$	$m^2$	Cross sectional area of cut, $A \approx d \times w$
$v_t$	$m \text{ s}^{-1}$	Volume weighted mean rake cutting velocity
$F_r$	$N$	Measured rake force
$F_f$	$N$	Measured feed force
$K_r$	$N \text{ m}^{-1}$	Specific rake force per unit width
$K_f$	$N \text{ m}^{-1}$	Specific feed force per unit width

## 7.1 Introduction

This chapter introduces and discusses an experiment aimed at determining the influence of lubrication regime, spindle speed and feed rate on cutting force and surface finish when machining AR20 high speed steel (HSS) valve seats using polycrystalline cubic boron nitride (pcBN) cutting inserts at typical production line spindle speeds and feed rates.

The data collected will be used to generate a prediction model for tool forces as a function of influential cutting parameters. The model will be used as an input to the numerical model developed in the final part of this work.

The advantage of a tool force model for use in a simulation such as the one proposed is that it removes the need to simulate the majority of phenomena presented in Chapter Two and only deals with the behaviour of the system as a generalized model on the scale of interest. This has benefits in that it reduces the number of models invoked and thus the number of experimentally gathered parameters required. It also reduces the computation burden since many of the models presented in Chapter Two require high element densities to function with any reasonable degree of accuracy (e.g. Thepsonthi and Özel, 2015, 3D finite element model of micro-end milling).

The key disadvantages however are the limited number of dimensions the model can support, for example, a change in flank angle would require a rerun of experiments presented in this chapter to gather new specific cutting forces. The model cannot tolerate any deviation outside of the limited number of independent variables that it uses. A further disadvantage is that this model cannot be used to predict chip

flow, geometry or breaking. In summary, the performance benefits of using a cutting force model outweigh the disadvantages when considering the aims of this work where it is more important to study the effects of cutter configuration on cutting load imbalance, than chip characteristics or the influence of cutter geometry.

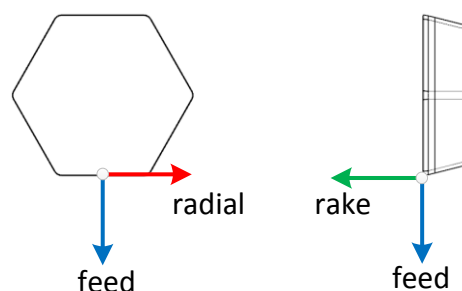
Bölling, Kuhne and Abele, 2017 applied a similar approach in their study of machining sintered steel valve seats with pcBN. They created a multivariate prediction model to calculate cutting force as a function of tool edge angle, cutting speed, feed rate, width of undeformed chip and pcBN cutting insert wear state measured in volume of material removed.

There is an economic and environmental interest in reducing the quantity of lubrication used during cutting, therefore this experiment aimed to determine the extent to which the lubrication regime has an effect on both rake and feed cutting forces. The two possible options tested were, dry (no lubrication) and minimum quantity lubrication (MQL). The MQL lubrication used is Castrol Hyspray A 1536.

A series of pcBN cutting inserts and AR20 HSS valve seat blanks were acquired for the experiment. The cutting inserts were all brand new, supplied in manufacturer packaging and had an initial edge radius of 30  $\mu\text{m}$ . The seats were made from a propriety sintered material which was not available in larger specimens. This cutting force experiment was therefore designed around the cylindrical valve seat geometry without any modification. Consequently, cutting was oblique, which was not ideal for analysis since this adds complexity during processing of the experimental data. Reasonable adjustments and approximations as described in this chapter were applied to the cutting force data to offer results that are valid in a universal context.

## 7.2 Theory

Figure 7-1 shows the local coordinate system for each cutting insert. For any intersection between the valve seat and cutter there would be a force applied in the rake and feed axes, normal and perpendicular to the rake face respectively. Ideally there would be no radial component relative to the cutting insert. In this experiment, the radial direction of the cutter was aligned with the radial axis of rotation, therefore there would be minimal radial component. For the full scale numerical model discussed later in this thesis, there will be a radial component relative to the central tool holder for any case where a cutting insert is not aligned with the tool holder axis.



*Figure 7-1 – Cutting insert local coordinate system, radial, rake and feed directions*

In order to produce generalised tool force prediction model, a mapping was created that related measurements taken from the oblique cutting system to equivalent values in an orthogonal system. In the physical experiment, cutting was oblique, there was a non-zero feed rate and a varying width of cut. In the simplified orthogonal equivalent, the material removed over a given time is represented by a simple

rectangular cuboid with zero feed rate. Figure 7-2 shows an example of this approximation for a segment removed after a partial spindle rotation,  $\theta$ .

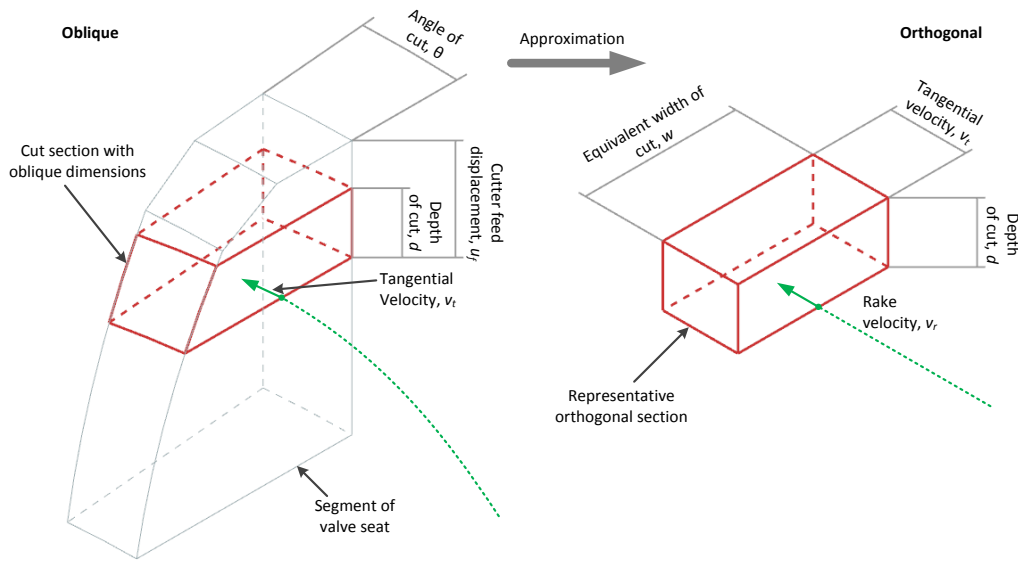


Figure 7-2 – Discrete cutting mapping from oblique (left) to orthogonal (right)

The dimension representing depth of cut is the same for both domains. In the orthogonal, the rake velocity is equal to the cutting tangential velocity at a radius that splits the cut volume into two sections of equal volume. Finally, a balance width is calculated such that the volume of the orthogonal approximation is equal to the volume of the oblique section.

Cutting forces are divided by this width to give the force per unit width parameters, as shown in equation 7-1.

$$K = \frac{F}{w} \quad (7-1)$$

To use the mapping as described, the simplified oblique model must be fully defined for any set of parameters (feed rate, RPM etc.) and for any condition likely to arise during simulation.



The coordinate system used throughout this chapter is defined according to figure 7-3. The degrees of freedom in the experiment were fixed in all translations and rotations, except for rotations of the workpiece about its axial symmetry axis and translations of the cutter through the feed axis. The cutter is pictured at zero feed displacement,  $u_f = 0$ , the point where the cutter first touches on the workpiece.

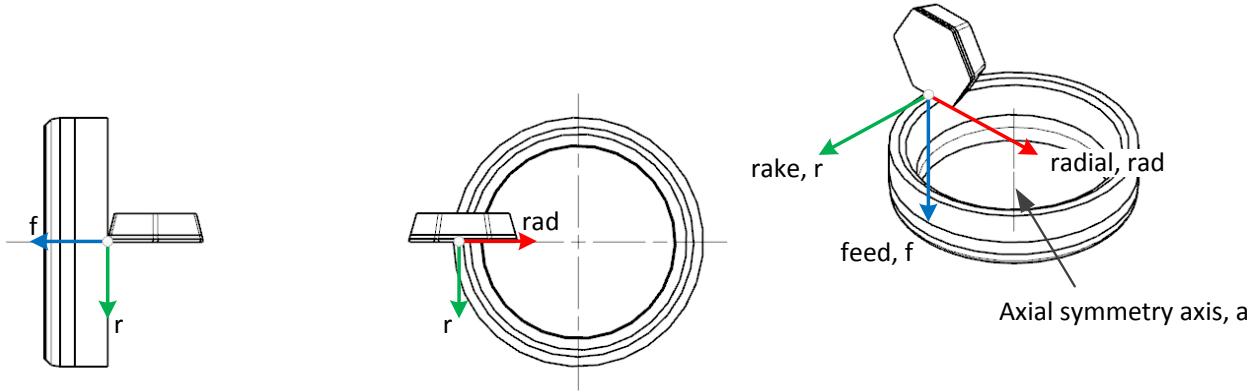


Figure 7-3 – Coordinate system definition

The displacement of the cutter is given as a function of feed rate and absolute spindle angle by equation 7-2 where  $U_f(f, 0) = 0$ , is the point where the cutter first touches the workpiece.

$$U_f(f, \theta) = f \frac{\theta}{2\pi} \quad (7-2)$$

The depth of cut is given as a function of cutter displacement,  $u_f$ , and feed rate,  $f$ , as shown in equation 7-3. This function is plotted in figure 7-4.

$$D(u_f, f) = \begin{cases} 0, & u_f < 0 \\ u_f, & 0 \leq u_f < f \\ f, & f \leq u_f \end{cases} \quad (7-3)$$

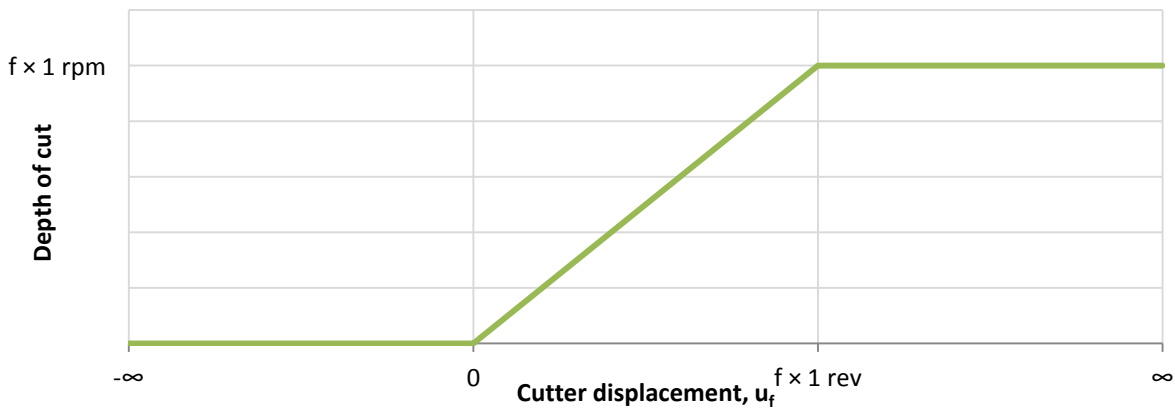


Figure 7-4 – Depth of cut as a function of cutter feed displacement

The width of cut was calculated analytically as a function of the distance offset from the initial contact point where cutting begins.

The valve seats are sintered from two powders of two different material compositions. Only the upper layer is machined during manufacturing. This layer extends at least 3.4 mm down from the top surface and the interface between the two materials is parallel with the base of the valve seat. For this reason, cutting data is only valid for the first 3.4 mm of cut. The experiment was designed such that this depth was not exceeded.

Figure 7-5 shows the dimensions of the exhaust valve seat. From these dimensions it was possible to derive a function that gives the width of cut for a given feed displacement of the cutter,  $u_f$ , as shown in equation 7-4.

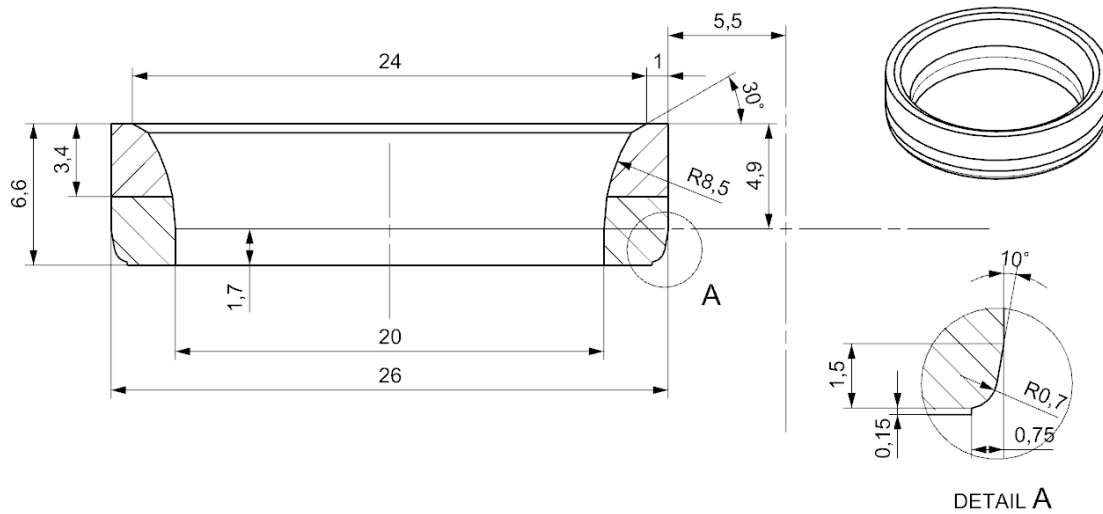


Figure 7-5 – Exhaust valve seat diagram

$$W(u_f) = \begin{cases} 0, & u_f < 0 \\ 1 + u_f\sqrt{3}, & 0 \leq u_f < 0.4165 \\ \sqrt{8.5^2 - (4.9 - u_f)^2} - 5.5, & 0.4165 \leq u_f \leq 3.4 \\ \text{invalid}, & u_f > 3.4 \end{cases} \quad (7-4)$$

Figure 7-6 shows the curve generated by this function when plotted to the maximum cutting depth.

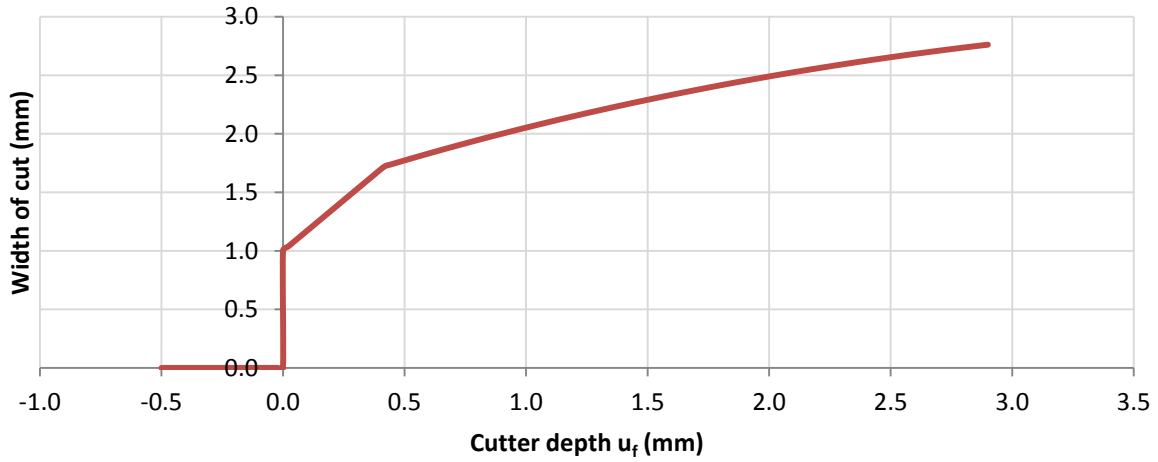


Figure 7-6 – Width of cut as a function of cutter feed displacement

The primary purpose of this function is to help calculate the volume of cut for a discrete segment of the valve seat. Each segment is removed as a result of the cutter moving through a helical path which adds considerable complexity to deriving an exact integral solution for segment volume as a function of depth and cutter angle. The segment volumes are more easily calculated as groups of discrete slices in translations through, and rotations about, the workpiece axial symmetry axis.

The front-facing area of the valve seat during a cut is approximated by the polygon MNPO as shown in figure 7-7, where MN is a straight line. Two areas, bounded by the workpiece axial symmetry axis and the lines MN and OP respectively are defined in preparation for integration.

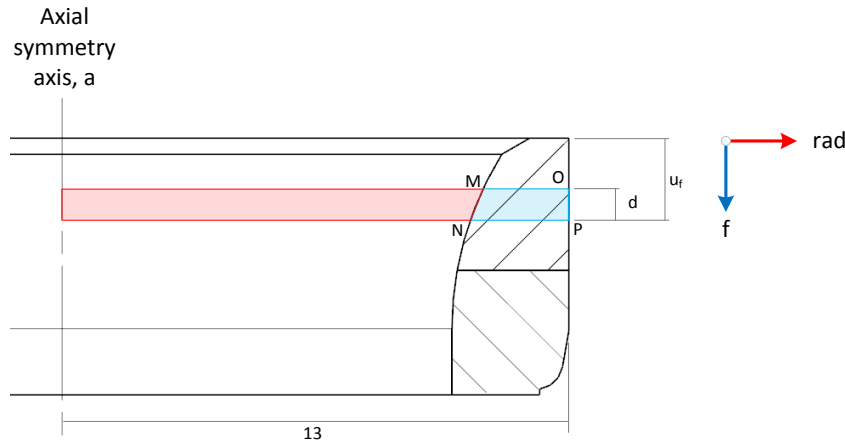


Figure 7-7 – Discrete area diagram for volume calculation

Let  $w_{MO}$  and  $w_{NP}$  define the width of cut for the horizontal lines MO and NP at their respective depths of cut as shown in equation 7-5, where  $d$  is the depth of cut as shown in equation 7-6.

$$w_{MO} = W(u_f), \quad w_{NP} = W(u_f - d) \quad (7-5)$$

$$d = D(f, \theta) \quad (7-6)$$

Thus, equations for the vertical lines OP and MN are expressed as shown in equations 7-7 and 7-8 respectively.

$$OP(f) = 13 \quad (7-7)$$

$$MN(f) = \frac{w_{MO} - w_{NP}}{d}(f - u_f) - w_{NP} + 13 \quad (7-8)$$

The cross sectional disc area as a function of feed depth is given by equation 7-9.

$$A(f) = \pi[OP(f)^2 - MN(f)^2] \quad (7-9)$$

This formulation can be expanded as shown in equations 7-10 to 7-13.

$$OP(f)^2 = 169 \quad (7-10)$$

$$c_1 = \frac{w_{MO} - w_{NP}}{d}, \quad c_2 = -\frac{w_{MO} - w_{NP}}{d}u_f - w_{NP} + 13 \quad (7-11)$$

$$AE(f)^2 = c_1^2 f^2 + 2c_1 c_2 f + c_2^2 \quad (7-12)$$

$$A(f) = \pi[169 - c_1^2 f^2 - 2c_1 c_2 f - c_2^2] \quad (7-13)$$

Finally, an integral for the volume can be formulated from equation 7-14 as shown in equations 7-15 and 7-16.

$$V = \int_{u_f-d}^{u_f} A(f) df \quad (7-14)$$

$$V = \pi \int_{u_f-d}^{u_f} 169 - c_1^2 f^2 - 2c_1 c_2 f - c_2^2 df \quad (7-15)$$

$$V = \pi \left( -\frac{c_1^2}{3} f^3 - c_1 c_2 f^2 + (169 - c_2^2) f \right) \Big|_{u_f-d}^{u_f} \quad (7-16)$$

Equation 7-16 gives the volume of a full revolution about the workpiece axial symmetry axis. The integration can be broken into discrete steps to account for the pitch of the cutter path as it feeds into the workpiece. Equation 7-17 gives the volume of a segment revolved through discrete angle,  $\alpha$ .

$$\delta V(\alpha) = V \frac{\alpha}{2\pi} \quad (7-17)$$

The method is further discretised by taking slices through feed direction i.e. using smaller values of  $f$  instead of selecting an  $f$  value which is equal to the feed rate.

Using this method combined with a discrete angle,  $\alpha$ , of 0.1 *radians* and a maximum discrete feed step of 0.02 *mm*, gives convergence to the exact solution (as computed using NX10.0) to within 0.2%.

### 7.3 Experimental Methodology

The experiment was performed using a Harrison M300 lathe. The lathe was partially controlled by an ABB IP20 three-phase Variable Frequency Drive (VFD). The VFD allowed the spindle speed to be finely adjusted after setting the approximate range using the appropriate gears on the lathe itself.

Forces were measured using a Kistler 9257B dynamometer which has a range of  $\pm 5$  kN. The dynamometer outputs three signals which were fed into three independent Kistler Type 5015 charge meters, which each output a signal between  $\pm 5$  V proportional to force. All three outputs were captured by a PicoScope 4424 USB oscilloscope and logged using PicoScope 6 oscilloscope data logging software.

A series of trial experiments were performed at a different range of feeds and speeds (designed to give the best spread of probable forces) to determine the ideal charge meter settings. The criteria for the maximum force selection (and therefore the output sensitivity) was that which provided the highest possible amplification for any cutting load in the experiment, but which did not exceed a 5 volt output, thus making maximum possible use of the oscilloscope analogue to digital converter (ADC) range. The figures in table 7-1 show the final charge meter settings which were used for all experiments.

Channel	X (radial)	Y (rake)	Z (feed)
Output range	+/- 5 volts		
Input sensitivity (pC / N)	-7.5	-7.5	-3.5
Max force (N)	500	1000	1000
Output sensitivity (N/V)	100	200	200

Table 7-1 – Charge meter settings

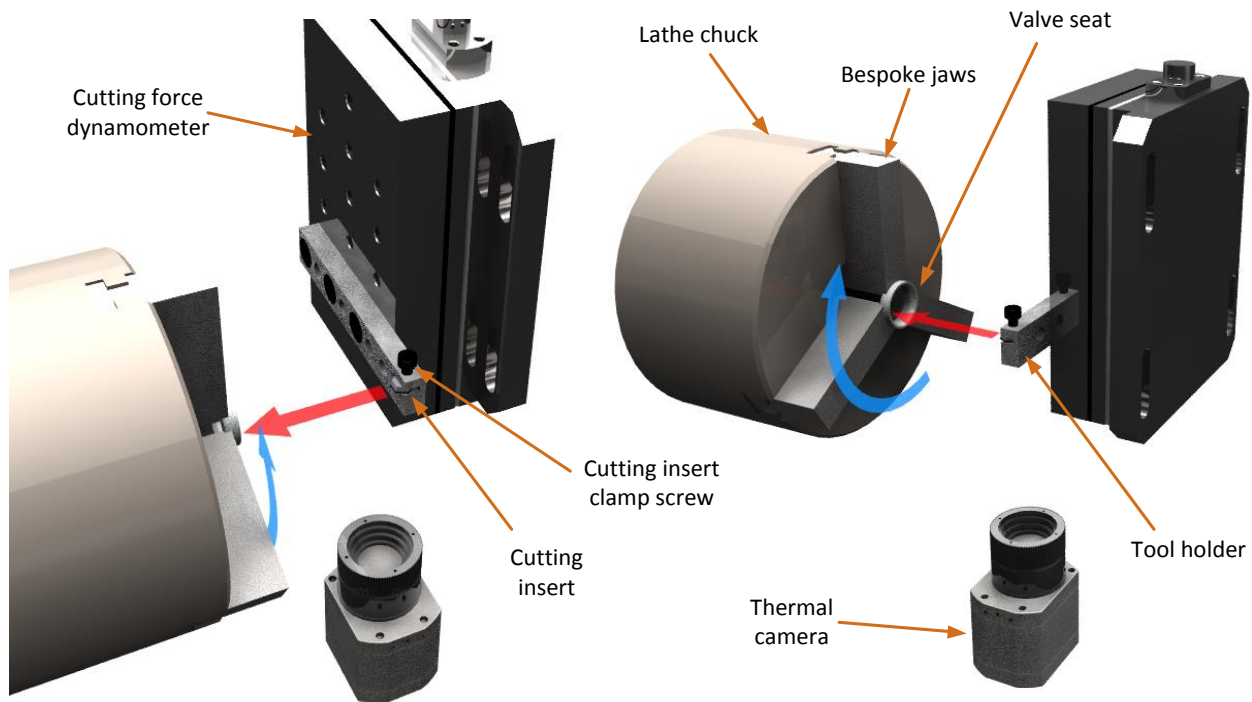
The X dimension represents the radial direction, therefore should not be subject to any significant net force. The input sensitivity was determined according to the Kistler 9257B datasheet. Any forces detected in this direction, other than expected vibrations, would have been possible indicators of alignment error between the cutter and the intended cutting plane of the valve seat.

The logging software was configured to log at approximately 80 kHz. This rate was selected to allow the experimental data to be used for vibration and resonance analysis in the event that chatter or other surface irregularities occurred.

A bespoke tool holder was built to hold the cutting insert and clamp rigidly to the dynamometer. In addition, a bespoke set of chuck jaws were machined to ensure the valve seat was clamped firmly and evenly around its diameter. This was particularly important, since the limited height of the seat and base radius presented very little surface available for clamping. Any unintended movement of the seat in the chuck jaws during machining could chip or damage the cutting insert or tool holder. The bespoke jaws also feature a flat-bottom surface on the inside to guarantee consistent flatness of the seats to speed up changes.

The valve seat was clamped in the lathe chuck and the tool holder was mounted directly to the dynamometer. This was in contrast to the production line configuration, where the valve seat is fixed throughout machining and the cutting inserts rotate. In both cases the relative motion between the tool and workpiece is the same.

Figure 7-8 shows the relative locations of the main experimental components including the cutting insert, the valve seat and the thermal camera. The figure also shows the direction of rotation of the lathe chuck (indicated by the blue arrow) and the feed direction and contact point on the valve seat (red arrow).



*Figure 7-8 – Experimental layout*

A SPI dial test indicator was used to assist in meeting or exceeding the geometric tolerances as defined in figure 7-10. The dynamometer alignment was set once for the entire experiment. The cutting insert alignment was set after each tool change and the seat total run out and front edge run out was set for every seat change. Figure 7-9 shows an example of how the dial test indicator was used to ensure proper alignment of a seat prior to machining.



*Figure 7-9 – Valve seat alignment check*

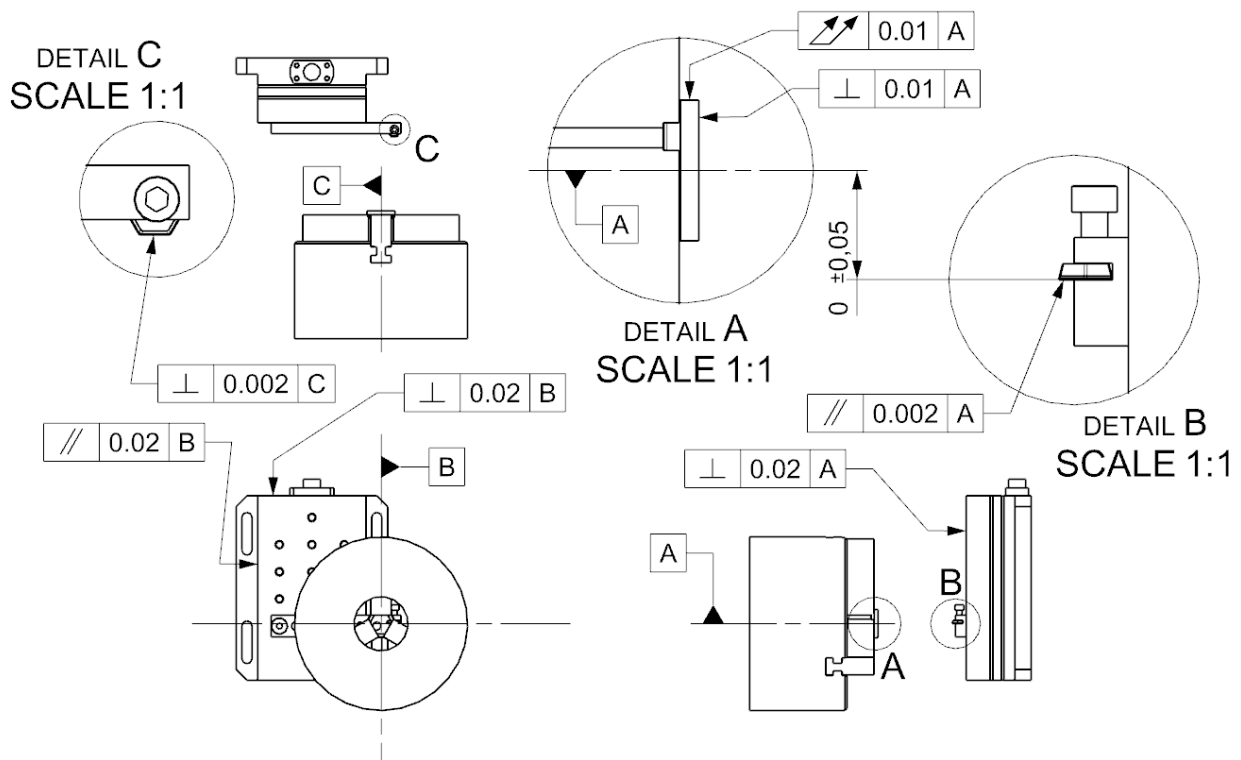


Figure 7-10 – Geometric tolerances

A light gate sensor tachometer was used in conjunction with a logic controller and LCD display to inspect the instantaneous RPM of the lathe during fine tuning as shown in figure 7-11 (c). The sensor (b) generated one pulse per rotation, which was fed to the RPM display (a) and oscilloscope to be recorded sequentially with the cutting force data. A recorded RPM reference allows the data to be corrected during processing to accommodate changes in speed due to loading conditions.

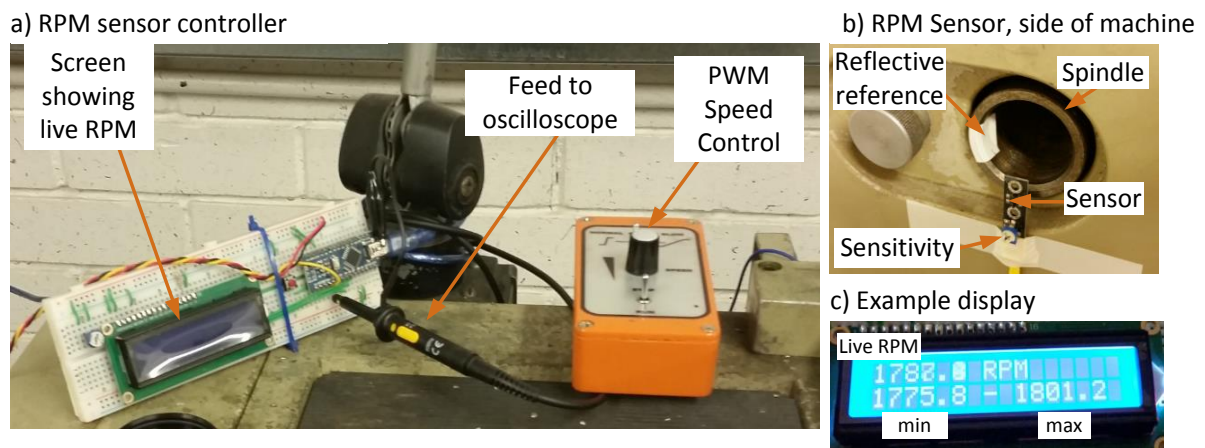


Figure 7-11 – RPM sensor

A Micro-Epsilon TIM400 thermal imaging camera was used to capture thermal images during cutting to show the temperature gradient across the cutting edge. The camera was also used to log the ambient room temperature for the experiment. The camera was attached to the lathe carriage between the bed bearings and aimed up at the cutting insert. This configuration ensured that the camera remained focused on the cutting insert and that the camera perspective remained constant, regardless of carriage movement during feeding and rapid withdrawal.

### 7.3.1 Parameter selection

All combinations of the parameters shown in table 7-2 were tested, totalling 32 cuts.

Speed (RPM)	Feed Rate (mm rev <sup>-1</sup> )	Lubrication
1200	0.03	Dry
1433	0.04	MQL
1667	0.05	
1900	0.08	

*Table 7-2 – Experimental parameter selection*

These parameters were selected to create a range that fully encompassed the full range of possible feed rates and spindle speeds used during multi-angle valve seat machining.

All dry cuts were performed before any MQL cuts to prevent contamination of the work area. A new cutter was used for the experiment, furthermore, the cutter was rotated to its opposite edge after completing the dry cutting set, such that all dry cuts were performed on a common edge and all MQL cuts were performed on a common edge opposite to the edge used for the dry cuts.

The MQL oil was applied manually using an atomising spray bottle to both the seat and cutting insert until both were saturated.

As well as the cuts listed above, two verification cuts, one for each dry and MQL sets, were performed at the end of each set at the initial 1200 RPM, and 0.03 mm rev<sup>-1</sup> feed rate. Those cuts were performed to set a benchmark to enable the cutter degradation to be measured and monitored as the experiment progressed. These additional data set also enabled independent validation of at least two other data sets recorded.

## 7.4 Data Processing

The aim of processing the data is to compute the specific force per unit width,  $K$  values, based on equation 7-1, for given cutting parameters such as spindle speed, depth of cut and lubrication regime.

All raw data were processed using a common algorithm written in Python. Each experiment yielded one thermal video and one comma separated values file containing columns for time, radial, rake and feed loads and tachometer state from the oscilloscope software. Processing was performed sequentially as described in the following sections for each data file.



### 7.4.1 Signal Conditioning

The files were read into memory by the script where the sensitivity factors for the charge meters were applied to each column to convert the recorded voltages to forces in Newtons according to the values given in table 7-1.

For the signal representing the tachometer input, the entire data file was stepped through to identify all points where the tachometer signal rises from a logic low to a logic high state, indicating a tachometer pulse corresponding to one rotation of the spindle. The time difference between each rising edge was used to calculate the rotational frequency and thus the RPM value as shown in equation 7-18 where  $t_n$  is the timestamp corresponding to the  $n$ th rising edge. These values were recorded against a time value that falls approximately half way between the pulses used to calculate it. Once all pulses were resolved to RPM readings, the remaining data points were assigned RPM values based on a linear regression between the calculated RPM readings as shown in equation 7-19 where  $RPM_t$  is the RPM at time  $t$ , and  $RPM_n$  and  $t_n$  are the  $n$ th calculated RPM and its timestamp respectively.

$$RPM_{\frac{t_{n+1}+t_n}{2}} = \frac{60}{t_{n+1} - t_n} \quad (7-18)$$

$$RPM_t = RPM_n + (t - t_n) \frac{RPM_{n+1} - RPM_n}{t_{n+1} - t_n} \quad (7-19)$$

All three force channels were zeroed individually using an average taken from  $t_0 - 0.5$  to  $t_0 - 0.2$  to compensate for constant offset error due to factors such as tool weight, etc.

Since the data were recorded at a very high sample rate, data points were then binned into discrete time step bands to improve the processing and display time. Each band was represented by its average for all columns including RPM.

### 7.4.2 Time Offset Approximation and Refinement

An approximate time offset was used for each data set to synchronise the calculated width of cut with the cutting force data. The offset was judged by eye and later refined by the processing algorithm using a convergence-based technique. The technique worked by introducing ever decreasing time offsets to the initial time approximation and repeating the analysis in order to discover the offset that gave the smallest standard deviation for an average taken from the  $K$  curve between two predefined representative margins,  $t_{s1}$  and  $t_{s2}$ . The refinement algorithm starts with an initial scan width of  $t \pm 0.1$ .

Points  $t_{s1}$  and  $t_{s2}$  were selected manually by looking at the plotted data and judging a range which was safely inside the period of sustained, full-depth cutting. The criteria for selection was to make the range as wide as possible (to increase the signal to noise ratio), but to also ensure that the range did not cover any period before the cutter was fully engaged, nor any point after the cutter was moved away.

### 7.4.3 Specific Force Calculation

Figure 7-12 shows an example construction of the steps up to this point for experiment 12. In that experiment the target spindle speed was 1667 RPM with a feed rate of  $0.08 \text{ mm rev}^{-1}$ . The graph shows the  $t_0$  point as discovered by the refinement algorithm to be  $t_0 = 1.375$ . In this case the initial cutting start guess was  $t_0 = 1.4$ , meaning the refinement algorithm would theoretically have found this true cutting

start time, so long as it was within the light blue shaded region. The boundaries  $t_{s1}$  and  $t_{s2}$  indicate the sample width over which the specific force curve is averaged.

The blue line shows the measured rake force in Newtons (primary axis), the red line shows the theoretical width of cut as calculated using equation 7-4 (secondary axis) and the green line shows the  $K_r$  values according to equation 7-1 (primary axis).

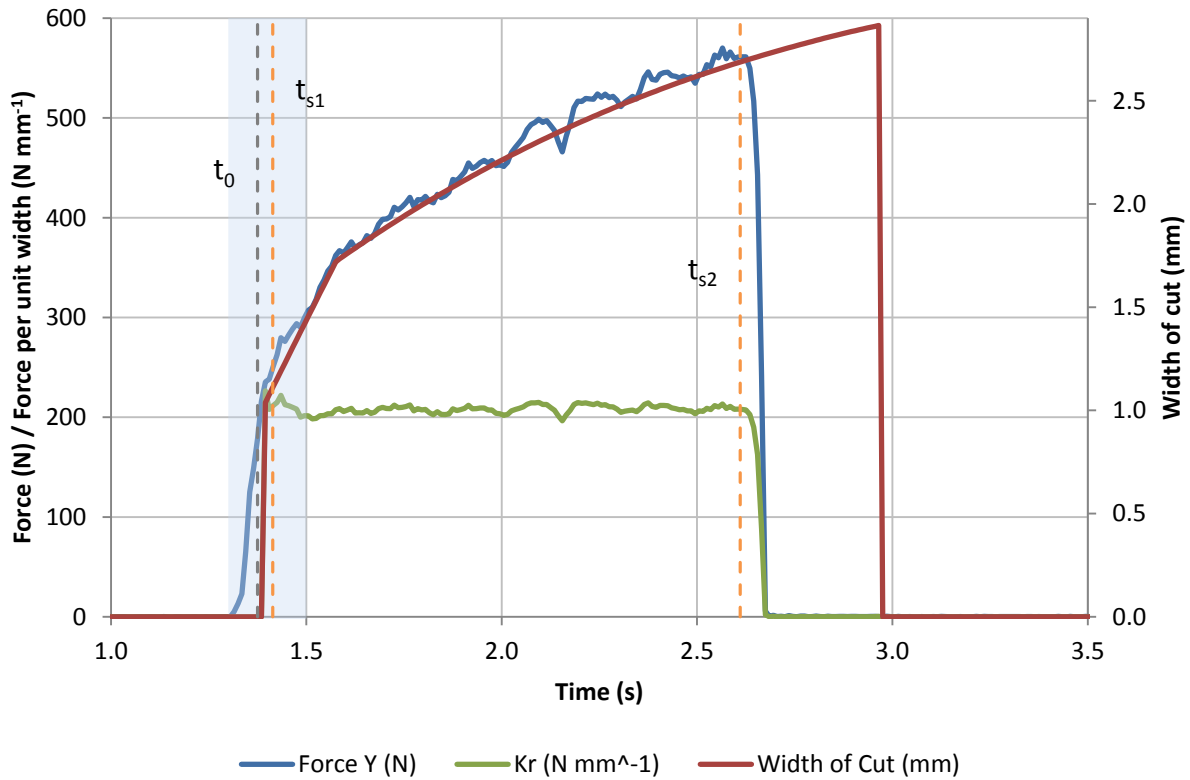


Figure 7-12 – Graph to show  $K_r$  function construction and quality

As the figure shows, the  $K_r$  curve between the selection boundaries approximates a flat line. This shows good agreement with the theoretical model proposed in this chapter which asserted that cutting force varies linearly with width of cut. The average of the  $K_r$  curve between these two boundaries was used as the final specific force for this dataset. This analysis was repeated again for the feed data which also shared this pattern.

During the initial force ramp, there was some disagreement between the force and width of cut profiles. This was because as the cutter began the cut, there was a brief period when the depth of cut was also ramping up to its otherwise constant level.

## 7.5 Results and Discussion

This section discusses the quality of results and presents the cutting force data and thermal data.

A full data set was recovered according to the planned experimental specification. No repeat experiments were required and no visible cutting insert damage was observed.

### 7.5.1 Radial Force Analysis

As explained in the introduction to this chapter, this experiment was designed with the assumption that there would be minimal radial force on the cutter. Any excessive radial force could suggest poor alignment of the cutter relative to the workpiece. Figure 7-13 shows the measured radial force as a function of feed rate for all spindle speeds in both MQL and dry sets overlaid. The graph also shows the largest standard deviation calculated for each data point.

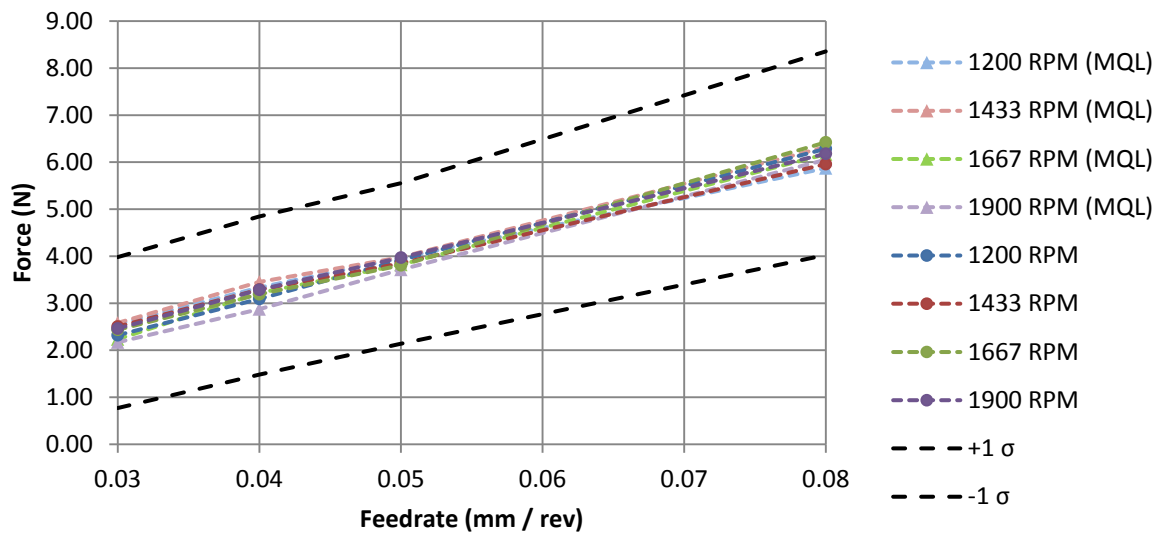


Figure 7-13 – Radial force as a function of feed rate

As the graph shows, the radial force magnitude is extremely low (in the range 2 N to 7 N) compared to the typical cutting force readings (around 350 N). Neither spindle speed, nor lubrication regime affected the radial force, however there was a slight increasing trend with feed rate. As presented in the next section, feed rate also increased the other force components, so this is to be expected of the radial forces since the alignment error is likely to be proportional to the other force components. Overall, the error is extremely low, suggesting good alignment of the tool and workpiece. Furthermore, the standard deviation is also very narrow, further suggesting that the stiffness of the experimental setup is adequate.

### 7.5.2 Specific Force Data

Table 7-3 shows a summary of cutting force coefficients as computed by the processing algorithm for all experiments. For each experiment the quoted RPM value was computed based on an average over the cut from the tachometer data. For both rake and feed force, a standard deviation is given as a percentage of each respective  $K$  value.

	Exp. ID	Real RPM	Feed rate (mm / rev)	$K_r (N m^{-1})$	$K_f (N m^{-1})$	$K_r$ Standard Deviation	$K_f$ Standard Deviation
Dry	1	1188.11	0.03	1.06E+05	1.13E+05	3.39%	2.69%
	2	1187.40	0.04	1.33E+05	1.30E+05	3.35%	3.04%
	3	1182.05	0.05	1.63E+05	1.54E+05	3.00%	2.92%
	4	1173.85	0.08	2.20E+05	1.83E+05	2.54%	3.58%
	5	1401.68	0.03	1.08E+05	1.21E+05	3.05%	2.25%
	6	1396.39	0.04	1.36E+05	1.41E+05	2.79%	2.29%
	7	1389.27	0.05	1.56E+05	1.53E+05	2.73%	2.60%
	8	1370.36	0.08	2.21E+05	1.90E+05	2.24%	3.33%
	9	1635.33	0.03	1.04E+05	1.20E+05	3.07%	2.18%
	10	1628.42	0.04	1.35E+05	1.45E+05	2.85%	2.39%
	11	1622.74	0.05	1.57E+05	1.57E+05	2.55%	2.26%
	12	1600.50	0.08	2.08E+05	1.81E+05	2.00%	3.11%
	13	1820.87	0.03	1.05E+05	1.25E+05	3.11%	2.30%
	14	1795.17	0.04	1.32E+05	1.43E+05	2.83%	2.31%
	15	1793.06	0.05	1.54E+05	1.57E+05	2.78%	2.36%
	16	1754.07	0.08	2.05E+05	1.82E+05	2.69%	3.48%
	17	1187.08	0.03	1.09E+05	1.30E+05	3.23%	2.51%
MQL	18	1187.56	0.03	1.07E+05	1.19E+05	3.34%	2.26%
	19	1183.71	0.04	1.35E+05	1.40E+05	2.78%	2.32%
	20	1181.62	0.05	1.56E+05	1.50E+05	2.71%	2.44%
	21	1174.13	0.08	2.10E+05	1.81E+05	2.63%	3.65%
	22	1404.69	0.03	1.07E+05	1.23E+05	3.20%	2.41%
	23	1396.87	0.04	1.35E+05	1.45E+05	3.45%	3.10%
	24	1391.02	0.05	1.56E+05	1.55E+05	2.39%	2.50%
	25	1372.09	0.08	2.13E+05	1.89E+05	2.34%	3.12%
	26	1637.85	0.03	1.06E+05	1.26E+05	3.59%	2.56%
	27	1627.61	0.04	1.34E+05	1.43E+05	3.08%	2.51%
	28	1624.06	0.05	1.58E+05	1.64E+05	2.89%	2.88%
	29	1603.97	0.08	2.06E+05	1.87E+05	2.84%	3.95%
	30	1827.06	0.03	1.06E+05	1.31E+05	3.16%	2.25%
	31	1814.99	0.04	1.31E+05	1.46E+05	3.18%	2.54%
	32	1796.71	0.05	1.52E+05	1.60E+05	2.65%	2.45%
	33	1750.60	0.08	2.04E+05	1.90E+05	2.40%	3.56%
	34	1187.07	0.03	1.04E+05	1.32E+05	2.91%	2.65%

Table 7-3 – Summary of results

For all experiments, the curve fitting standard deviation was less than 4% suggesting good agreement with the theoretical model. A larger deviation would suggest that cutting force does not share a linear

relationship with width of cut and/or other unidentified factors have a greater influence on the cutting forces.

Each experiment was performed in the order as listed in the table. Experiments 1 to 17 are for the dry set and experiments 18 to 34 are for the MQL set. Experiments 17 and 34 are the check experiments for 1 and 18 respectively.

In each pair of check experiments, the RPM values were well matched to draw fair comparisons between the specific force values. For rake force, both pairs of check experiments showed good agreement, to within 2.0% and 2.2% for dry and MQL respectively.

However, for the feed specific force values, the check experiments suggested an increase in force of 12.1% and 14.7% for dry and MQL respectively. Although the check experiments suggested a wide margin of error, they are consistent with one another which suggests that this increase in force was primarily due to degradation of the cutting edge over the course of each set. If this was the case, the data suggests that MQL slows this degradation slightly.

For all relationships, it was found that cutting force per unit width varies only with depth of cut and that rake velocity produced no appreciable variation for the ranges tested. One possible explanation is due to minimal plastic deformation of the valve seat due the high hardness of the sintered material. In this circumstance the cutter loads the exposed cut face until the force exceeds the yield limit of the material and the chip shears from the bulk of the material.

Figures 7-14 to 7-17 show the relationships for dry and MQL, rake and feed specific force values respectively. In each figure, the red crosses represent the check experiments. The solid horizontal line serves as a datum for the dotted lines which represent the  $\pm 1 \sigma_{max}$  limits for each set of data points.

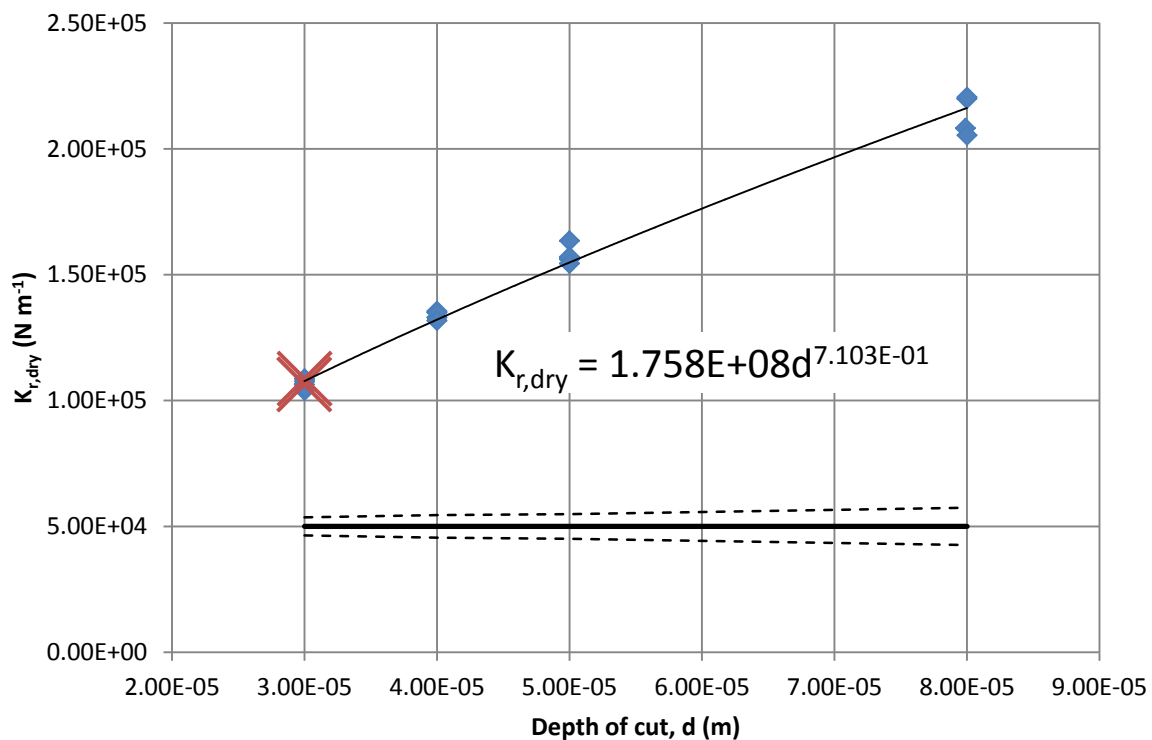


Figure 7-14 –  $K_{r,dry}$  - Dry set, rake force per unit width vs. depth of cut

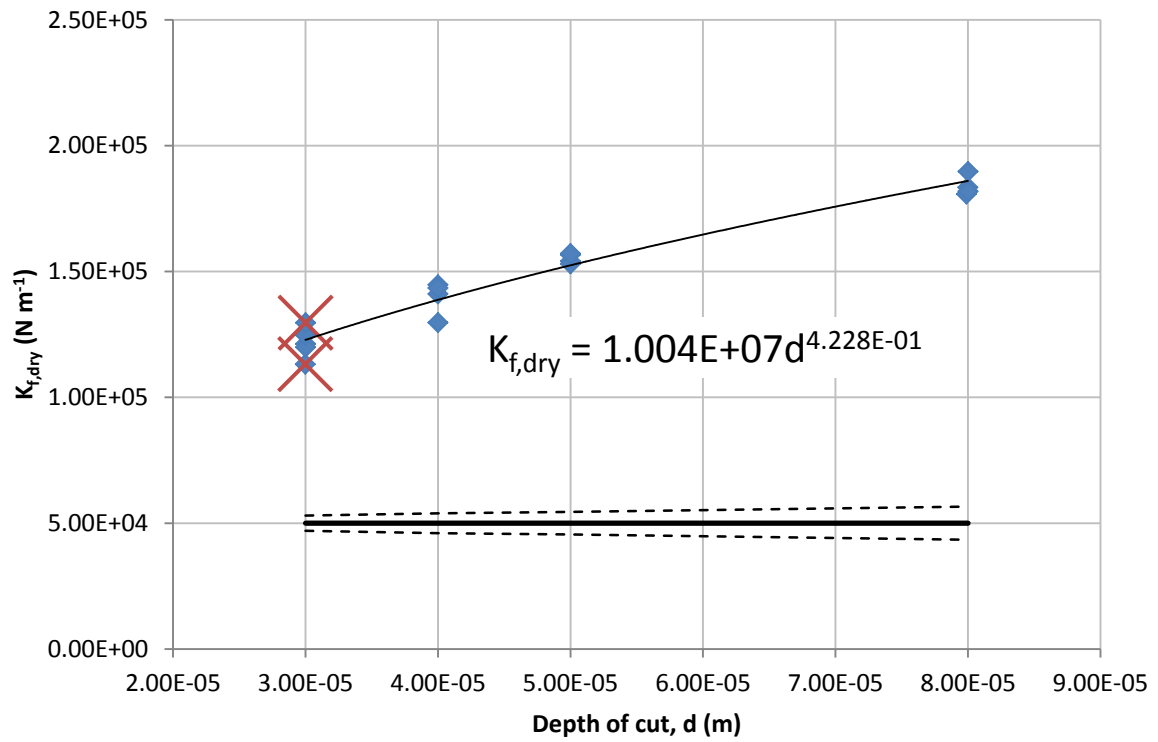


Figure 7-15 –  $K_{f,dry}$  - Dry set, feed force per unit width vs. depth of cut

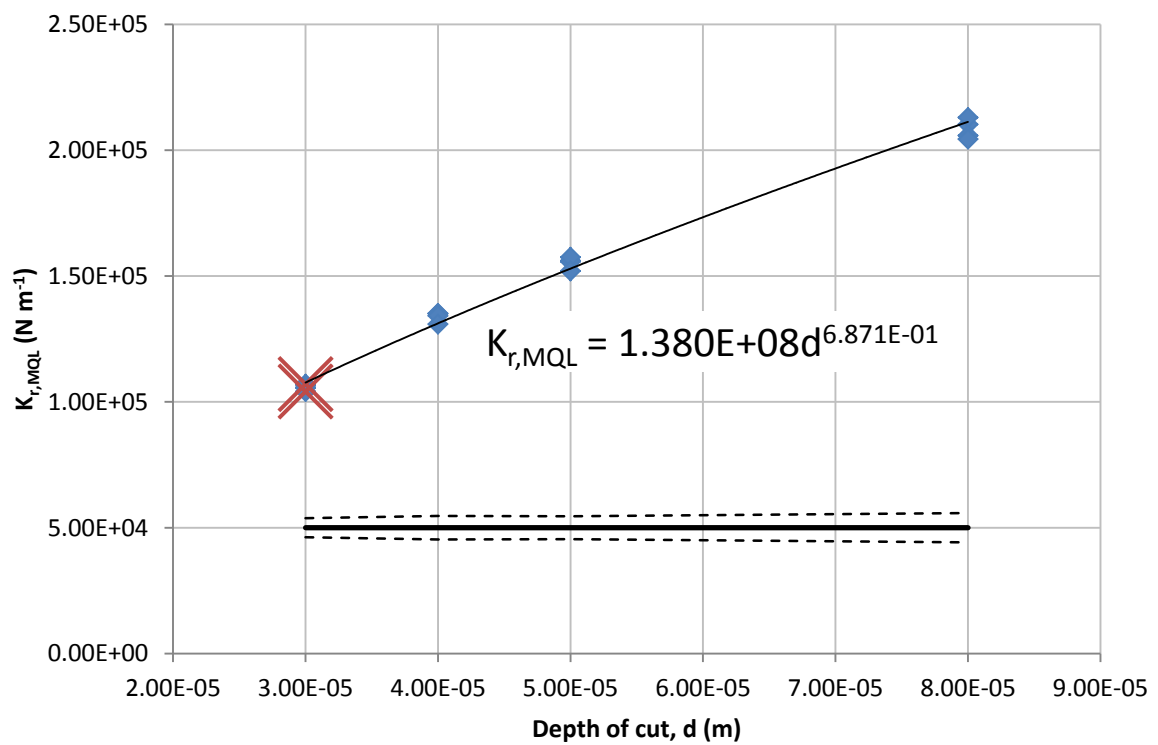


Figure 7-16 –  $K_{r,MQL}$  - MQL set, rake force per unit width vs. depth of cut

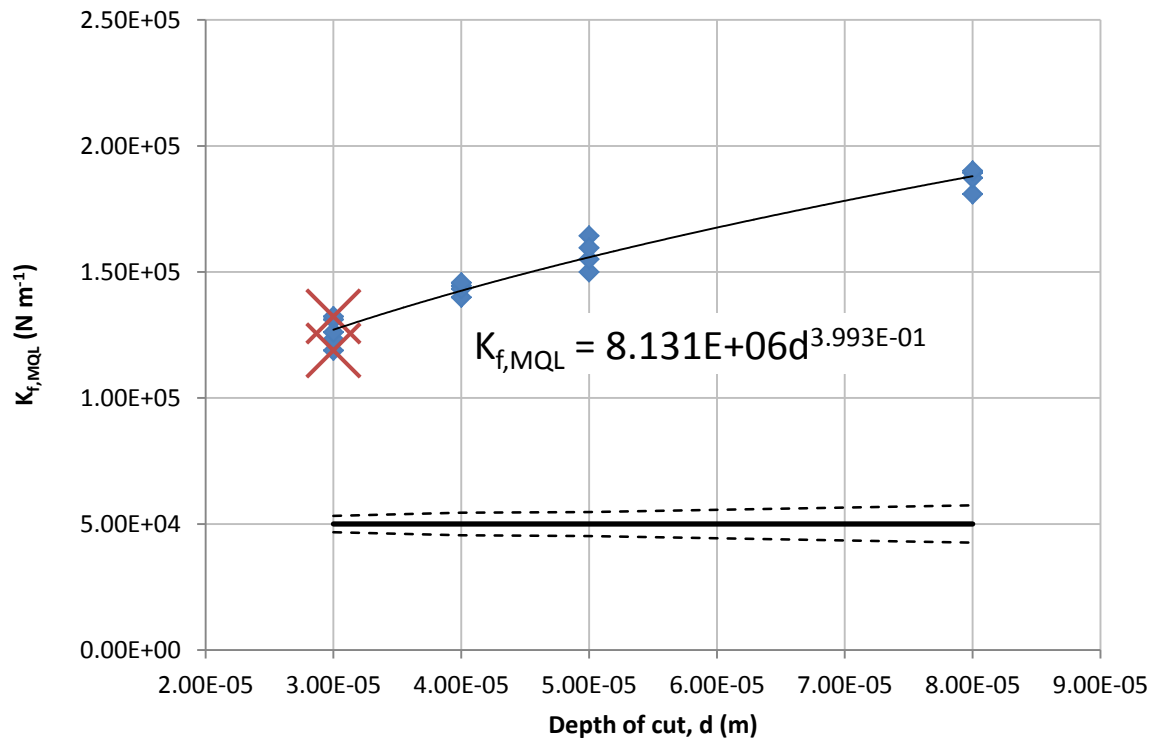


Figure 7-17 –  $K_{f, MQL}$  - MQL set, feed force per unit width vs. depth of cut

The two rake check experiments on each the dry and MQL sets were in close agreement with one another, suggesting good repeatability of the experiment. However, as the feed data shows, the check experiments consistently disagreed by approximately the same amount in both the dry and MQL check pairs. For each feed check pair, the second of the pair (the very last cut performed by the cutting insert edge) was always lower, which suggests that as the cutting insert ages during cutting, the feed force required to remove material drops regardless of lubrication.

As the error bars show, the distance between the initial and check experiment data points was significantly larger than one standard deviation from the mean which suggests that this anomaly cannot be explained by error alone and is instead due to physical change in performance of the cutting insert.

The specific force for both dry and MQL sets can be calculated as a function of depth of cut using the equations given in table 7-4. This relationship is used by the simulation presented in Chapter Nine to calculate the correct cutter force to apply for any given interaction between the cutter and workpiece.

Lubrication	Axis	Relationship
Dry	Rake	$K_{r, dry} = 1.758E^8 d^{7.103E^{-1}}$
	Feed	$K_{f, dry} = 1.004E^7 d^{4.228E^{-1}}$
MQL	Rake	$K_{r, MQL} = 1.380E^8 d^{6.871E^{-1}}$
	Feed	$K_{f, MQL} = 8.131E^6 d^{3.993E^{-1}}$

Table 7-4 – Specific force relationships for dry and MQL conditions

The model developed by Bölling, Kuhne and Abele, 2017, based on identical AR20 workpiece material and similar 65%wt. pcBN cutting inserts was used to generate equivalent specific force data as a function of depth of cut for the range of cutting parameters used in this work. These data are compared to the dry rake specific force relationship (given in table 7-4) as shown in figure 7-18. As the data show, the two models largely agree, especially at lower feed rates, but begin to diverge by approximately 10% at higher feed rates. This divergence could be explained by a difference in pcBN coating or cutting edge radius (unspecified in the referenced work).

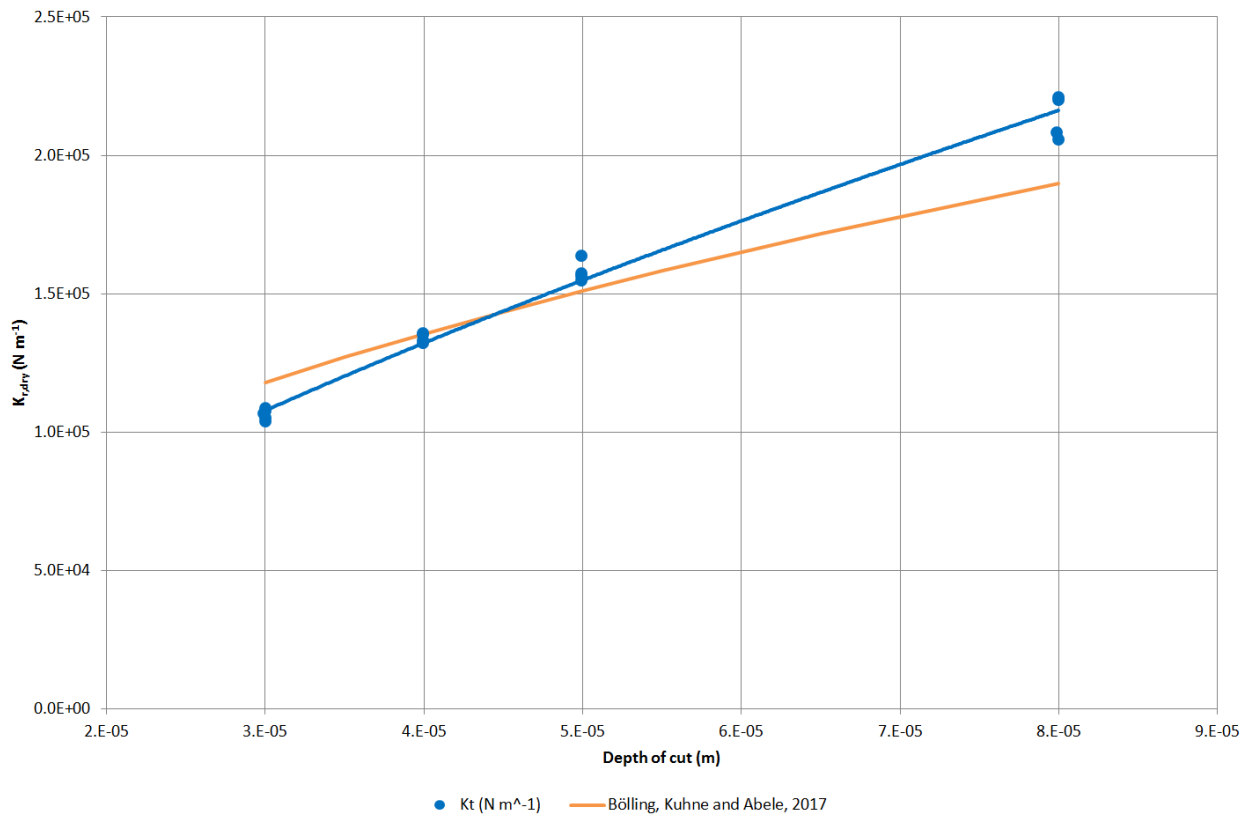


Figure 7-18 – Bölling, Kuhne and Abele, 2017, model vs. dry rake specific force



Figures 7-19 and 7-20 show the MQL and dry sets overlaid for both rake and feed specific forces respectively. As the graphs show, the presence of MQL confers no appreciable benefit in reducing cutting force.

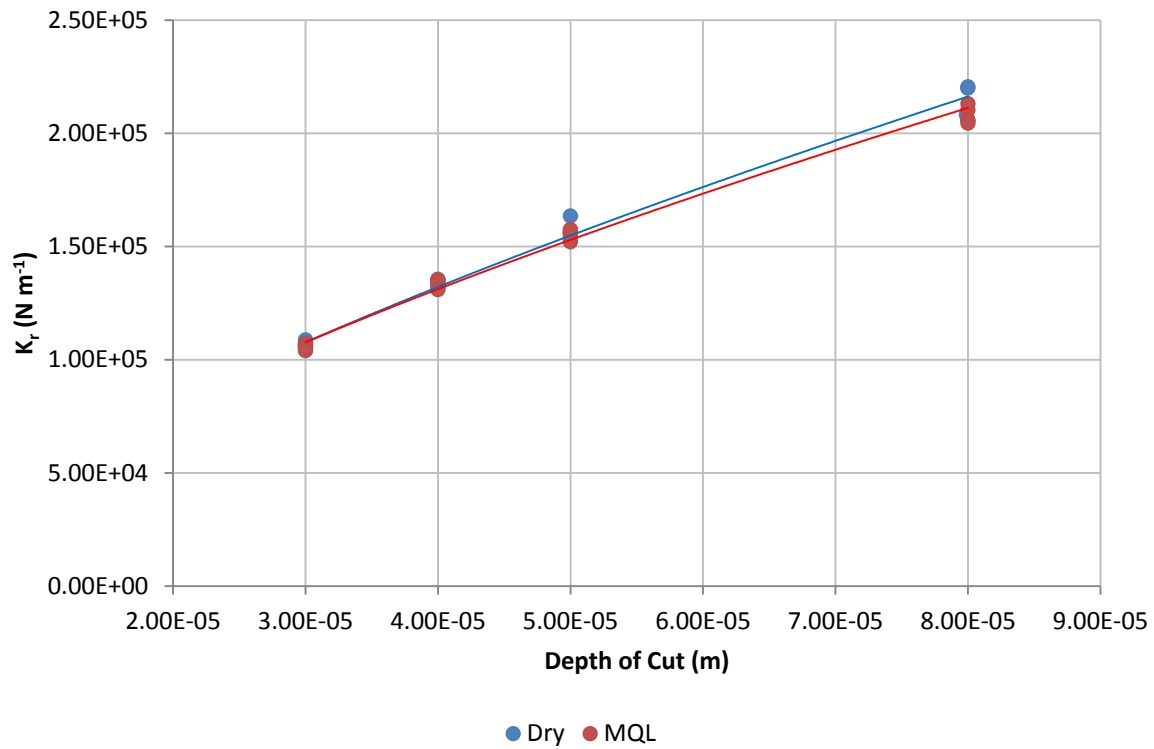


Figure 7-19 – MQL & Dry, rake force per unit width vs. depth of cut

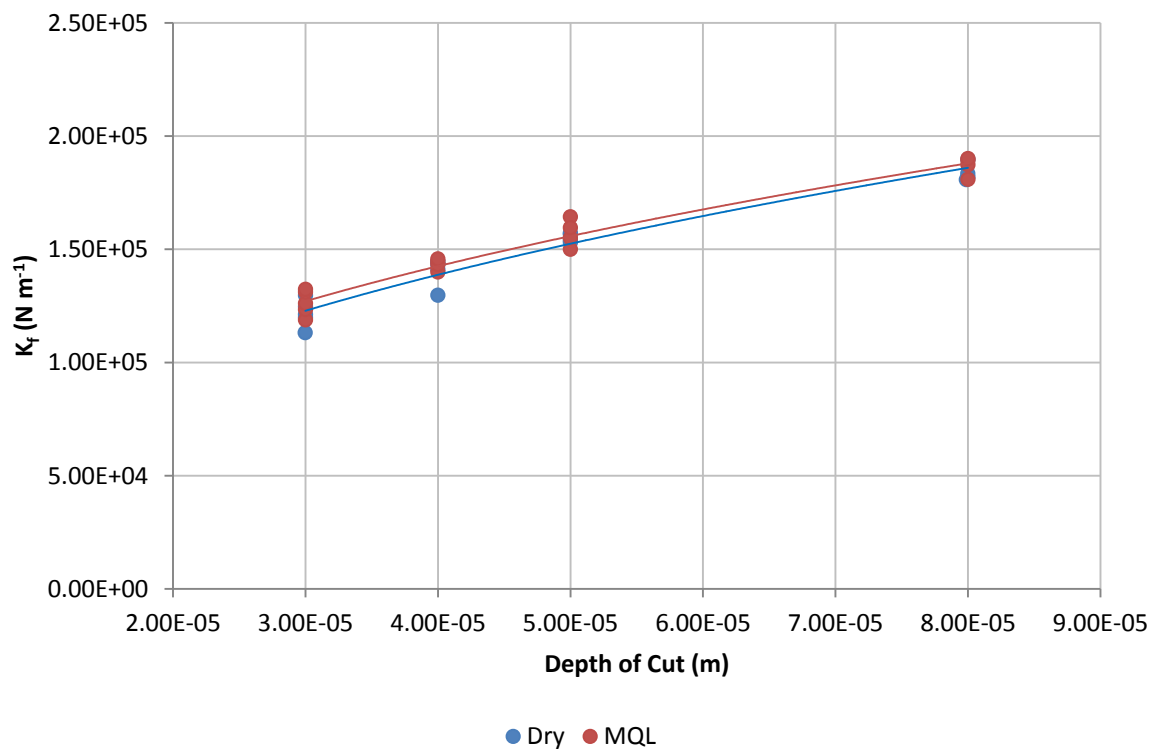


Figure 7-20 – MQL and Dry, feed force per unit width vs. depth of cut

### 7.5.3 Thermal Analysis

A Micro-Epsilon TIM400 thermal imaging camera was used to capture thermal images of the exposed tool tip face immediately after cutting. Figure 7-21 shows the location of the camera relative to the cutting insert.

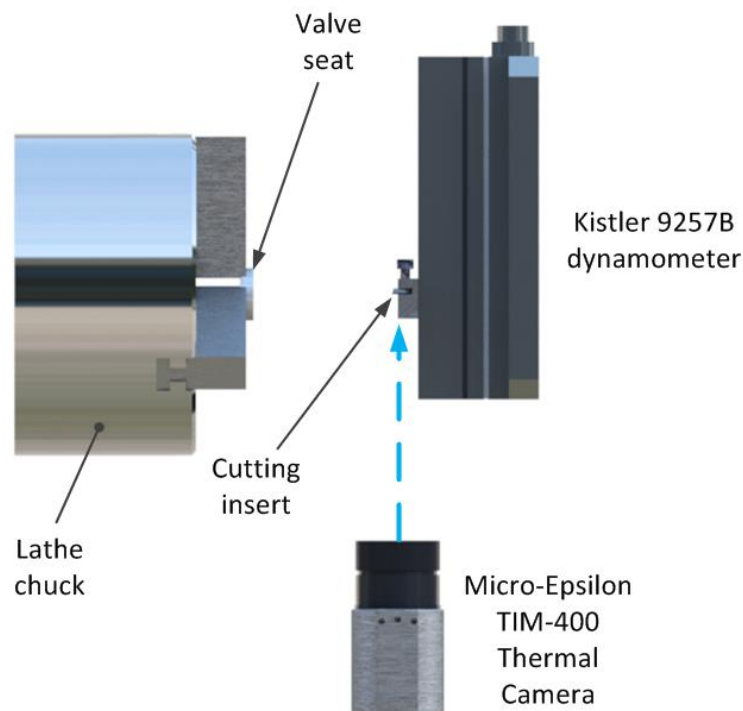


Figure 7-21 – Thermal camera position

Figure 7-22 shows the view from the thermal camera (left) and the corresponding schematic view (right). The camera was directly below the cutting insert looking up at the process zone. From this perspective the camera could capture images of the cutting insert edge as well as the bulk temperature of the valve seat. Temperatures were sampled from area 2 as indicated in the figure.

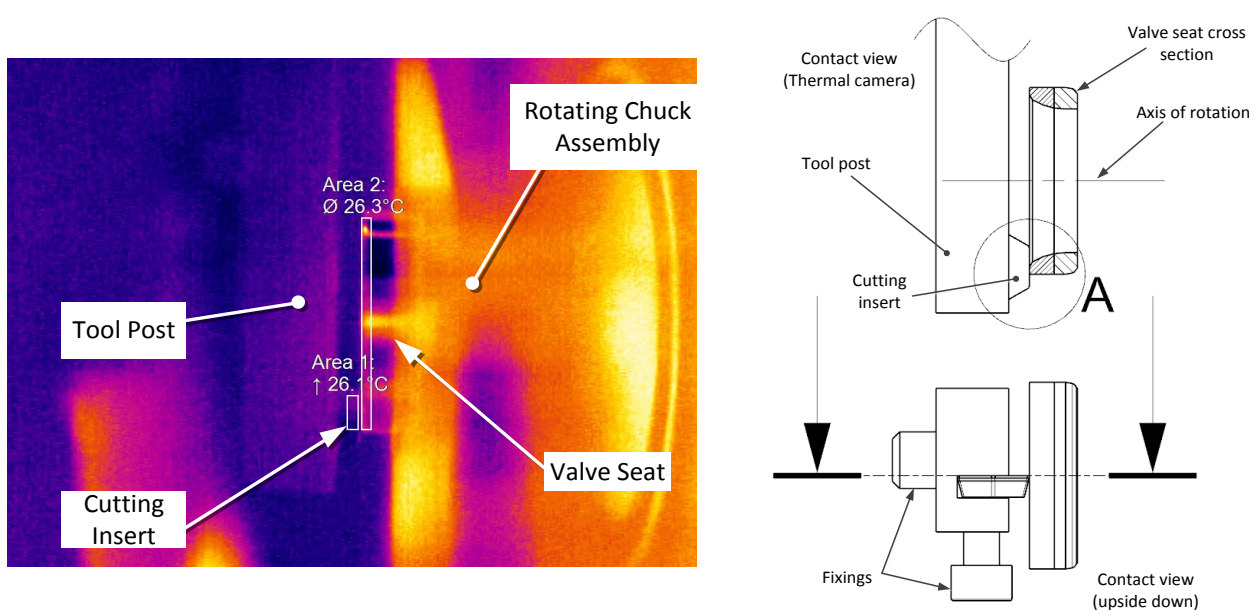


Figure 7-22 – Thermal camera perspective

All TIM-400 cameras are factory calibrated using a LandCal P550P calibration source with emissivity of  $>0.995$ , and uncertainty of  $<\pm 2$  K. (AMETEK, 2009). During calibration, readings are compared to a reference device calibrated by Physikalisch-Technische Bundesanstalt (PTB) and calibration parameters are selected to force agreement to within 5%. During operation, the camera performs an offset correction every few minutes. A physical flag is placed in front of the image sensor and each image element is adjusted to achieve uniformity (Micro-Epsilon, 2012).

Figure 7-23 shows the maximum recorded temperatures of the valve seat bulk for all feed and speed settings for the dry set. Likewise, figure 7-24 shows the same data for the MQL set.

From the dry set it was clear that feed rate played an important role in heat generation, and that this role was not linear. Cuts performed at a feed rate of  $0.04 \text{ mm rev}^{-1}$  consistently produced the largest amount of heat regardless of the RPM.

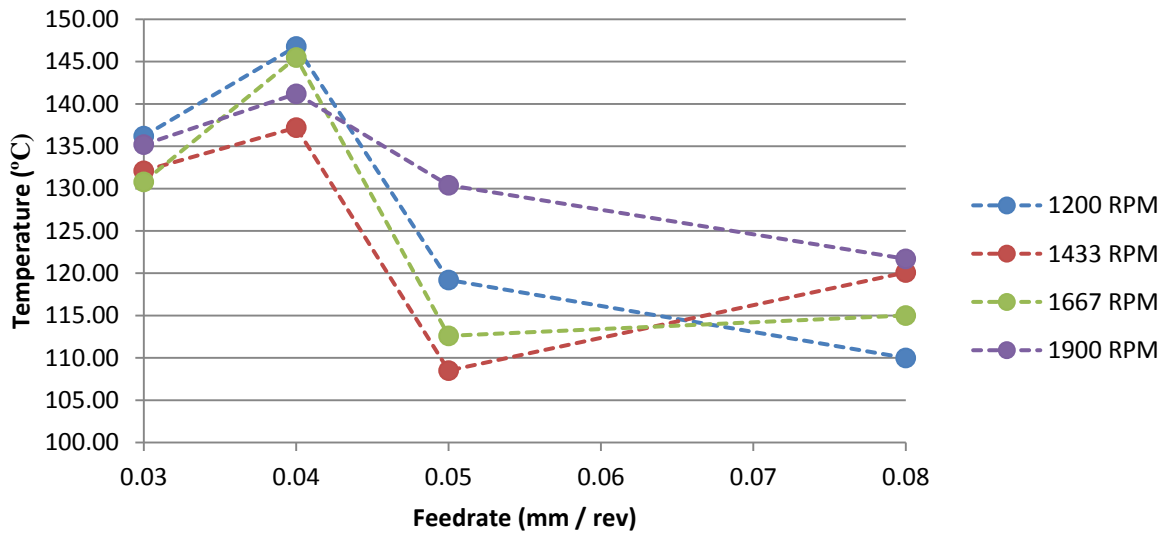


Figure 7-23 – Dry valve seat bulk maximum temperatures

For the MQL set, the same correlation between feed rate and temperature exists, albeit slightly attenuated. In all cases, the presence of MQL lubricant reduced the cutting temperature, up to 20% but typically around 9%.

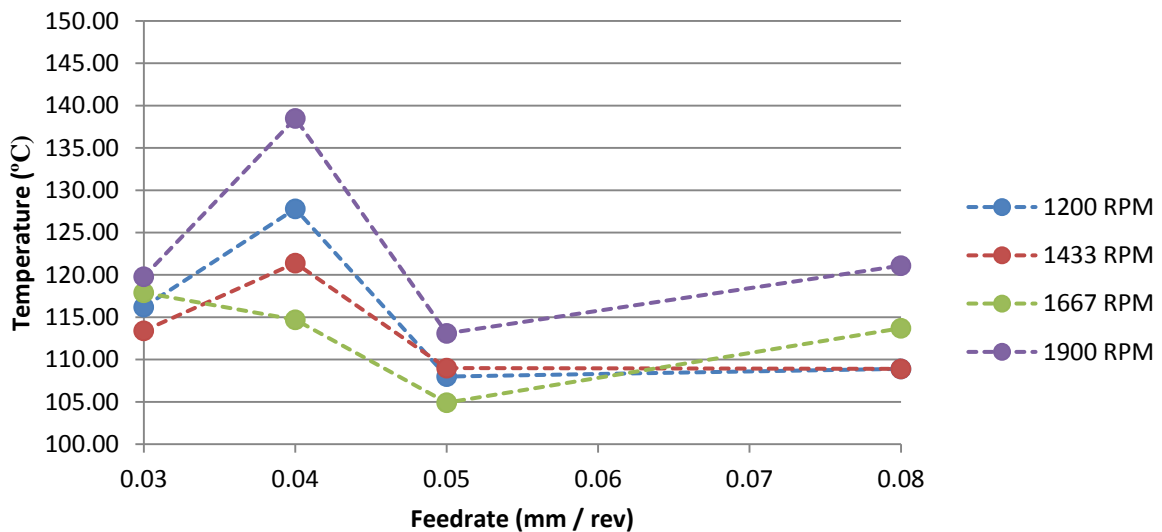


Figure 7-24 – MQL valve seat bulk maximum temperatures

Figure 7-25 shows the maximum MQL valve seat bulk temperature as a fraction of the maximum dry valve seat bulk temperature for all feed rates and spindle speeds. As the figure shows, the MQL set developed consistently lower temperatures during cutting than its dry counterpart for all cases except for a feed rate of 0.05 mm rev<sup>-1</sup> at 1433 RPM. The effect is more pronounced at lower feed rates, with a feed rate of 0.08 mm rev<sup>-1</sup> showing negligible difference in all but one case.

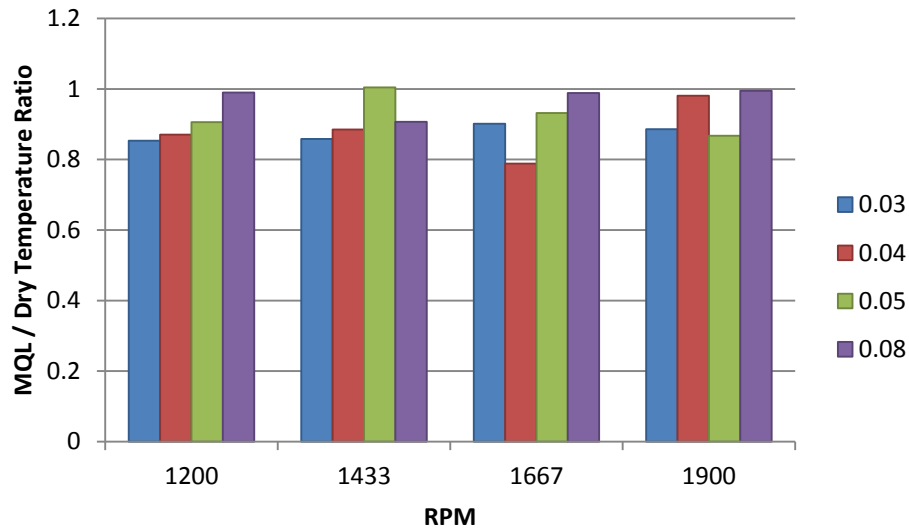


Figure 7-25 – MQL valve seat bulk temperature as a fraction of dry temperature

As discussed earlier in this chapter, the experiments were all performed sequentially according to the experiment number. No repeat experiments were performed and the cutting insert was rotated to a new edge for the MQL set. This feature of the experiment gives some insight into life-dependant effects taking place in the cutting insert.

Figure 7-27 shows the cutting edge temperature gradients for all experiments viewed from the perspective shown in figure 7-26. As the perspective diagram shows, only the tip of the hexagonal cutting insert was visible, since the bulk of it was obscured by the tool post. The images were taken from the frame immediately visible after the cut finished. Prior to this point, the view was obscured by chip. Some images were not available due to unfortunately timed internal calibration cycles performed by the camera in which the frame was frozen for a short period of time.

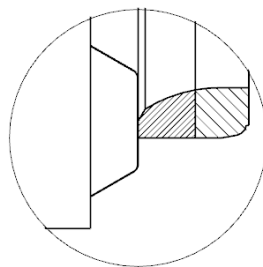


Figure 7-26 – Detail A as indicated in figure 7-22

As the figure shows, generally the MQL set thermal gradients were smoother and were less likely to show thermal inclusions into the cutting insert compared to their dry counterparts.

Dry and MQL cuts, 4 and 21, both show a thermal gradient along the cutting edge with most thermal energy centred around the part of the insert which cuts at the largest radius. This is understandable since this part of the insert has the most demanding duty cycle.

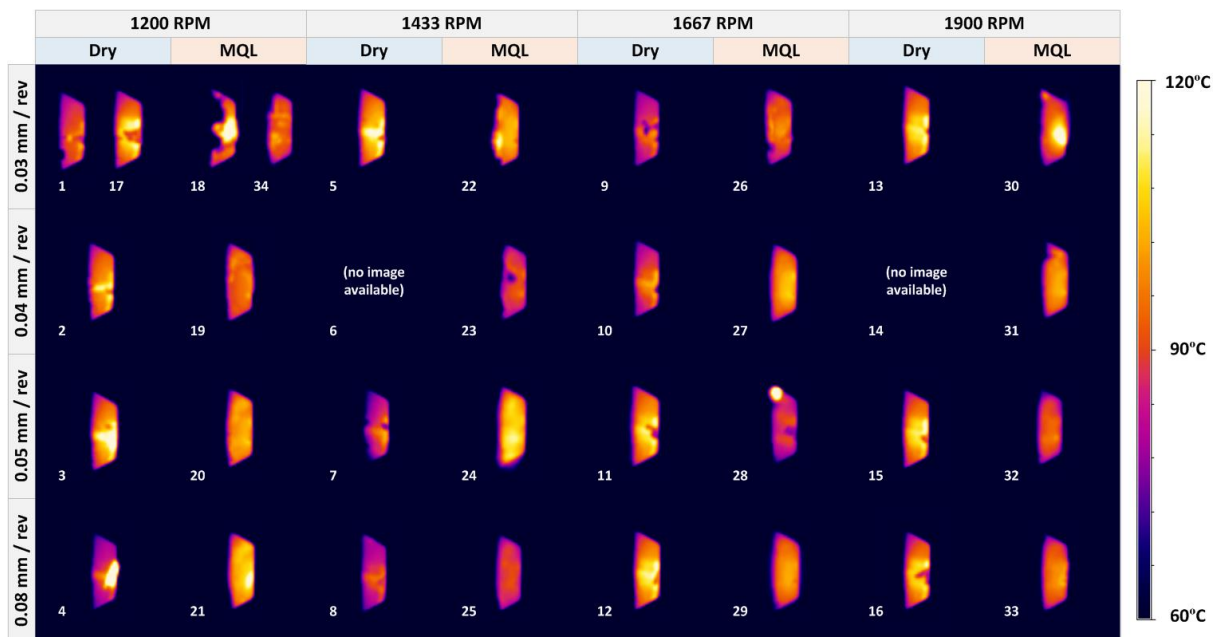


Figure 7-27 – Cutting insert edge temperature gradients

Examination of the dry set, 0.08 mm rev<sup>-1</sup> feed group (experiments, 4, 8, 12 and 16), revealed that as the experiment progressed, the phenomenon reversed and in fact the edge exposed to the largest duty cycle was the coolest immediately after the cut. Not only was it the coolest, but it was also below the bulk temperature of the surrounding material. This suggests that a chemical change may have been occurring in the insert as the number of heat cycles increased, which reduced the thermal conductivity of the affected areas rendering them less able to store heat from the cut.

The effect was more pronounced when compared to the dry check experiments 1 and 17 which were performed at the same spindle speed and feed rate. In those images, which represent the very start and end of the cutting insert edge life, it was apparent that significant change had occurred in the insert which changed the conductivity of the substrate.

#### 7.5.4 Surface Analysis

The surface finish of valve seats shares a complicated relationship with engine performance. High surface roughness can detrimentally affect leak resistance (LoRusso *et al.*, 1984), limit the rate of heat transfer from the valve to the cylinder head (Stotter, 1965) inhibit fuel film atomisation during lift which can lead to higher hydrocarbon emissions (Wang, Wilkinson and Drallmeier, 2004) and accelerate valve recession due to increased rates of adhesion wear (Wang *et al.*, 1995; Lakshminarayanan, 2001). Since valve seats are commonly finished by cutting as opposed to grinding (Lin and Chen, 1995), it is important to understand the relationship surface roughness shares with cutting parameters.

An Alicona InfiniteFocus surface finish measurement device was used to capture the topology of the machined valve seat faces. The Alicona was set to 20 X optical magnification providing a Ra sensitivity of 150 nm. Figure 7-28 shows the scan area (to scale) captured by the Alicona to determine the surface roughness.

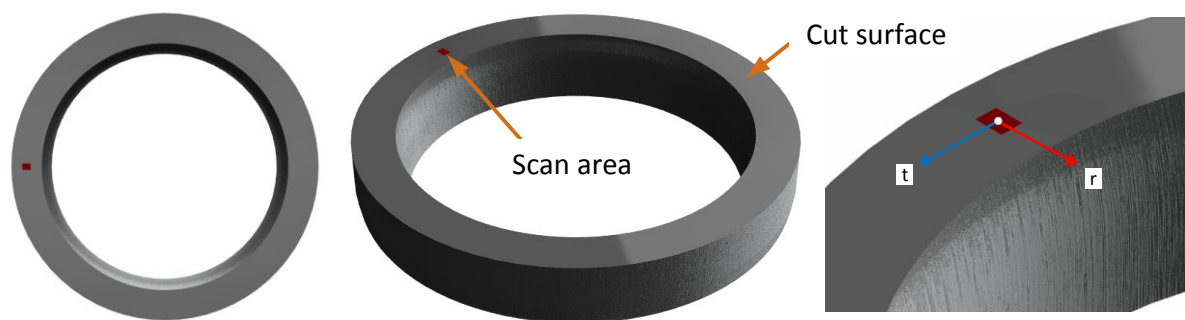


Figure 7-28 – Location of scan area for roughness measurements

Figure 7-29 shows the Ra roughness values for the valve seats from all experiments measured in accordance with the ISO 4287 specification.

The upper and lower rows show the measured roughness values for a set of points taken along a tangential and radial path respectively. As the figures show, the roughness was generally anisotropic especially for the dry cuts. The dominant factor influencing roughness was the presence of MQL oil, which decreased the surface roughness. For radially measured roughness, a feed rate of  $0.05 \text{ mm rev}^{-1}$  produced the most consistent roughness across all speeds tested. For the tangentially measured roughness however, a feed rate of  $0.04 \text{ mm rev}^{-1}$  was consistently better than  $0.05 \text{ mm rev}^{-1}$ .

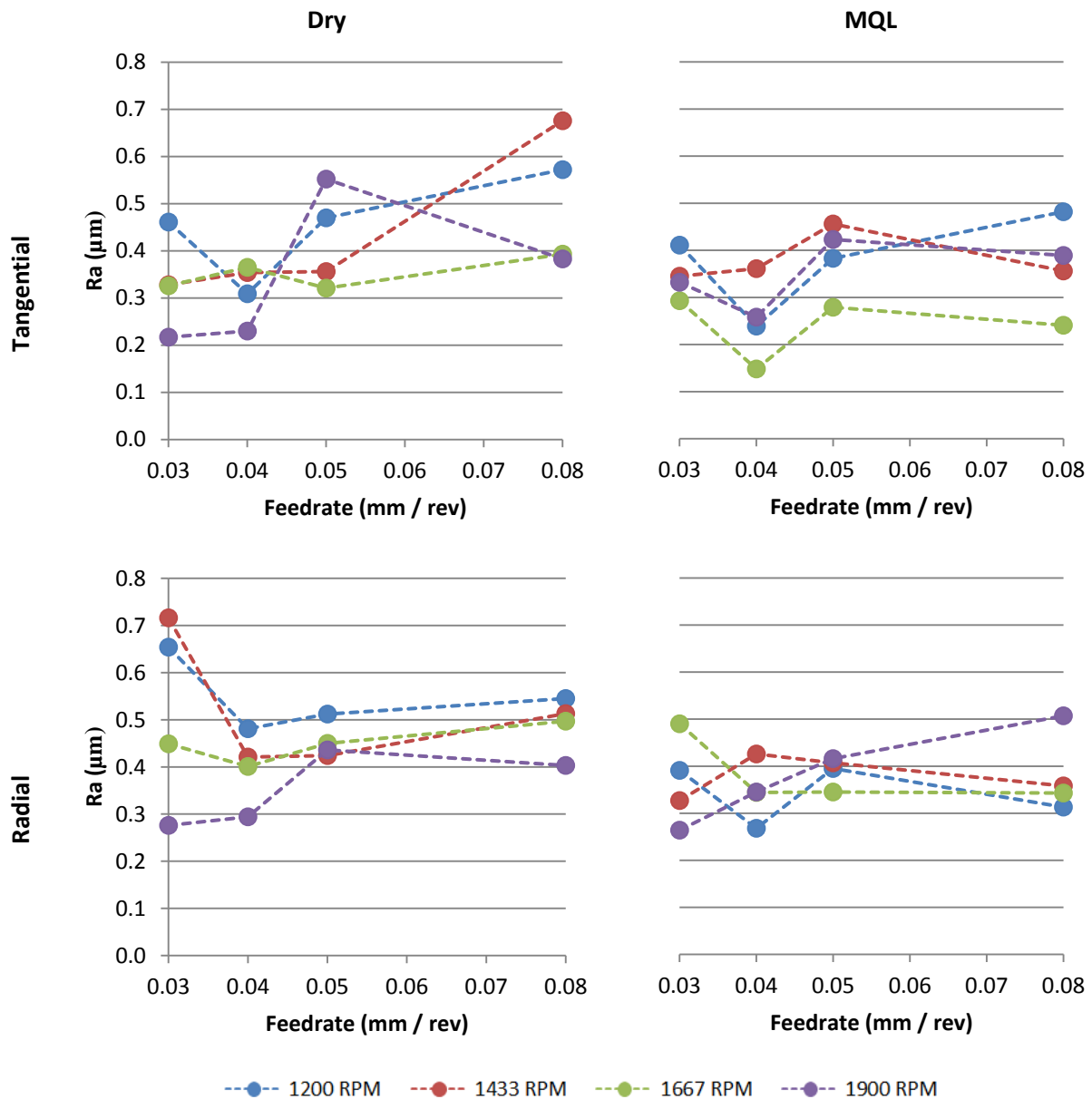


Figure 7-29 – Seat roughness (Ra) values for all experiments

Figure 7-30 shows surface morphology from experiment three performed at 1200 RPM and  $0.05 \text{ mm rev}^{-1}$ . This morphology was typical of all seats inspected. In this figure, the tangential direction shows the direction of the cutter motion relative to the seat. As the figure shows there were clear striations in the cutting direction which gave rise to the anisotropic surface roughness. Pitting was also visible on many of the seats inspected, however from prior experiments these pits are known to be exposed pores which were formed during the sintering process. The orange details on the surface are copper present in the seat from

the copper infiltration process. These orange artefacts were smeared in the direction of cut suggesting that the relatively soft copper deposits were smeared during cutting rather than shearing neatly with the remaining bulk of the valve seat.

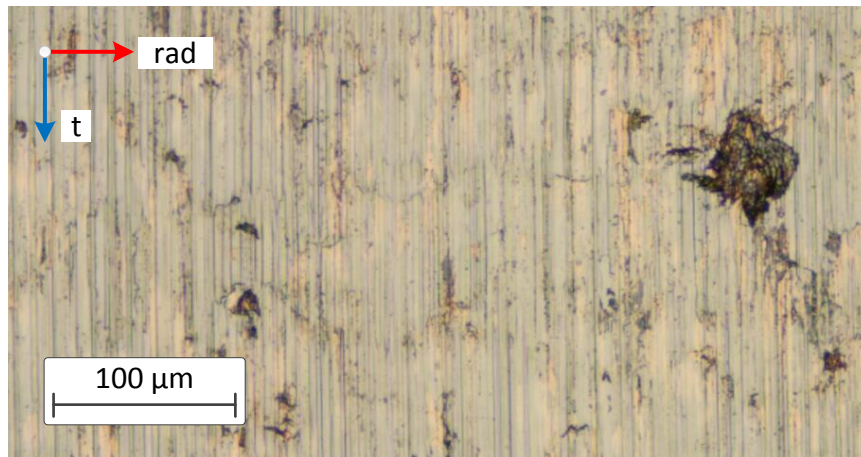


Figure 7-30 – Surface morphology

The Alicona was also used to check the final state of both the dry and MQL cutting edges using 50 X magnification as shown in figure 7-31. The view perspective for each edge is indicated in the figure. As the figure shows, the quality of the edge shows no appreciable degradation or damage. This is expected due to the extremely light life cycle of the cutting insert.

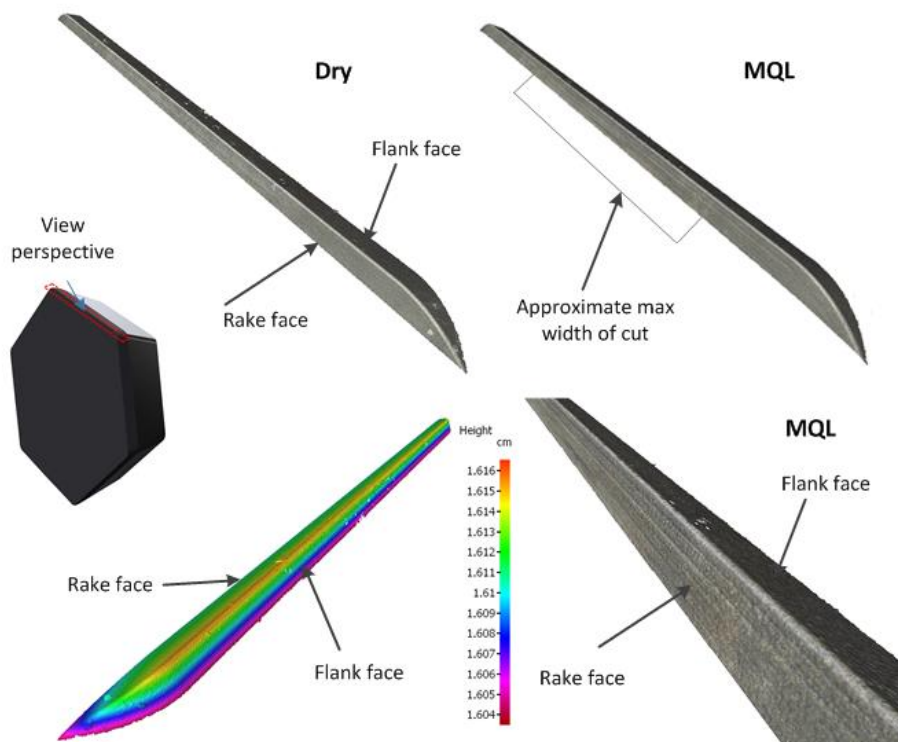


Figure 7-31 – MQL and dry cutter edge quality



## 7.6 Chapter Summary

The experimental data presented in this chapter shows that the experiment to determine cutting forces as a function cutting parameters was a success. A full data set was acquired which displayed a clear relationship between depth of cut and cutting force. The assumption that cutting force varies linearly with width of cut has also been proven by the data presented.

Despite initial expectations, it was found that the key factor in determining the cutting force was depth of cut and that rake velocity has a negligible effect. This outcome is attributed to the high hardness of the sintered valve seat material and the lack of plastic energy absorption.

These data have allowed the formulation of a tool force prediction model that describes the cutting force per unit width as a function of depth of cut for both dry and MQL cutting conditions, as summarised earlier in table 7-4.

This experiment demonstrated that MQL offered no benefit in reducing either rake or feed cutting forces when compared to dry conditions. MQL lubricant did however improve the thermal gradient across the cutting edge and generally reduced the measured temperature in the valve seat bulk up to 20% in some cases. However, it is important to note that MQL oil was applied only once for each cut whereas CNC machines are capable of continuously misting the newly exposed workpiece face. This was a limitation of this experiment, however since the MQL oil was not used for chip ejection and because both the workpiece and tool were saturated throughout the MQL experiments, it was unlikely to have dramatically affected the cutting force data. This assumption is supported by the data which show MQL does affect temperature, indicating that MQL oil was present throughout cutting.

The thermal data showed that cuts with a feed rate of  $0.04 \text{ mm rev}^{-1}$  generated significantly more heat than any other feed rate tested, (including faster and slower feed rates). There was strong evidence in the thermal data to suggest that cutting insert underwent a chemical change which affected thermal conductivity.

A surface inspection of the cut valve seats revealed that surface finish shares a complex relationship with spindle speed and feed rate. However it was found that MQL oil generally improved the surface finish.

# Chapter Eight – Development of Theoretical Finite Element Model

---

In the research for this thesis, it was suggested that valve seat cutting systems using polycrystalline cubic boron nitride (pcBN) are particularly vulnerable to tool chipping due to the high radial forces that develop during imbalanced multi-angle machining, in conjunction with the poor fracture toughness of pcBN.

The ability to simulate this imbalance in order to test alternative designs would be beneficial to tooling manufacturers. Many numerical models in literature over several decades have gone to great lengths to simulate metal cutting processes. Traditionally these simulations attempt to directly model a wide range of physical phenomena including plasticity, chemical changes in the material, shear flow, friction and chip geometric formation and flow. Whilst these models often achieve a noteworthy degree of accuracy, the results they produce are not universal and are not readily transferable to new tool or workpiece combinations. Furthermore they are extremely sensitive to scale, with many only achieving very small scale models and none presenting models that make multiple passes over previously worked material.

Alternatively, a tool force prediction model, developed in the previous chapter can be used within a simulation to determine tool forces based on the geometry of intersection between the tool and workpiece. This simplification drastically decreases the computation demand vs. more precise models that model chip flow and other complex cutting phenomena.

This chapter therefore presents the design and development of a parametric numerical model aimed at simulating multi-angle valve seat cutting operations in 3D based on the tool force prediction model developed in the previous chapter.

## 8.1 Justification

During the machining operation, three individual inserts spaced 120° apart around the axis of rotation cut three different angles of the valve seat. Each insert engages the workpiece at a different time and has a unique inclination and depth of cut and, therefore, varying cutting loads and states of wear. All inserts are orientated such that cutting velocity is not constant across the cutting edge. All of these factors combined result in an arrangement of cutters that each develop significantly different radial loads.

Lacerda and Siqueira, 2012, show that unopposed inserts when cutting valve seats leads to high amplitude vibration. In their research they found that this vibrational energy dissipates into the workpiece causing an unacceptable roundness deviation. However their seat material was notably less hard than that used in the case study for this work at approximately 390 HV versus more than 480 HV in the case study. For the harder material, it is reasonable to assume that more of the vibrational energy would be reflected back into the cutting insert. pcBN is particularly susceptible to cracking and chipping when exposed to sudden high loads due to its poor fracture toughness, (Rocha *et al.*, 2004).

One solution to Ford's problem, might be to redesign the tool holder, so that instead of using three tools (one per angle of the seat) it uses six tools (two per angle) where common angles oppose one another.

However, this solution would not always be practical, since space would need to be found within the relatively small ring diameter for the insert clamps, coolant channels and clamping system for the reamer. In this work, the valve seats used have an outer diameter of 26 mm, and an inner diameter of 20 mm.

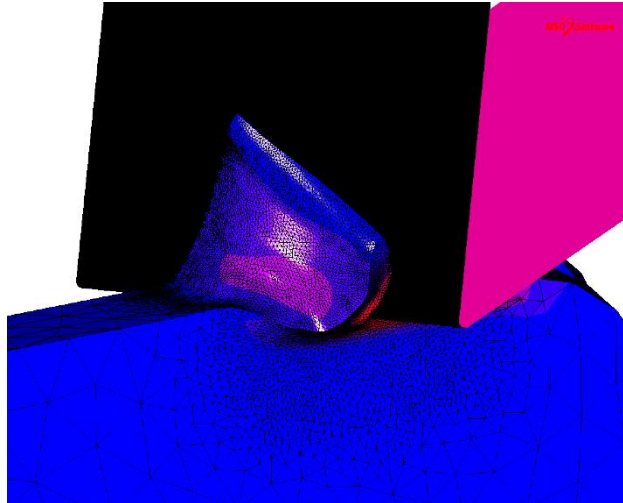
H. Lacerda and I. Siqueira report that for their configuration, imbalanced radial loads of up to 400 N were observed. The starting configuration in this work is likely to suffer from imbalanced radial loads near that magnitude. It would be reasonable to assume that stability can be achieved at some peak imbalance below this level, but not necessarily zero.

This presents the possibility of a compromise, for example, only three inserts are used again, but are set at unequal offsets around the tool (e.g. 330°, 30° & 180°), or as another alternative only the insert that is most responsible for the imbalance is opposed at 180° and the remaining two angles are left unopposed, calling for just four cutting inserts. Many possible configurations exist, some of which may reduce the magnitude of the imbalanced load just enough to drastically increase the predictability of pcBN insert life when used for valve seat cutting.

Further research on this topic may be interested in a number of other properties of this machining system, such as the wear profile along the edge of each cutter, the temperature distribution along the edge, the engagement times and durations, the effect of lubrication and so on.

To simulate multi-angle valve seat machining using conventional finite element methods would take considerable computational effort due to the complexities of simulating machining as presented in Chapter Four. Phenomena such as chip plasticity, shear failure, built-up-edge (BUE), friction, heat dissipation, thermal expansion and many others would need to be first measured experimentally and then simulated to a high degree of accuracy to ensure stability over the entire duration of the cutting simulation.

Figure 8-1 shows an early attempt at modelling this type of detail for this work. Although far from optimised, it is clear that such detailed representation of the geometry is an unaffordable luxury. The simulation progress shown in this model took several days to reach and suffers from a number of degeneracies (see rounding off of right hand side). Ultimately this model failed as Marc's internal mesher was unable to create a mesh for the next increment. Although Marc's tetrahedral mesher is considered to be extremely capable, it is by no means stable enough to trust for a prolonged simulation with very large geometry changes. It would be completely impractical to simulate a full valve seat machining operation using this approach.



*Figure 8-1 – Failed chip flow modelling attempt*

The chip geometry and flow is of little interest to those concerned with balancing the cutting process. Furthermore, such a simulation would require extremely precise material and friction models as geometric error will compound with each increment.

These constraints are not practicable for industry where there is pressure to produce speedy results at low cost, nor do they yield sufficient data to justify the cost.

In summary, it is clear that a numerical model to simulate multi-angle valve seat machining would be very beneficial to tooling designers. Current numerical methods and software packages do not provide a means to create reliable simulations that produce results in cost-effective time frames. Therefore it is necessary to create a new model that uses a wide range of simplification techniques to greatly accelerate the simulation process whilst maintaining accuracy.

## 8.2 Objectives

This thesis will demonstrate that this type of simulation can be drastically accelerated using a number of simplifications and specially written code. The following simulation design objectives are established. The model must:

- be very easily adjusted with parametric inputs to control the workpiece geometry, cutter geometry, cutter count, cutter layout and cutting parameters (such as RPM, feed rate, etc.);
- require the minimum possible experimental effort to determine material and cutting system properties for input into the simulation;
- simulate the cutting process in a number of hours, rather than a number of days or weeks; and
- not be constrained to some fixed coordinate system, but instead should be free to configure in any orientation so that the model can be coupled to a larger simulation, for example a flexible cylinder head and fixture model.

The required outputs must be the;

- individual cutter loads and moments;
- overall spindle load and moment;
- overall workpiece load;
- cutting face duty cycle map; and
- incremental changes in workpiece geometry due to cutting.

Furthermore, the following further work outputs must be considered where possible:

- roughness of the finished surface; and
- temperature of the surface during cutting.

Section 8.4.1 gives a list of assumptions required in order to simplify the model sufficiently to meet these objectives.

## 8.3 Strategy

Marc does not contain the necessary built-in features to directly meet these objectives. However, Marc does provide an exceptionally powerful interface through its user subroutine subsystem. User subroutines are precompiled functions and subroutines that Marc can be instructed to call at key points during simulation. The subroutines are written in Fortran and are compiled to an executable for high performance execution. This feature of Marc allows key functionality, such as calculating the cut workpiece geometry, to be implemented by writing bespoke Fortran code.

Furthermore, almost every aspect of the Marc interface can be controlled using Python, which is perfectly suited to meeting the parametric objectives of this model. This feature can be used to build geometry, set boundary conditions and input the job and loadcase settings.

With these objectives, and the relevant strengths and weaknesses of Marc in mind, the following strategy is established.

**Chapter Eight (this chapter)** – will present the design and implementation of a series of Python scripts to generate and set the initial process geometry, boundary conditions and other Marc settings within the Marc user interface from a user defined parametric configuration file. The model will contain instructions

for Marc to invoke a series of user subroutines at key points during simulation to calculate outputs, such as cutting forces, reaction forces and changes in geometry.

**Chapter Nine** – will present the design and implementation of a Fortran program, the engine underpinning the model, which contains user subroutines and other code to be called by Marc during simulation in order to process cutter-workpiece intersections, modify newly cut geometry and output data such as cutting forces based on the cutting force prediction model developed in the previous chapter.

For the end user, the basic workflow will follow the procedure as shown in figure 8-2. The user must first gather experimental data in order to create a cutting force prediction model for their particular valve seat and cutter combination, according to the methodology presented in Chapter Seven. The user must then run simulations for each cutter layout configuration by first generating a Marc model using the parametric model building script developed in this chapter, and finally run the simulation in Marc using the Fortran program developed in this next chapter. Once several configurations have been simulated, the user can compare the results to select the superior configuration based on their criteria, for example, lowest peak radial force or lowest sustained radial force.

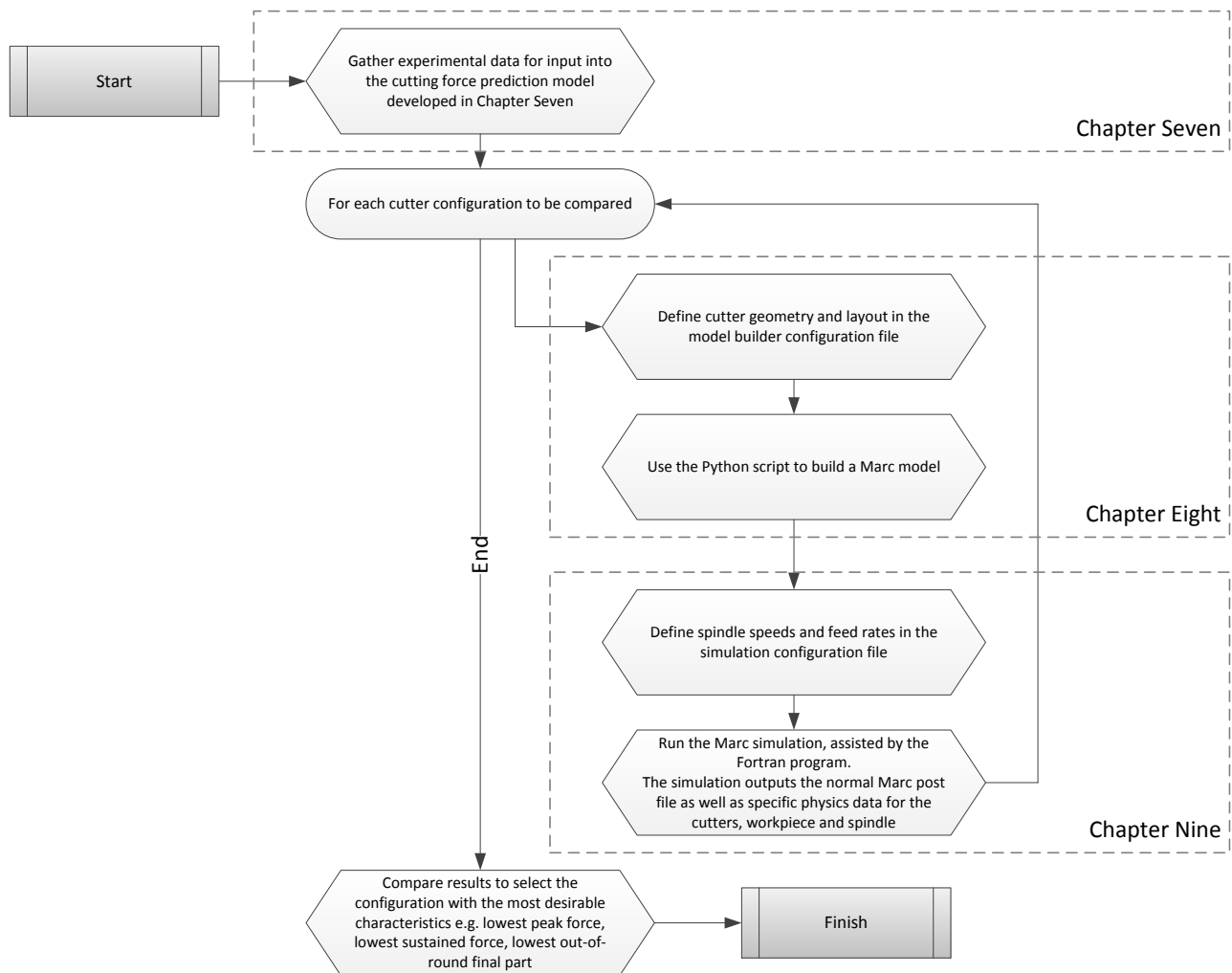


Figure 8-2 – User workflow

Figure 8-3 shows the development environment diagram. The diagram shows the control relationship and data flow between the key components discussed throughout the remainder of this thesis. The branches coloured blue and red relate to development and debugging and are not required for the end user.

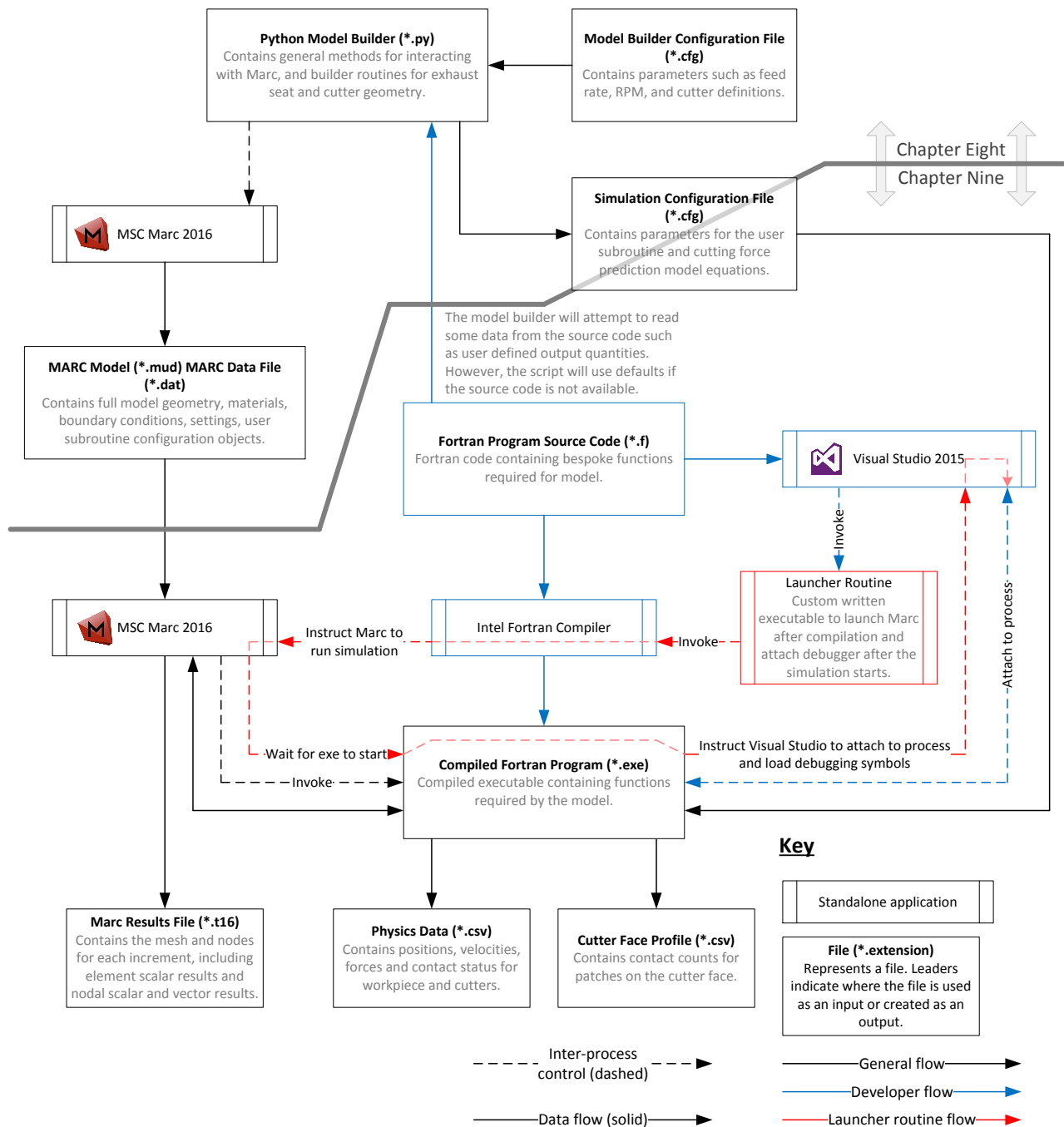


Figure 8-3 – Development environment design

As the diagram shows, this work also calls for the development of a number of ancillary programs to make Marc work with Microsoft Visual Studio. These are discussed in greater detail in Chapter Nine.

## 8.4 Model Design and Performance Constraints

In order to model the machining operation from start to finish, it is necessary to process many spindle rotations, potentially thousands of increments. For this reason, great care must be taken to simplify every aspect of the model both in the geometry input stage and the Fortran program processing stage.

This section deals with presenting the model that will be built using the parametric generator script.

### 8.4.1 Assumptions

The following assumptions are made in order that the model can be simplified as much as possible:

- no significant vibration enters the valve seat from the machine fixture (e.g. from other machines on the production line. This assumption is made since the machine is isolated from the factory floor by a spring-damper foundation.);
- the valve seat fixture is perfectly rigid, to represent the most ideal case;
- the machine turret, quill and spindle are all perfectly rigid since their CNC machine equivalents are significantly more rigid than the workpiece;
- the tool holder is perfectly balanced when rotating unloaded, since the tool supplier balances the tools before delivery;
- cutting inserts form a rigid connection to the tool holder, to represent the most ideal case;
- all components are at room temperature - 20°C since the production line is held at room temperature;
- there is no wear or damage (such as chipping) on inserts throughout the simulation (the inserts are modelled as new);
- forces due to interaction with fluids (coolants and lubricants) and gasses (narrow air gap between workpiece and tool holder) are negligible, since there is no reason to suspect the momentum of fluids affects balancing of the operation;
- the valve seat material is homogenous as specified by the supplier datasheet (this is only true down to a depth of 3.4 mm, however no material below this point is removed);
- the pressure applied to the outside of the valve seat from the interference fit in the cylinder head has negligible effect on the machining process. As the valve seat is thinned, one might expect this pressure from the interference fit to cause the valve seat to deform due to its diminished strength. It is assumed that if this process is occurring, it is negligible;
- spindle rotational velocity is constant, even under load and is not affected by power supply ripple or phase since the production line machines are computer controlled to maintain spindle speed and feed rate.

### 8.4.2 Coordinate System

From a design and programming point of view, although it would be greatly beneficial to use a fixed coordinate system (for example, feed always in the negative Z direction) this would limit the layout of the simulation. One of the design objectives of this work is to produce a model that could work with valve seat



geometry embedded in flexible cylinder head geometry. This would greatly enhance the usefulness of this work by allowing end users to visualise the magnitude of fixture displacement, which in turn could help to quantify out-of-round error during cutting.

For rigid fixture simulations, the components are laid out as shown in figure 8-4. The valve seat base is aligned with the XZ plane. The centre axis of the valve seat is coaxial with the Y axis. The feed direction is along negative Y axis. Each cutter has its own local coordinate system representing the local rake, radial and feed directions.

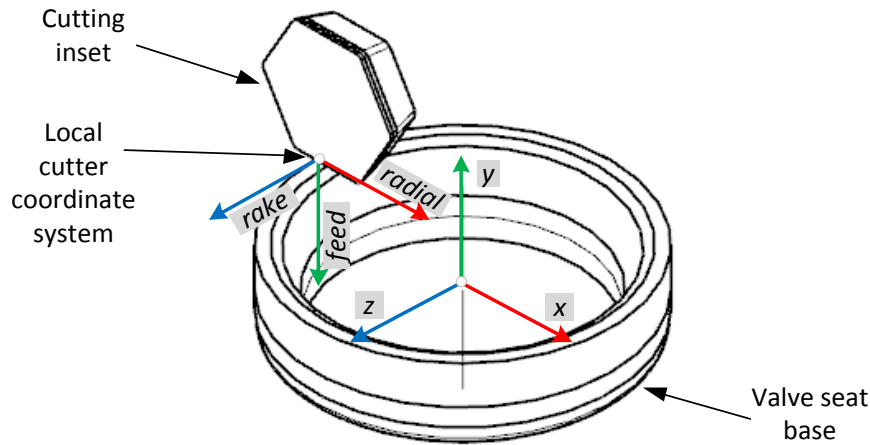


Figure 8-4 – Coordinate system

### 8.4.3 Geometry

Figure 8-5 shows the simulation geometry with a cross section of the valve seat and three cutter edges at various angles.

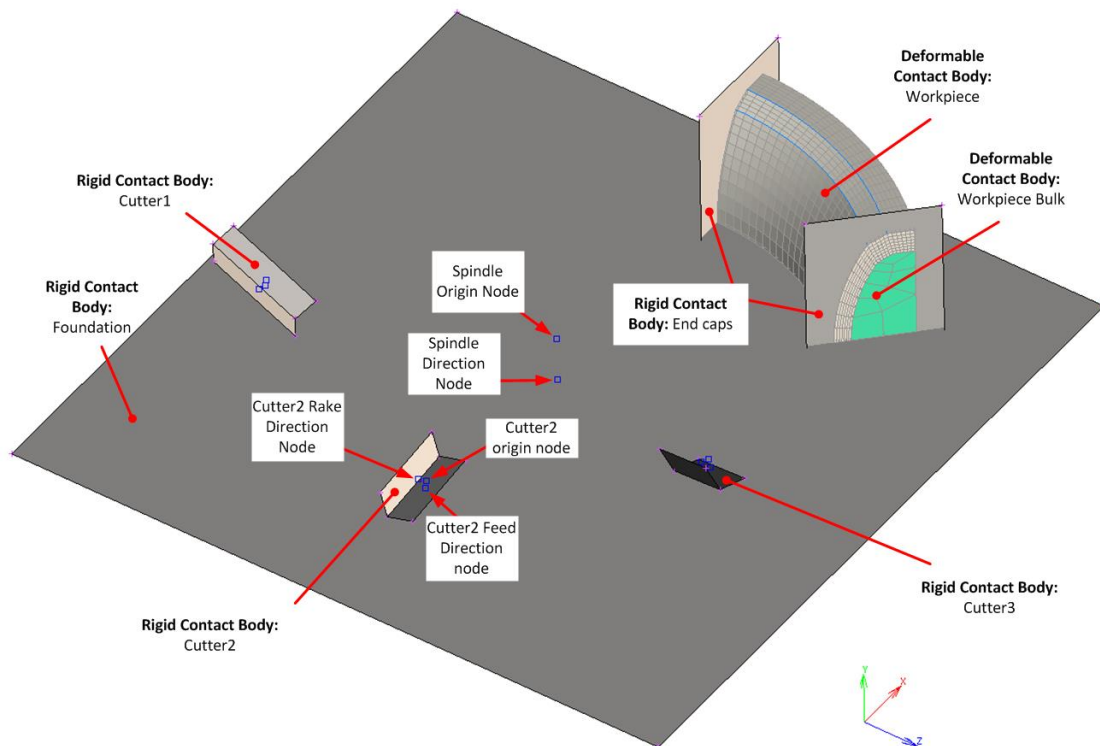


Figure 8-5 – Simulation domain diagram

The cutting insert geometry is reduced down to just the portion of the cutting edge in contact with the workpiece. All non-contacting parts are omitted. Furthermore, there is no cutter radius as its effect is not significant since no plasticity or chip flow is modelled directly. Each cutting insert has three nodes associated with it to define the local coordinate system. The vector from the origin node to the rake direction node, defines the local rake axis and likewise the vector from the origin node to the local feed direction node defines the feed axis.

Similarly, there are two nodes defining the spindle axis. Cutters rotate around a vector from the spindle origin node to the spindle direction node according to the right hand rule. Modelling the spindle as rigid is considered to be a fair assumption since it was discovered during experiments described in Chapter Six that the aluminium valve seat head and head fixture was by far the most flexible part of the cutting system.

The workpiece geometry is split into two parts, the *workpiece bulk* and *workpiece* itself. These bodies are separate to reduce the number of nodal proximity checks required, and to reduce the local volume near the cutter that must be remeshed during simulation.

The workpiece is attached to a rigid foundation plane. It must be stressed however, that this is a simplification imposed for development of the model and it is not a limitation of any part of the model developed in this chapter and Chapter Nine. Further work investigations can easily remove this rigid base plane and instead attach the workpiece to another deformable mesh using a glue contact i.e. a cylinder head on a fixture.

For sub segments, like the one modelled in this work, the two exposed ends of the ring are glued to a rigid plane. This better represents a sub-segment of a full ring model by adding resistance to distortion in response to cutting torque.

At the start of simulation, the workpiece geometry is represented by 8-node, hexahedral, Marc type 7 elements. These elements are used because they are very easy to generate in a parametric script, furthermore, they very accurately and uniformly represent the workpiece geometry.

When the simulation starts, this mesh is immediately remeshed into 4-node, tetrahedral, Marc type 157 elements as shown in figure 8-6. Tetrahedral elements are used (instead of hexahedral elements) during simulation due to the ability of tetrahedral elements to transition from fine to coarse elements whilst maintaining the aspect ratio of all elements in the mesh. This feature allows a tetrahedral mesh to be adjusted to conform to the new cut geometry. This is not possible with hexahedral brick elements in Marc because hexahedral elements can only be adjusted during simulation by subdivision and adjacent elements with similar edges must share at least one node on that edge.

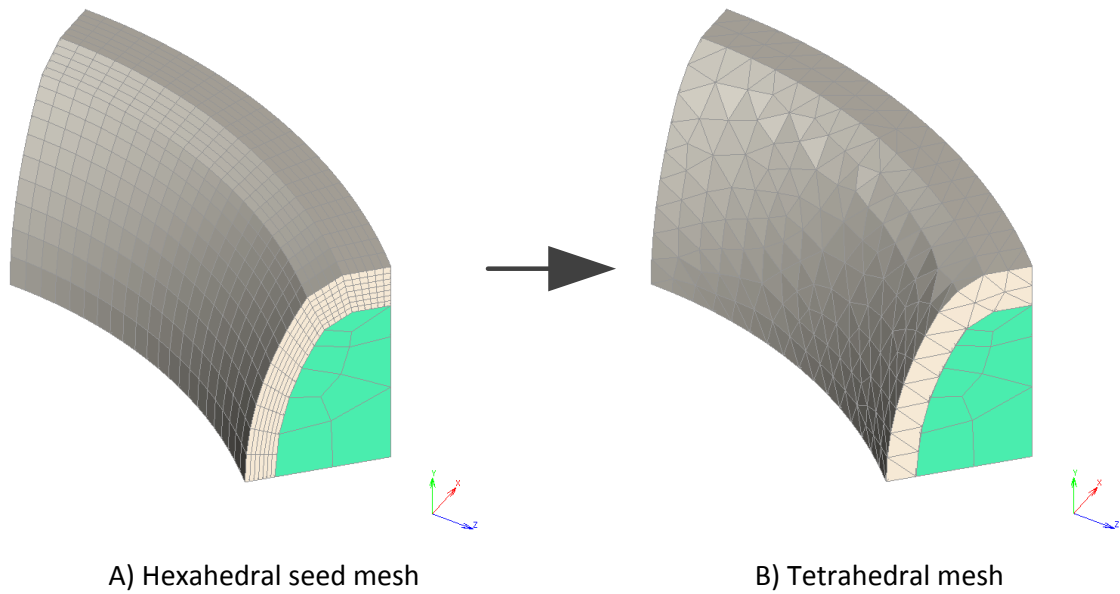


Figure 8-6 – Tetrahedral remesh of hexahedral workpiece mesh

Figure 8-7 shows a series of brick element subdivisions from  $n=1$ , to  $n=3$ . As the figure shows, element count increases exponentially with each subdivision. This is unsustainable given that the cutter must make multiple passes.

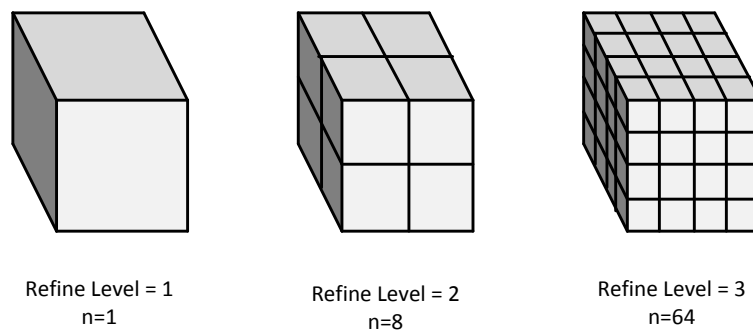


Figure 8-7 – 3D brick element subdivisions

Using the UMAKNET Marc user subroutine, it is possible to set the positions of new nodes introduced to accommodate subdivision, furthermore it is also possible to set the subdivision order in each direction of brick elements. This can drastically reduce the number of elements required to satisfy a split, however there are further caveats that cannot be overcome when using brick elements.

Marc requires that the transition between brick elements is limited to single steps in refine level. Consider the diagram given in figure 8-8. The number inside each element shows the refine level, where 1 is the initial level. A) shows two connected brick elements that share two nodes, B) shows the state of the mesh after the left hand side element in A) has been subdivided. Both A) and B) represent legal mesh states. C) shows the mesh when the level two elements have been subdivided again. This state is illegal because the difference in refine levels between adjacent elements is greater than one. This kind of subdivision would force the right hand side element in C) to subdivide as indicated by the dotted line, to satisfy the transition criteria.

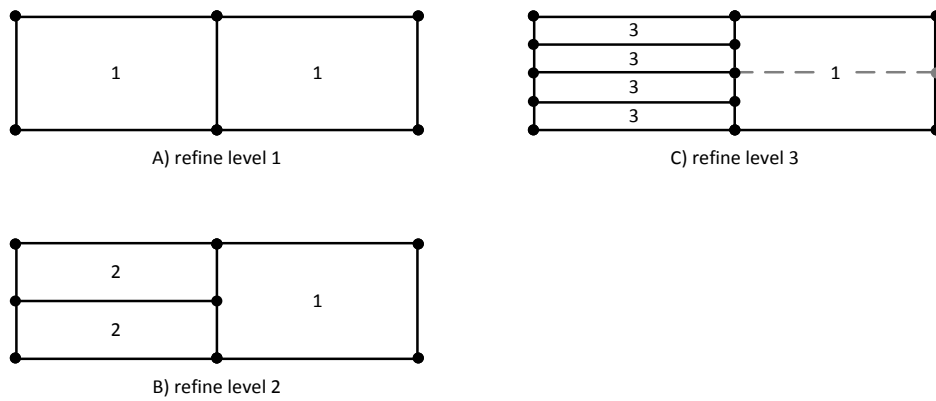


Figure 8-8 – 2D brick element subdivisions

This property quickly leads to runaway subdivision due to historical subdivisions. Consider the diagram given in figure 8-9. Mesh A1) shows the proposed split line overlaid on the mesh. This split is legal because none of the resulting elements will be adjacent to an element which differs by more than one refine level, as shown in B1).

The split line is then moved over the new mesh in A2) and a new split is calculated as shown in B2). This time, the new elements created from splitting level two elements are all level three. Some of these elements are adjacent to a level one element, thus forcing its subdivision.

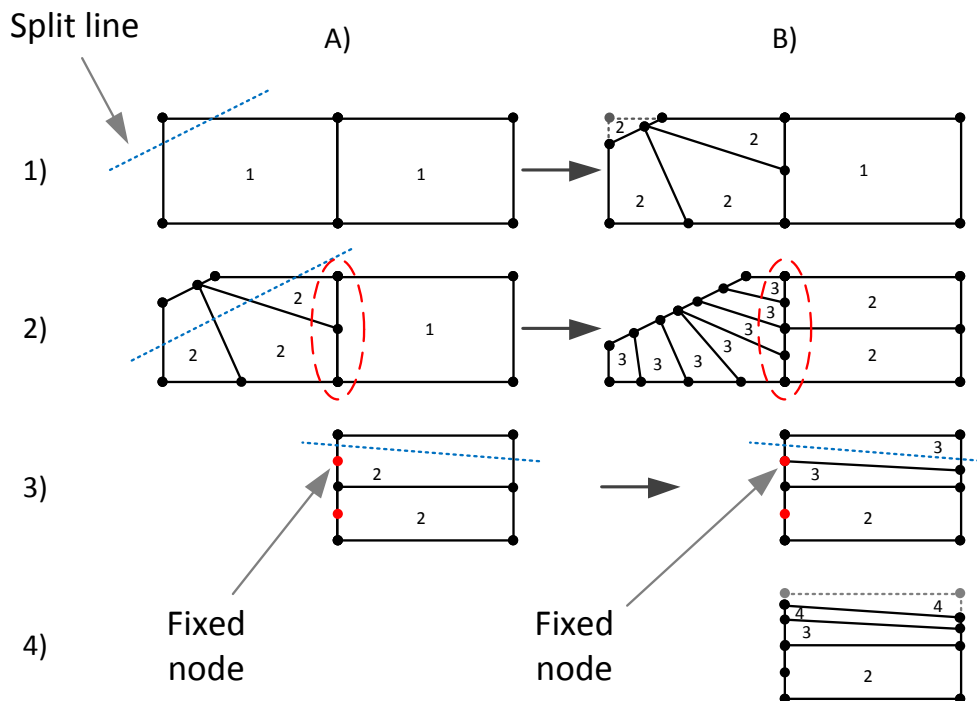


Figure 8-9 – Hexahedral element splitting

The next split that takes place in A3) is blocked by the fixed node (indicated in red). To split this geometry a sacrificial subdivision must be made to consume the fixed node. After that, the mesh is free to be subdivided according to the blue split line as shown in B3), resulting in the mesh shown in B4).

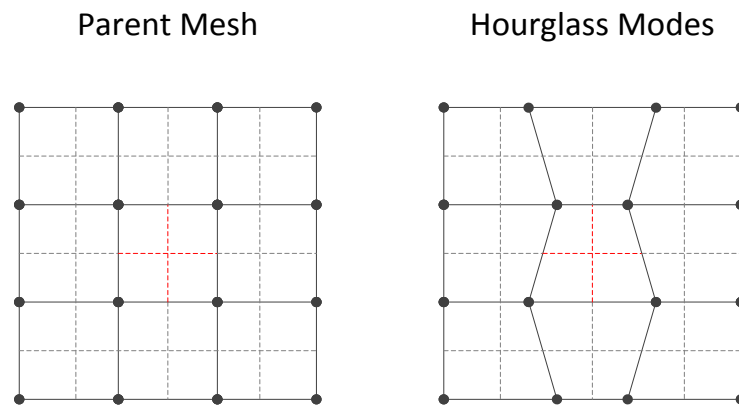
Performing such sacrificial subdivisions simply to free nodes is obviously not sustainable, particularly when those subdivisions trigger the wasteful subdivision of surrounding elements. Hexahedral subdivision

methods can work for very simple cutting models (e.g. models that could be represented in 2D), but after several passes, especially if several different cutter angles are used, the method quickly breaks down.

Tetrahedral elements do not suffer from this fundamental limitation, but the programming effort required to generate local tetrahedral meshes is extreme, but nonetheless possible and necessary to deliver the objectives of this work.

The workpiece bulk mesh is represented by 8-noded, hexahedral, Marc type 117 elements. Type 117 elements are a reduced integration version of type 7 elements. This element type is selected to take advantage of the reduced computational effort required to process these elements between increments. A reduced integration element is considered acceptable in this case as the valve seat bulk is not expected to undergo any large deformation.

Reduced integration elements are traditionally vulnerable to hourglass modes, which is characterised by a warping of elements to form an hourglass shape as shown in figure 8-10.



*Figure 8-10 – Hourglass modes*

This happens because reduced integration elements are formulated using a single integration point at their centre. Energy balance is reached based on a measure of distortion that uses the distances between the integration point (at the centre of the element) to the midpoints of the boundaries (lines connecting nodes). These distances are marked in the figure for one element by the dotted red lines. For both the parent and hourglass modes mesh diagrams, the lengths of, and angles between the red lines are the same. The diagrams demonstrate that this element has no way of resisting this kind of distortion, since the connecting lines, integration point and midpoint positions remain unchanged.

Fortunately however, Marc type 117 elements are protected against this weakness by the inclusion of an additional stiffness term, whilst maintaining the majority of the performance increase offered by reduced integration elements.

#### **8.4.4 Boundary Conditions**

There is only one explicitly prescribed boundary condition in the Marc model, which is the workpiece reaction force load. All workpiece elements are assigned to a element volume pressure load, managed by the Marc user subroutine, FORCEM. User subroutines are described in greater detail in Chapter Nine.

Movement of the cutters is controlled by the MOTION user subroutine. However, no explicit boundary condition needs to be applied in Marc. The cutters are detected automatically during simulation and their positions are updated according to the effective spindle speed and feed rate at that time.

The foundation plane is defined as a position controlled rigid body and is therefore fixed by default.

#### 8.4.5 Contact

Contact between the cutters and workpiece is entirely controlled by user subroutines (described in greater detail in Chapter Nine).

There are three contact definitions in the model, which are as follows:

- the workpiece bulk is attached to the foundation plane using a glue constraint;
- the workpiece is attached to the workpiece bulk via a glue constraint; and
- for sub-segment models, the two exposed ends of the workpiece and workpiece bulk are glued to rigid end cap surfaces.

There is no direct contact between the workpiece and foundation to avoid over constraining the workpiece.

#### 8.4.6 Material (Novofer AR20)

The valve seat material is Novofer AR20, a copper-infiltrated high speed steel. For purposes of this simulation, under smooth cutting conditions this material is not required to deform plastically, nor is its damage behaviour required to decide when to deactivate elements.

The Young's modulus of the valve seat is taken to be 150 GPa, as indicated on the material datasheet (Bleistahl, 2006). Poisson's ratio is estimated to be 0.3 based on similar powder metallurgy compositions (ASTM, 2004). This material property is applied within MARC to both the workpiece and workpiece bulk contact body elements.

#### 8.4.7 Additional Output

A key metric of interest to engineers is the duty cycle of each cutter. Aside from imbalanced forces, the cutting system studied in this work also imposes imbalanced duty cycles on each cutter. This means that after the first cut, cutting inserts will continuously exist at different states of wear throughout subsequent cuts. Figure 8-11 below shows the section view of a valve seat blank with the dimensions of the finished cut overlaid in red.

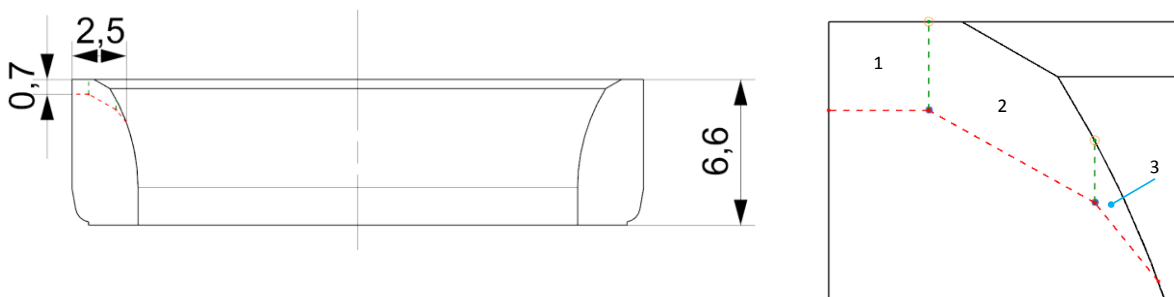


Figure 8-11 – Valve seat material removal zones

Figure 8-11 shows three zones, each corresponding to the volume of material removed by each cutter respectively. As the diagram shows, the material removed by the cutter responsible for zone two is much larger in volume than that which is removed by the third cutter in zone three. Naturally it is reasonable to expect the middle cutter to wear at a much faster rate than the innermost cutter.

Wear acts to decrease the cutting radius of the tool, as well as damages the flank and rake faces. As previously discussed in literature (Lacerda and Siqueira, 2012), an increase in cutting radius alone can have a dramatic effect on the amplitude of vibration during the cutting process and therefore some cutters will

contribute to vibration more than others. It is therefore important that the varying duty cycle on the cutters can be predicted during simulation.

This functionality is implemented by a feature of the Fortran program developed in Chapter Nine that tracks the contact areas on the cutter face and outputs the cutting power dissipated per unit area into a comma separated values (CSV) file. The CSV file can be loaded by a support program written in Python which plots the distribution over a diagram of the cutter face using a heat map colour gradient to distinguish areas of low and high contact duty cycle.

#### 8.4.8 Initial Parameter Configuration

The initial parametric configuration is based on a tool trial conducted by Ford at their Bridgend engine manufacturing plant in Wales, United Kingdom. In their experiment, pcBN tools were compared to coated tungsten carbide tools. The tests confirmed that pcBN inserts were prone to minute chipping which left difficult to detect defects on the valve seat. The parameters of their experiment are given in table 8-1.

Parameter	Value
Cutting Speed ( $\text{m min}^{-1}$ )	120.00
RPM	1646.00
Feed rate ( $\text{mm min}^{-1}$ )	66.00
Feed rate ( $\text{mm s}^{-1}$ )	1.10
Cut time (s)	6.22
Expected tool life (min)	27.00
Expected tool life (parts)	450

*Table 8-1 – Table of parameters known to lead to failure of pcBN inserts*

These parameters were based on manufacturer recommendations and no other parameters were tested. These parameters are known to lead to a scenario in which chipping will spontaneously occur on pcBN inserts.

### 8.4.9 Parametric Compatibility

A key component of the model is the ability to simulate the machining operation under varying parameters. As discussed in the introduction to this chapter, this ability is implemented using a Python script.

The Python script loads a configuration file containing the parameters for the desired simulation. The user may keep multiple such configuration files and does not need to alter the Python script itself to change the simulation parameters. Table 8-2 gives the list of available parameters that a user can specify in the configuration file.

Parameter Name	Description	Units	Required
<i>toolRPM</i>	The spindle speed in revolutions per minute (RPM).	RPM	Yes
<i>feed rate</i>	The spindle feed rate.	mm / rev	Yes
<i>depthToRunTo</i>	The minimum depth to simulate, after which the simulation stops.	mm	Yes
<i>degreesPerIncrement</i>	The target number of degrees to jump per increment (temporal resolution).	Degrees / increment	Yes
<i>seatAngle</i>	The user can opt to only simulate a segment of the valve seat to increase performance. This approach yields data that is not valid for a full force balance unless all cutters contact the slice at the same time, however the option is useful for testing and validation.	Degrees	No (default = 360)
<i>degreesPerSlice</i>	The element thickness resolution around the perimeter. Element thickness will target the number of degrees specified here.	Degrees	Yes
<i>rapid</i>	Rapid mode is only applicable when the user specifies a <i>seatAngle</i> of less than 360 degrees. When enabled, if the Fortran program detects that none of the cutters are near the workpiece, the spindle speed will temporarily accelerate. This mode can dramatically reduce the number of increments required to simulate a segment.	(Boolean)	No (default = False)
<i>rapidMultiplier</i>	See 'rapid' above. This setting specifies how much faster rapid speed is than normal speed.	(multiplier)	No (default = 1)
<i>processorCount</i>	Allows the user to specify how many processor cores to use on the system. If omitted the script will use one core.	(count)	No (default = 1)

Table 8-2 – Configuration file parameters

In addition to these parameters, the user must also use the configuration file to specify the cutter configuration. A cutter configuration is made according to the following example:

```
[CUTTER]      0.0,    9.418762,    0.65+0.349i,    0.0,    4.5,    0.0,    x6
                1        2                3        4        5        6        7
```



The definition starts with the keyword [CUTTER] and is followed by seven comma separated parameters. These parameters are as follows:

1. inclination – the inclination of the cutter rotated about it's local feed axis;
2. radius – the distance from the spindle to the control node of the cutter;
3. height – the relative height offset (all heights are automatically adjusted by the script so that cutting starts very soon into the simulation and no increment cycles are wasted on simple motion);
4. angle – the angle of the cutter control node about the rotation axis of the spindle;
5. width – the width of cutting edge to generate;
6. slide offset – the control node is not necessarily at the midpoint of the portion of cutter face in contact with the workpiece, so this parameter allows the user to correct this by sliding the width of cutter as defined earlier, radially towards or away from the spindle;
7. type code – which refers to the type of insert. This is used for labelling purposes and, in output files, to allow automatic type detection when plotting the cutter face duty cycle over the outline of the applicable cutter.

## 8.5 Marc / Python Interface

One of Marc's best features is its ability to be controlled by an external Python script. Almost every operation in Marc, including geometry building, adding elements, defining materials etc. can be executed by feeding commands directly to the command line window within the user interface, shown in figure 8-12.

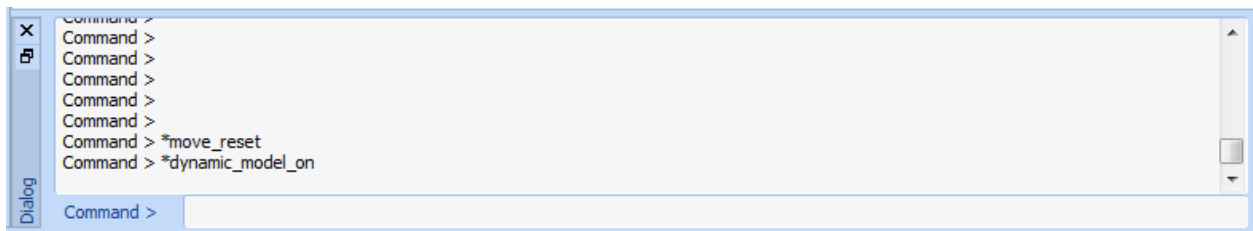


Figure 8-12 – Marc command line window

The interface is made available by importing the *py\_mentat* library (which is provided with Marc along with Python version 2.7.1). This library makes available a very limited number of commands in Python, which can be used to send any number of a much larger array of commands to Marc. The commonly used Python commands are [py\\_send](#), [py\\_get\\_int](#), [py\\_get\\_float](#) and [py\\_get\\_string](#).

Although it is possible to generate complex geometry using just these four commands to send Marc commands and retrieve data, it is very cumbersome. This is both a strength and a weakness of the Marc / Python interface. Its strength lies in the absolute power it provides over the Marc interface, but its weakness is in the complexity of code required to model effectively.

When sending instructions to Marc, the exact command must be sent as a string. Consider the following examples:

Example 1, to add a node at the coordinates,  $x = 0, y = 0, z = -2$ :

```
py_send("*add_nodes 0 0 -2")
```

Example 2, to rotate a node about the  $x$  axis by 90 degrees:

```
py_send("*move_reset")
py_send("*set_move_rotation x 90")
py_send("*move_nodes 1 #")
```

By far the biggest problem with creating geometry in this way is keeping track of entity IDs. In the examples above, the second example would affect the node added in the first example. However, the user can only be sure of this since it is observed that no nodes already exist in the model and then one node was added which Marc will number sequentially as node 1, then the move command in example 2 is executed which refers to node 1. If however, the model was already partially populated before the node was added, the user would have to try something similar to the following to ensure they use the correct node ID:

```
py_send("*add_nodes 0 0 -2")
lastNodeId = py_get_int("node_id(-1)")
py_send("*move_reset")
py_send("*set_move_rotation x 90")
py_send("*move_nodes {} #").format(lastNodeId)
```

These issues make for messy, repetitive code and compound as the complexity of the model grows. A further level of abstraction would be helpful in this case. For this reason, a separate Python script of approximately 1,000 lines was developed, called MarcTools that provides this abstraction. A technical description of MarcTools is given in Appendix A. This script was kept separate to the model script to make it easier to use in other projects and share with colleagues. MarcTools has a number of powerful features to make interfacing with Marc much easier. MarcTools maintains a parallel database of entity IDs, freeing the user from worrying about this aspect. Using MarcTools, examples 1 and 2 above can be reduced to one line as follows:

```
MarcNode(z=-2).Rotate(tx=90)
```

MarcTools seamlessly manages node IDs, caches data to reduce communication with Marc, prevents contamination between uses of user interface tools (e.g. when the user may forget to reset user interface tools) and protects against other traps that commonly slow development. MarcTools also incorporates a powerful feature selection tool, e.g. selecting entities in a box, along a path, by association, etc.

## 8.6 Python Model Generator Script

This section deals with the technical aspects of the model generator script itself, including how it interfaces with Marc, how it works and how the model design considerations raised in section 8.4 are implemented within the script.

The model generator script is approximately 1,200 lines and relies heavily on MarcTools discussed in the previous section, to create and manipulate geometry. Figure 8-13 shows the order in which the model generator script builds the model.

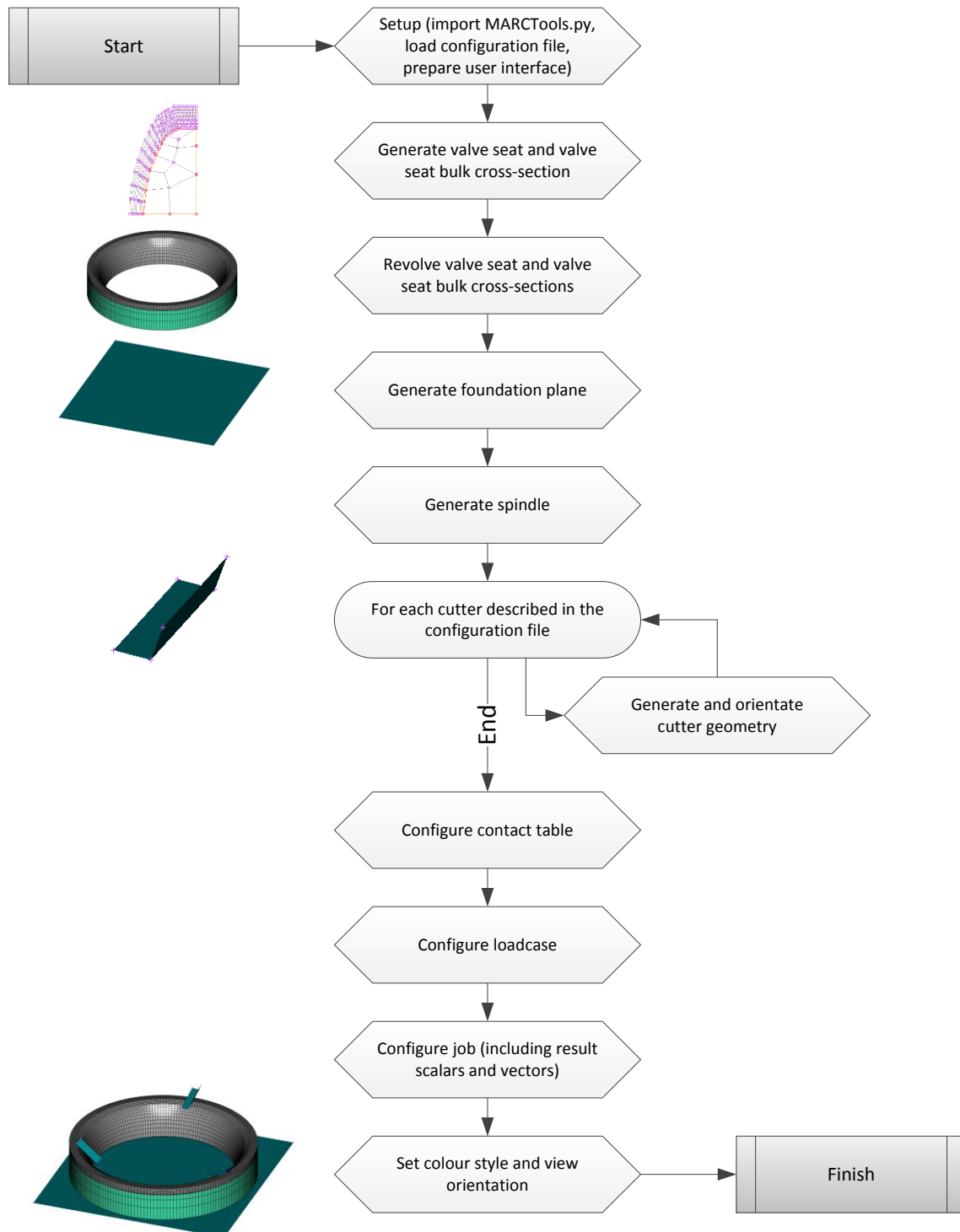


Figure 8-13 – Model builder task order

### 8.6.1 Setup

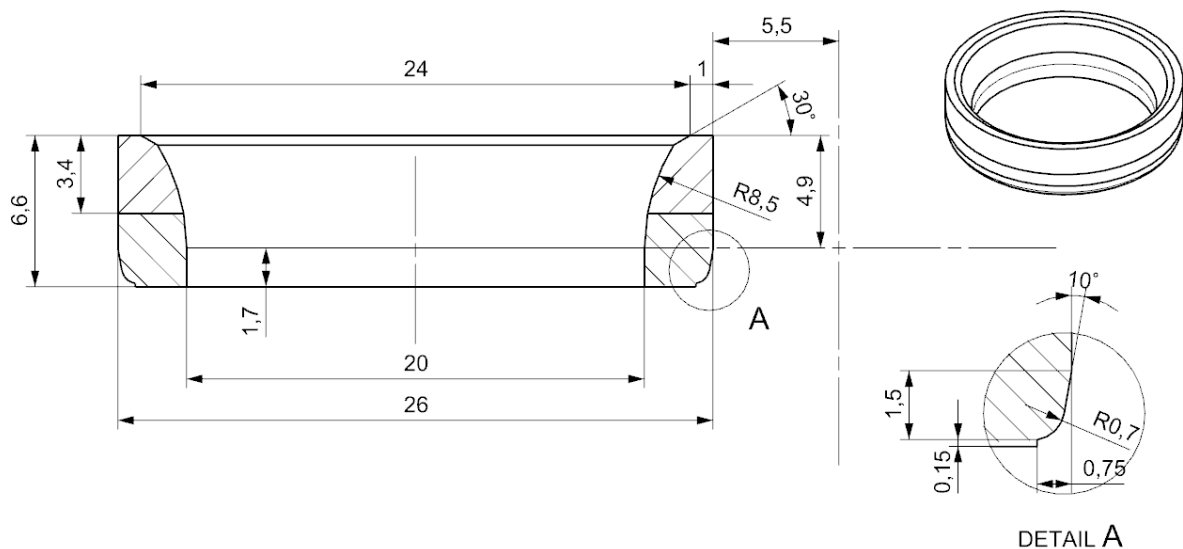
The user calls the model generator script from the Marc user interface. The first responsibilities of the script are as follows:

- check that MarcTools.py is present in the current working directory and import if possible;
- check for the presence of a parametric configuration file in the current working directory. If present, then parse its contents, and store the settings in a global dictionary named configuration;
- clear the model domain;
- disable drawing (this speeds up geometry building by instructing Marc not to draw every incremental change);
- set the model length mode to millimetres; and
- create the "steel" material type in Marc according to the specification given in section 8.4.6.

### 8.6.2 Valve Seat

The valve seat generation phase encompasses the generation of three contact bodies, the workpiece, workpiece bulk and foundation plane. The valve seat can be generated in both closed and open ring configurations depending on the *seatAngle* configuration parameter. The angular element density is controlled by the *degreesPerSlice* configuration parameter.

Figure 8-14 shows the dimensions of the valve seat cross section. This is the geometry that the valve seat generation in this model generator script is based on.



*Figure 8-14 – Exhaust valve seat blank dimensions*

Figure 8-15 shows a cross section of the valve seat before and after machining. As the figure shows, the difference is minimal, and the lower portions of the seat are not touched at all. For this reason, only the upper 4.9 mm of valve seat geometry is modelled.

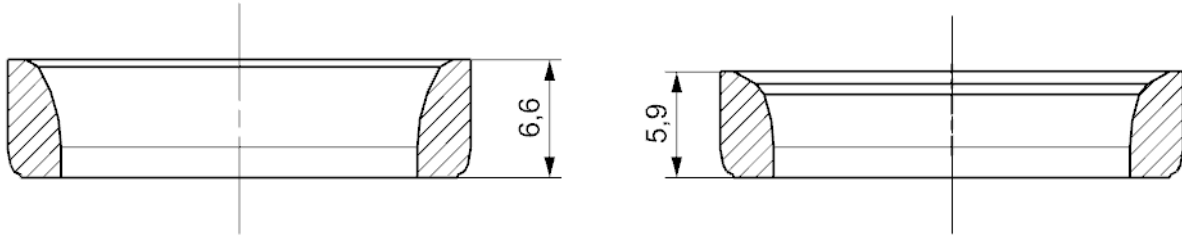


Figure 8-15 – Valve seat cross section before (left) and after (right) machining

Construction of the valve seat starts by adding cross section nodes, as shown in figure 8-16 A). The following commands are used to construct the points shown in A):

```
pt0 = MarcPoint(x0,y0).Make()
pt1 = MarcPoint(x1,y1).Make()
pt2 = MarcPoint(x2,y2).Make()
pt3 = MarcPoint(x3,y2).Make()
```

Likewise, the following commands are used to construct the profile curves of the valve seat shown in B):

```
line1 = MarcCurve(MarcPoint(x_arc,y_arc),pt1,pt0)
line2 = MarcCurve(pt1,pt2)
line3 = MarcCurve(pt2,pt3)
```

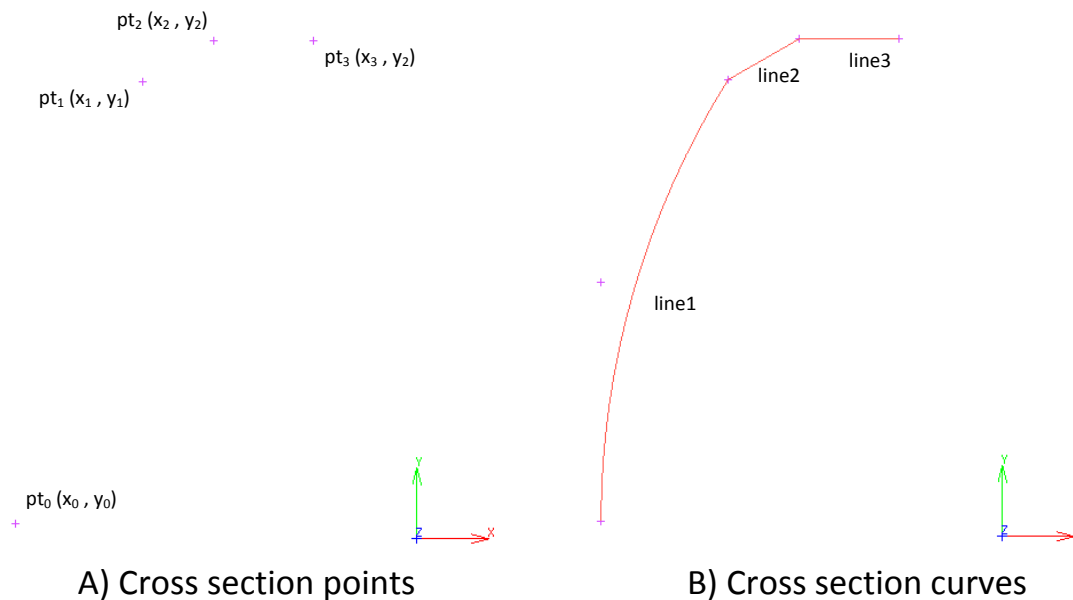


Figure 8-16 – Cross section points and curves

Next, curve divisions are applied evenly to lines 2 and 3, and transitionally to line 1. Meshing these divisions yields the line element mesh shown in figure 8-17 A). This line mesh is expanded using the following expand tool command to yield the quadrilateral mesh shown in B):

workpieceMesh =

```
Expand(rep=n,ox=o.x,sX=scale,sY=scale,*Filter(MarcElement,workpieceMesh))
```

In this command, o is the origin **MarcPoint** object, n is the number of repetitions, scale is a scale factor less than 1, but greater than zero, and workpieceMesh initially contains the **MarcNode** and **MarcElement** objects that make up the line mesh shown in A), which is later updated by **Expand** to contain the **MarcNode** and **MarcElement** objects that make up the quadrilateral mesh shown in B).

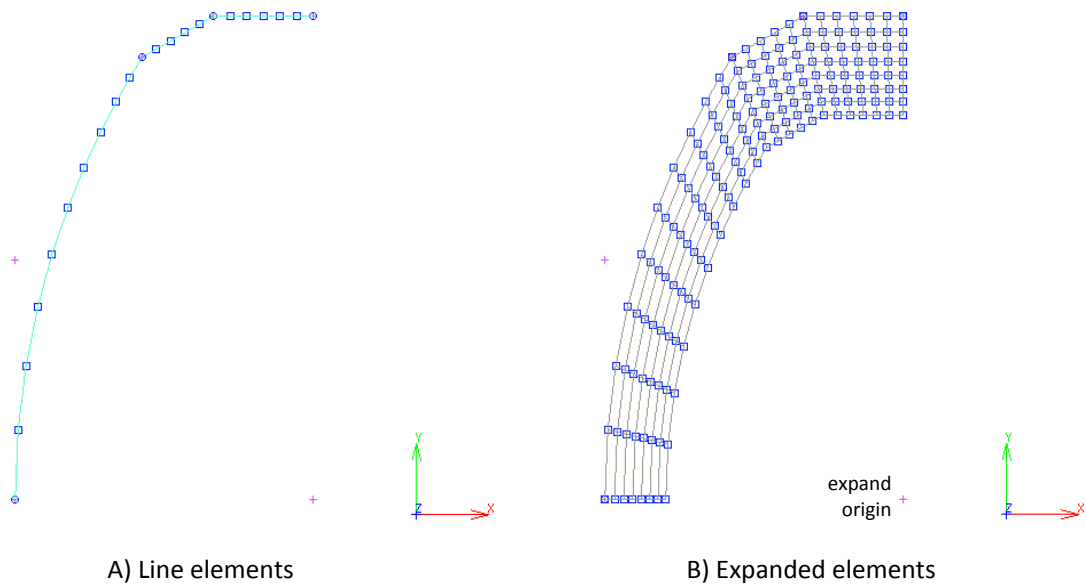
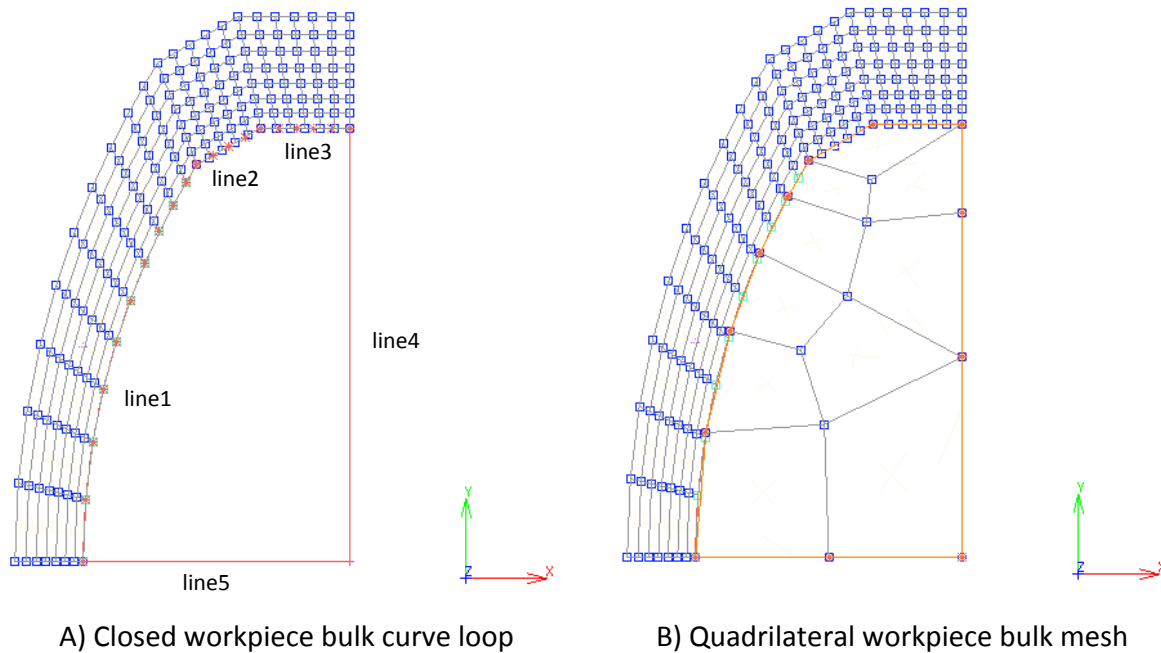


Figure 8-17 – Cross section element expansion

The construction lines generated in the initial step are then transformed to the inner edge of the new mesh using the **Move** command, these curves are then closed by adding two additional curves that attach to the origin node as shown in figure 8-18 A).



*Figure 8-18 – Cross section workpiece bulk mesh*

A **MarcCurveLoop** is created from the new set of curves, which is then meshed using the **QuadMesh** function to yield the mesh shown in B). This mesh is created using the following command:

```
bulkMesh = MarcCurveLoop(line1,line2,line3,line4,line5).QuadMesh()
```

This concludes construction of the valve seat cross section. However, before revolving the cross section to generate the 3D geometry, a number of configuration tasks are performed as follows:

- element type 7 is applied to all elements in the workpieceMesh list;
- element type 117 is applied to all elements in the bulkMesh list;
- the Steel material type is applied to all elements in both the workpieceMesh and bulkMesh lists;
- a foundation contact body, and foundation plane geometry is created.
- two deformable contact body entities are created in Marc to represent the workpieceMesh and bulkMesh elements respectively; and
- all elements in the workpieceMesh list are assigned a force / volume boundary condition, which is set to defer to the user subroutine FORCEM.

The final step of workpiece generation is to revolve the cross section around the spindle axis, according to the *seatAngle* parameter loaded from the configuration file, yielding the geometry as shown in figure 8-19. Element properties set in the steps above are automatically inherited by the 3D mesh generated by revolving the 2D mesh.

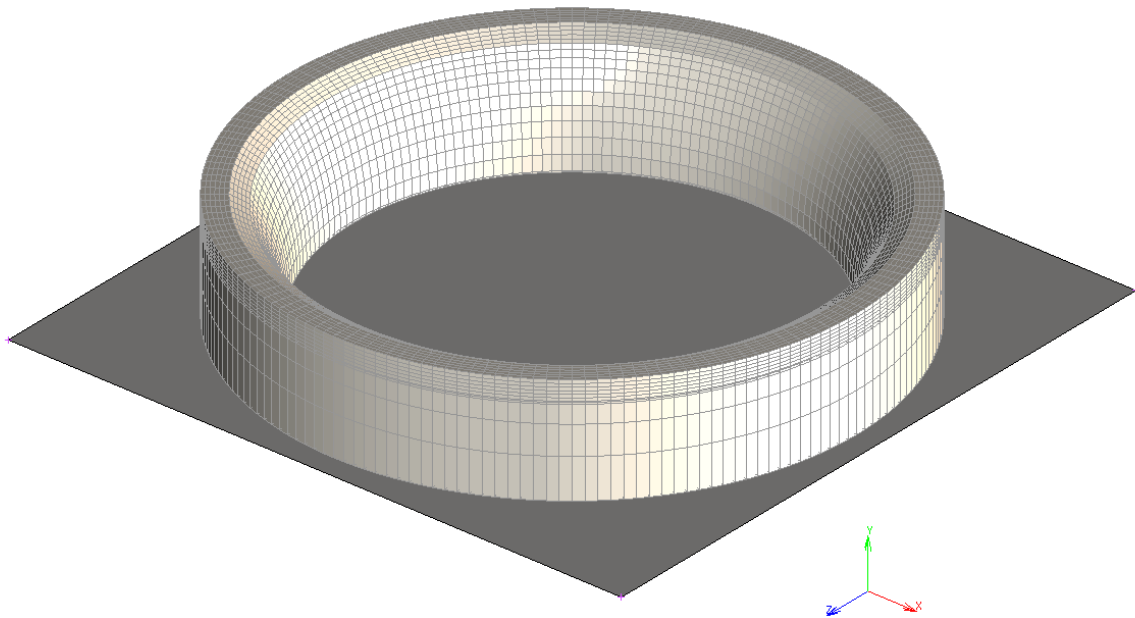


Figure 8-19 – Valve seat and foundation plane geometry

### 8.6.3 Spindle

The spindle is little more than just an axis of revolution definition. This definition is represented by two nodes, a spindle origin node, and a spindle direction node.

Positive rotation about the spindle occurs according to the right hand rule, where the vector from the origin node, to the direction node is the axis of rotation. The spindle axis vector is calculated by the Fortran program developed in Chapter Nine. It is not necessary to normalise the vector in the Marc model, as this is done automatically by the Fortran program.

For this development model, the spindle rotation axis is collinear with the model Y axis. Therefore these nodes are created according to the following commands:

```
spindleOriginNode = MarcNode().Make()  
spindleDirNode = MarcNode(y=-1.0).Make()
```

The first line creates a node at  $x = 0$ ,  $y = 0$ ,  $z = 0$ , and likewise, the second line creates a node at  $x = 0$ ,  $y = -1$ ,  $z = 0$ .

There is no dedicated tool to assign a special meaning to a particular node in Marc. However, it is possible to create and name sets that contain just one node. These sets can be queried during simulation by the Fortran program. This serves as a convenient method of designating particular nodes, for both the spindle axis definition and cutter axes definitions.



These sets are simply added according to the following two commands, where *spindle\_origin* and *spindle\_dir* are the set names that the Fortran program will look for at initialisation:

```
py_send("*store_nodes spindle_origin {} {}".format(spindleOriginNode.MarcId()))
py_send("*store_nodes spindle_dir {} {}".format(spindleDirNode.MarcId()))
```

#### 8.6.4 Cutters

All cutters in the model are defined as rigid contact bodies, represented by one or more NURBS surfaces. Since only a very small portion of each cutter has the potential to contact the workpiece, most of the cutter geometry is omitted. Figure 8-20 shows an example of the modelled portion of the cutter shown in red, relative to the full cutter geometry.

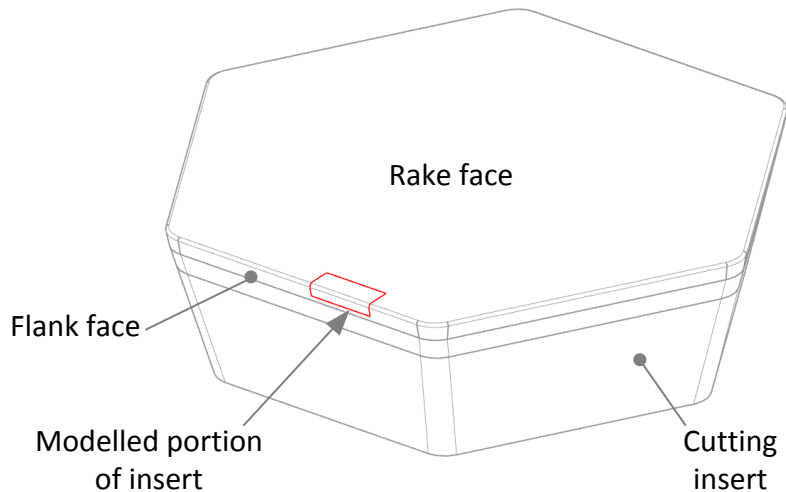


Figure 8-20 – Modelled portion of cutter geometry

Furthermore, since there is no chip flow or plasticity being directly modelled in this simulation, the radius of the cutter has no effect, but does however complicate remeshing. For this reason, the 30  $\mu\text{m}$  cutting radius is also omitted.

All cutters are generated on the model X-Y axis before being transformed to their final positions as dictated by the configuration file. Figure 8-21 shows the side view of a cutter after construction of the 2D profile.

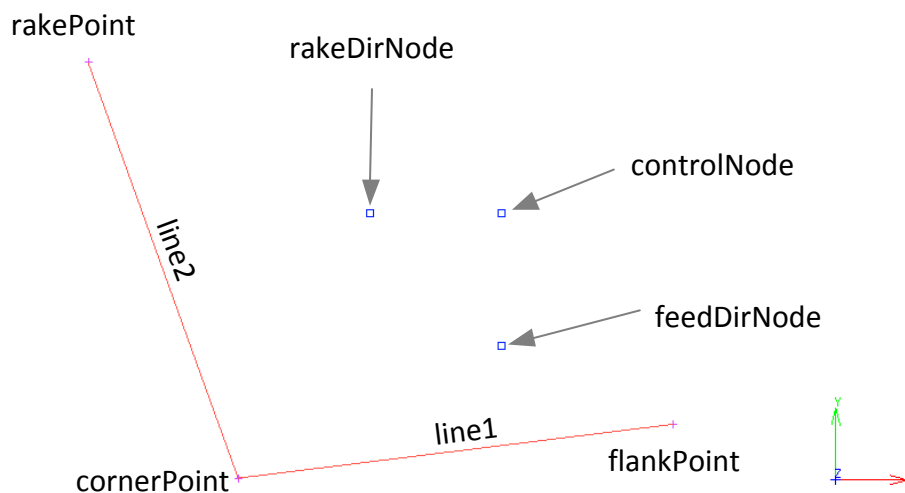


Figure 8-21 – Side view of cutter geometry construction

Each cutter requires a local coordinate system definition for compatibility with the cutting force prediction model developed in the previous chapter. These definitions are made by adding an origin node, feed direction node and rake direction node according to the following commands:

```
originOffset = 0.6
controlNode = MarcNode(originOffset, originOffset).Make()
feedDirNode = MarcNode(originOffset-0.3, originOffset).Make()
rakeDirNode = MarcNode(originOffset, originOffset-0.3).Make()
```

As with the spindle direction nodes, these nodes are communicated to the Fortran program by placing them in specially named sets.

The Fortran program will automatically detect the number of cutters according to contact body names. All rigid contact bodies that start with the keyword *cutter\_* are assumed to be cutters. This model generator script will name the contact body representing each cutter according to the following pattern, *cutter\_angle\_geometryType* where *angle* refers to the relative starting angle offset around the spindle (which is assumed to be unique), and *geometryType* is a type code, detected by post processing scripts to load the correct cutter profile to plot duty cycle data over. For example, a six sided cutter starting at 120 degrees around the spindle, would be named, *cutter\_120\_X6*.

Building on this pattern, the Fortran program will look for the following set names when collecting the local coordinate system direction nodes.

- Control (origin) node: *cutter\_angle\_geometryType\_origin*, (e.g. *cutter\_120\_X6\_origin*)
- Rake direction node: *cutter\_angle\_geometryType\_x*, (e.g. *cutter\_120\_X6\_x*)
- Feed direction node: *cutter\_angle\_geometryType\_y*, (e.g. *cutter\_120\_X6\_y*)

Like the spindle direction nodes, the vectors formed between the origin, feed direction and rake directions nodes do not need to be normalised during construction of the model, as this is done automatically by the Fortran program.

The cutter profile is extruded away from the spindle origin along the z axis, by a length set by the *width* parameter in the cutter configuration file definition, according to the following command and as shown in figure 8-22:

```
cutter = Expand(dz=width, *Filter(MarcCurve, cutter))
```

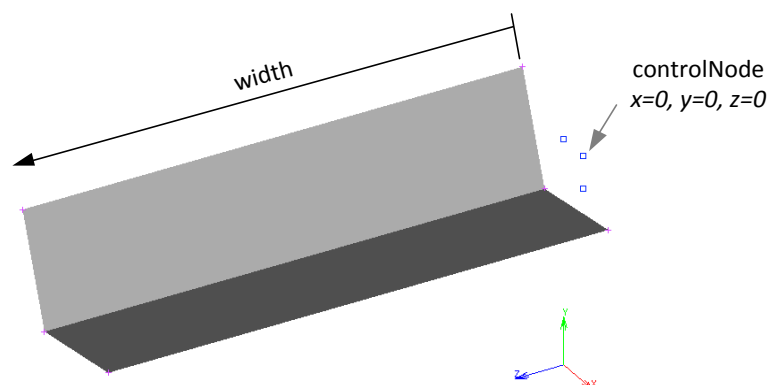


Figure 8-22 – Cutter extrusion

The extruded body surfaces are then moved such that the direction nodes are on the X-Y mid-plane of the cutter. If a non-zero *slideOffset* parameter is given in the cutter configuration, then this is also applied in the same step, according to the following command, yielding the configuration shown in figure 8-23:

```
Move(dZ=-width/2+slideOffset,*cutter)
```

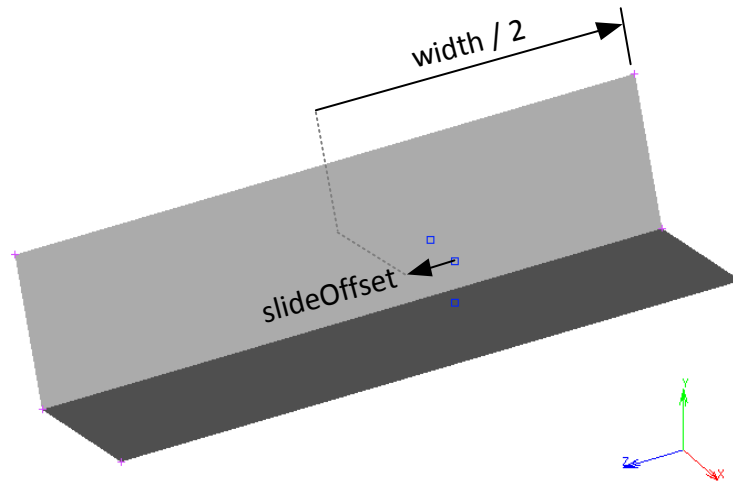


Figure 8-23 – Cutter radial offset

The next operation is to apply the inclination angle of each cutter, according to the following command, where figure 8-24 A) shows before and B) shows after inclination:

```
Rotate(tX=-inclination,*cutter)
```

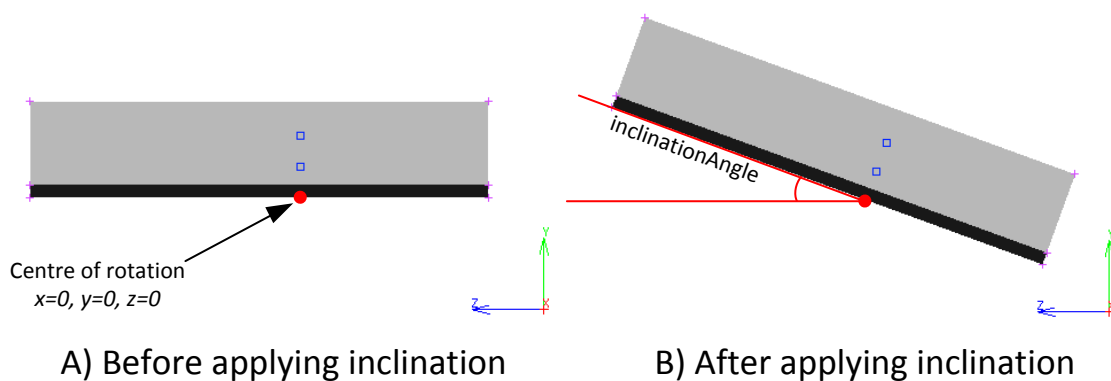


Figure 8-24 – Before, A) and after, B) inclination of cutter

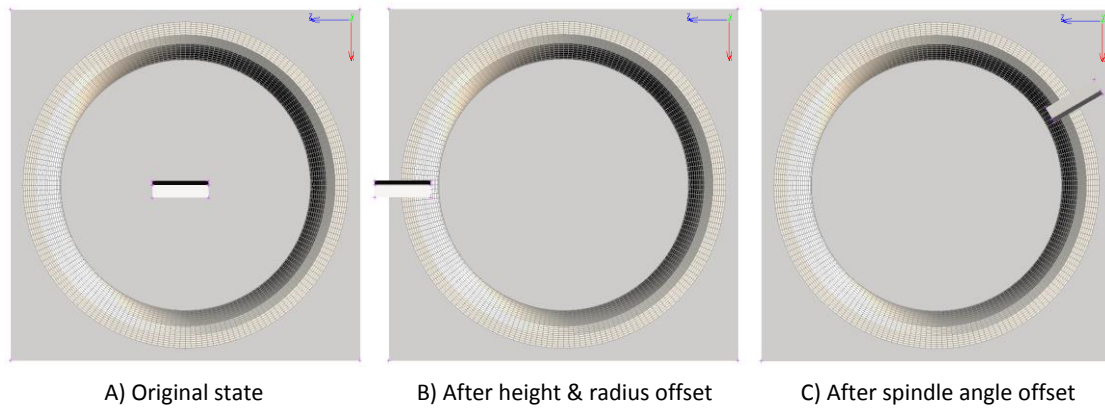
The radius and height offsets are applied in one step according to the following command:

```
Move(dY=height,dZ=radius,*cutter)
```

And finally, the spindle angle offset is applied according to the following command:

```
Rotate(tY=angle+90,*cutter)
```

These steps are shown in figure 8-25, A) before transformation, B) after height and radius offset and C) after spindle angle offset.



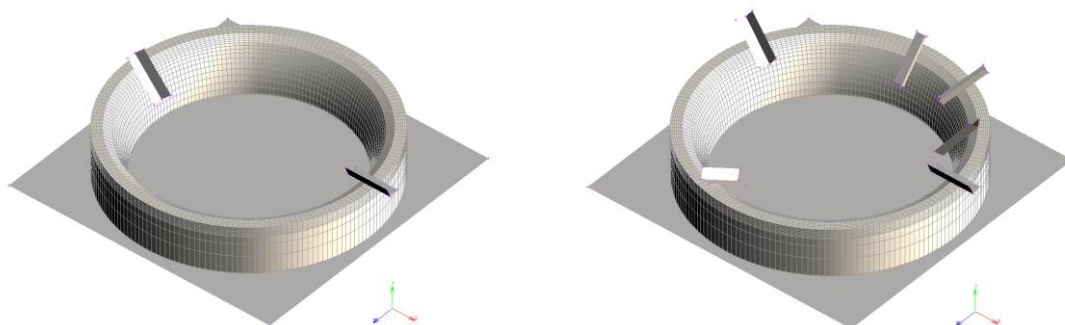
*Figure 8-25 – Height, radius and spindle angle transformations of cutter*

A velocity controlled rigid contact body is created to represent this cutter and named according to the pattern explained earlier in this section. No boundary conditions associated with cutters are input via the Marc model. Instead, the Fortran program discussed in Chapter Nine, automatically detects the cutters and applies the appropriate change in rotation and position using the MOTION user subroutine, according to the feed rate and RPM parameters set in the model configuration file.

This approach is used as it gives the Fortran program total control over the position of cutters. In simulations that only model a segment of the valve seat, the ability to have total control over cutter positions allows the implementation of a rapid mode setting which reduces the number of simulation increments spent on pure motion by temporarily increasing the feed rate and RPM until any cutter moves near the workpiece again.

Future implementations of the model may find controlling motion in this way is the most straightforward way of implementing the production line feed cycles, which are often complex and include rapid periods, withdrawal motion and, in some cases, even relative motion between cutting surfaces.

The entire process explained up to this point in this section is repeated for every cutter defined in the configuration file. By changing the configuration file cutter definitions, end users can quickly reconfigure the simulation environment to a new layout. There is no formal limit on the number of cutters allowed. Figure 8-26 shows some examples generated within seconds of one another.



*Figure 8-26 – Various cutter configurations*

### 8.6.5 Contact Table

Most contact is handled directly by the Fortran program developed in Chapter Nine. However, some basic contact constraints are added by the model generator script. These are glue contact between the workpiece and workpiece bulk, and the glue contact between the workpiece bulk and the foundation plane.

It was found during experimentation that the third obvious contact, between the workpiece itself and foundation plane, caused some issues with convergence in the solver. This was not investigated to conclusion as the two glue contacts described above are sufficient to constrain the mesh.

### 8.6.6 Remeshing Instruction

A global remeshing instruction is added for the workpiece geometry. This remeshing instruction serves two important purposes:

- Firstly, it converts the hexahedral input mesh into tetrahedral elements for the reasons discussed earlier in section 8.4.3. Put briefly, although hexahedral elements are easy to work with when generating the model geometry, they cannot be used with the Fortran mesh splitting algorithm developed in Chapter Nine. For this reason, the valve seat geometry must be remeshed using tetrahedral elements at the start of simulation.
- Secondly, it instructs Marc to call the UMAKNET user subroutine every increment, which is the entry point the Fortran program developed in Chapter Nine uses to generate a new mesh that represents the cut geometry.

The remeshing instruction is created by sending the evaluated Python strings shown in table 8-3 to Marc using the `py_send` command.

Command	Effect
"*new_adapg *adapg_type patran_tetra"	Creates the remeshing instruction.
"*adapg_name {}".format(ADAPT)	Names the instruction.
"*adapg_option immediate_crit:on"	Signals to execute immediately at the start of simulation (increment 0).
"*adapg_param increment_freq 1"	Signals to process the instruction every increment (required by UMAKNET).
"*adapg_param pat3_edge_len {}".format(configuration["targetElementEdgeLength"])	Sets the target element edge length from the user configuration file parameter.
"*adapg_option change_el_type:on"	Signals to immediately change the element type from hexahedral to tetrahedral.
"*adapg_rmsh_body {} {}".format(BODY_VALVE_SEAT)	Selects the contact body to be remeshed.

*Table 8-3 – Global remeshing commands*

Although the meshing instruction is global (as opposed to local), it should be noted that aside from the initial mesh, remeshing during simulation is performed on a strictly local basis by the code developed for UMAKNET in Chapter Nine.

### 8.6.7 Loadcase Configuration

All Marc simulations require a loadcase definition that defines the type and duration of simulation. The loadcase in this model is defined according to table 8-4.

Setting	Value	Comment
Loadcase type	Structural / static	
Time stepping mode	Fixed	Fixed, as opposed to variable depending on the convergence quality
Total loadcase time (s)	(calculated according to equation 8-1)	
Time step between increments (s inc <sup>-1</sup> )	(calculated according to equation 8-2)	

Table 8-4 – Loadcase settings

$$\text{Simulation time (s)} = \frac{\text{depthToRunTo (mm)} \times 60 (\text{s min}^{-1})}{\text{feedRate (mm rev}^{-1}) \times \text{toolRPM (revs min}^{-1})} \quad (8-1)$$

$$\text{Time Step (sec inc}^{-1}) = \frac{\text{degreesPerInc (degrees inc}^{-1}) \times 60 (\text{s min}^{-1})}{360 (\text{degrees rev}^{-1}) \times \text{toolRPM (revs min}^{-1})} \quad (8-2)$$

### 8.6.8 Job Configuration

The final step in producing the model is configuring the job definition. Job definitions in Marc define the solver type and system resources that should be used. The job definition also configures the desired results that should be calculated in post processing. Table 8-5 shows the job configuration used in this model.

Setting	Value	Comment
Nonlinear procedure	Large strain	
MOTION user subroutine	Enabled	The MOTION user subroutine is used to drive the cutters.
Solver	Pardiso Direct Sparse	
Number of cores	1	Must be run in single core mode due to an unsolved bug in the Fortran program described below.
User subroutine file	usub.f	Source code of the Fortran program

Table 8-5 – Job settings

It was found during testing that the Marc database would occasionally become unstable and nodal coordinates would no longer be recoverable, returning NaN (not-a-number) instead of a real value. This bug is almost certainly due to a multithreading problem with the Fortran program developed in Chapter Nine, or an incorrect or missing compiler flag. Unfortunately, there is very limited literature in this area of Marc user subroutine development so this problem remains unsolved. Fortunately, it was found that the speed difference is very minimal and in some cases the multithreaded variant was slower. Multi-core processing would be more beneficial if the mesh contained much larger quantities of elements.

The Fortran program described in Chapter Nine outputs a number of custom element scalars, nodal scalars and nodal vector post values. The model generator script developed in this chapter will locate the Fortran program source code and attempt to extract the definitions automatically. However, if the source code is not available, the model generator script will silently revert to an internal list. This facility saves time during development, as it is often the case during debugging, that a particular variable is useful to have plotted over the mesh. In such a situation, the model generator script does not need to be modified.

## 8.7 Chapter Summary

This chapter opened by taking stock of the experimental findings to this point, in that:

- the cutting inserts related to the case study in this work, did appear to be of good material consistency and were free from manufacturing defects; and
- although the processes used in the case study are vulnerable to resonant vibration at specific spindle speeds and over-flexibility of specific fixtures, these findings do not explain why other spindle speeds also develop the chipping problem and why other, significantly more rigid fixtures also suffer from the chipping problem.

The literature already suggests that cutting imbalance can create seemingly random chipping. With all other plausible explanations ruled out, this chapter set about designing a numerical model aimed at predicting these out of balance forces throughout machining.

A discussion was offered, reasoning that simulating all these effects, in 3D and with high enough accuracy to support many hundreds of simulation increments, would be nearly impossible with today's technology, especially if the simulation is to be used to examine many cutter configurations and would be expected to produce results in a very short time frame.

This chapter proposed using a cutting force prediction model developed in the previous chapter to take into account all these effects, without separately measuring them and creating individual material and system models for friction, plasticity, damage, etc.

This chapter then proposed a simulation model that would be split into two parts, a parametric script to assemble the inputs in a Marc simulation file and a Fortran program used to perform the advanced remeshing and force calculations required during simulation.

This chapter then set about presenting a design for the model, considering geometry, element performance, remeshing, materials, further work objectives and parametric compatibility. Following that design, this chapter then presented the technical implementation of the model generator script in the form of two Python scripts.

The scripts presented represent powerful modelling tools that can be used to generate a wide variety of cutter configurations for simulation. A lot of attention has been paid to preparing the scripts for further work expansion, such as incorporating the valve seat into a 3D cylinder head mesh.

The simulation procedure laid out in this chapter and illustrated in figure 8-2, consists of the following steps, the end user first:

- applies the methodology developed in Chapter 7 to a workpiece-tool material combination in order to determine specific force values for a given geometry, feed rate and width of cut, then
- programs tool and workpiece geometry and spindle speed into the parametric model generator developed in this chapter. The output of this generator is a finite element model complete with geometry, contact bodies, boundary conditions, materials and positional data for each cutter, then
- executes the simulation which invokes the Fortran Program developed in the next chapter which incrementally computes the cutting loads and machined geometry, and finally,
- compares the data from multiple simulations of different configurations to determine an ideal cutter configuration for minimising radial load.

# Chapter Nine – Design and Development of Fortran Program

---

## 9.1 Introduction

In section 3.6, MSC Marc was identified as a suitable finite element package for the cutting simulation developed in this work. However, since MSC Marc does not contain all the prerequisite functionality required e.g. splitting the workpiece mesh along a cutting plane and calculating cutting forces based on the cutting force prediction model developed earlier in Chapter Seven, this functionality must be added to Marc through user subroutines in the form of a Fortran program. The design and development of the Fortran program is the subject of this chapter. The Fortran program is the most substantial component of this research. The full source code is over 15,000 lines and represents thousands of hours of development.

To fully implement the numerical model requires tools to manipulate points and shapes in 3D space. Unlike many higher level languages, such as Matlab, that have pre-written tools to carry out low-level functionality e.g. vector based mathematics, Fortran provides no such features, and no such functionality is provided by Marc libraries (at least none is mentioned in the Marc user literature).

For this reason, achieving the higher level aims of this model, for example, slicing a group of elements along a plane, requires the development of many lower level functions, such as, calculating the intersection point between a ray and plane. The distinction between these two levels of programming is a convenient point to split this chapter. Section 9.8 deals with low level mathematical techniques and sections 9.7 and 9.9 deal with higher level functionality.

This chapter deals with the development, design and implementation of a Fortran program containing a series of user subroutines which are used by MSC Marc to calculate the cutting forces and simulate the removal of material in a highly optimised way.

This chapter will cover the following areas:

- introduction to the development environment, development tools and debugging practices;
- the broad design of the Fortran program, including a rationale for the how data in the subroutine is structured, accessed and stored, taking into account memory efficiency, computational speed, source code readability and ease of debugging;
- a board explanation showing how the Fortran program configures itself at the beginning of a simulation and a walk through of each step of the program during an increment;
- low-level functions – methods that are not specific to this problem, but are required by the higher level functionality of the program, e.g. ray tracing, vector transforms, volume calculations. These functions are typically characterised by their:
  - ability to perform a very limited number of very specific functions (simple well defined inputs and outputs);
  - design for efficiency and high call counts;
  - ease of transferability to other projects.
- High-level functions - methods that are directly responsible for delivering the stated aims of the program, including higher level concepts such as generating a volume mesh. These functions are typically characterised by their:



- long procedural flow (many steps, jumping between concepts and ‘building’ the end result from some starting point);
- design for high levels of input tolerance, with multiple possible paths depending on the state of the input;
- robustness against errors and recourse to contingency strategies; and
- highly bespoke nature - not readily transferable to other projects.

## 9.2 Definitions

This chapter frequently refers to certain programming, geometric and finite element concepts. This short section defines the terminology and introduces some basic concepts used in this chapter.

### 9.2.1 Programming Terms

**Program** – This term is used to refer to the complete set of code compiled for this work, including all private functions and subroutines, custom Marc user subroutines and everything else that is compiled to an executable and called upon by Marc.

**Subroutine** – A subroutine is a routine isolated from the scope of the calling routine that accepts zero or more arguments of any type and returns no values. Subroutines can modify the variables specified in the calling statement.

**Function** – A function is identical to a subroutine, with the one key difference being that a function can directly return a single value of any type.

**User Subroutines** – The term ‘user subroutine’, not to be confused with ‘subroutine’ has a very specific meaning in this work. A user subroutine is a subroutine defined in the source code that is called directly from Marc. The calling header of user subroutines is predefined and must match the Marc definition (even if variables in the calling header are not required). The body of user subroutines can be defined entirely by the programmer, but the headers and requirement to return certain variables cannot be changed.

User subroutines are set in various places throughout the Marc interface, for example, when adding a nodal boundary condition, the user has the option to activate the [FORCDT](#) user subroutine. When adding an element boundary condition, the user can activate [FORCEM](#). Or elsewhere in the program, for example when setting up remeshing, the user can defer to the [UMAKNET](#) user subroutine. Whenever a user subroutine is activated in this way, Marc will attempt to call it at runtime.

Although private functions and subroutines can be defined, Marc can only be configured to call subroutines with the predefined user subroutine headers listed in the Marc literature. Calling a custom subroutine or function can only be achieved via a user subroutine.

## 9.2.2 Procedure Diagrams

For procedure diagrams used throughout this chapter, the key defining their shape meanings is given in figure 9-1.

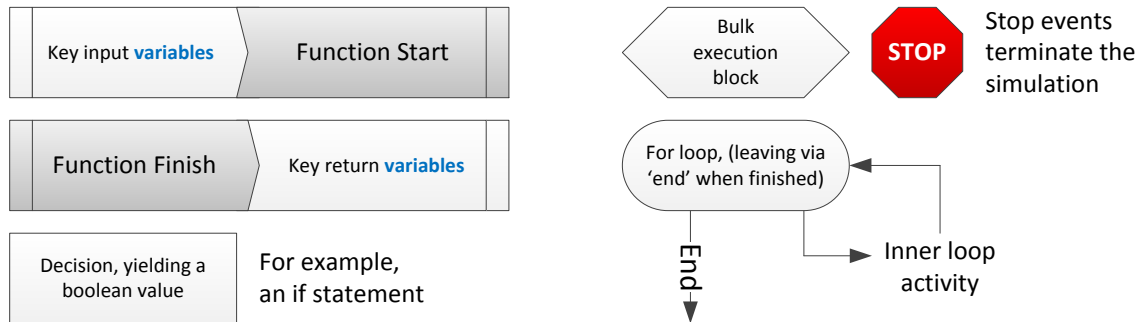


Figure 9-1 – Procedure diagram key

## 9.2.3 Pre-Processor Definitions

Before compiling, the source code is fed through a text pre-processor that can be used to modify the code and substitute values. Pre-processor definitions are used to include certain extra code blocks in debug mode, and to define mathematical constants (e.g.  $\pi$ ). Several other useful mesh constants are defined using these definitions which are explained as follows.

EPSILON,  $\varepsilon = 1E^{-11}$ . Unlike integers, floating point numbers will rarely equate neatly to one another. For example, the floating point arithmetic core of most processors will often find the assertion in equation 9-1 to be **False** (despite being **True** for all practicable purposes in reality).

$$\left| \sin \frac{\pi}{4} - \cos \frac{\pi}{4} \right| = 0 \quad (9-1)$$

This quirk of floating point arithmetic is due to small uncertainties that manifest as the least significant bits of the floating point number are culled to fit the data in the space allocated to the variable. For this reason, it is necessary to define an epsilon value. Values that differ by a magnitude less than epsilon are considered equal. For example, the assertion in equation 9-2 would evaluate to **True**.

$$\left| \sin \frac{\pi}{4} - \cos \frac{\pi}{4} \right| < \text{Epsilon} \quad (9-2)$$

SWEEP =  $5E^{-4}$  mm. The SWEEP parameter is a distance threshold, below which, two or more nodes are considered to be coincident. Geometric detail below this level is not required (and very often creates problems) and so where possible, these nodes are merged to simplify the mesh.

SWEEP\_NURBS =  $1E^{-8}$  mm. It was found during development that Non-Uniform Rational B-Spline (NURBS) based operations, such as parametric to real, and real to parametric routines cannot quite generate coordinates that consistently agree to less than EPSILON, however they were significantly more precise than SWEEP requires, hence the need for a special point equivalence threshold somewhere between SWEEP and EPSILON.

COARSE =  $8E^{-3}$  mm, This parameter is used very similarly to SWEEP, but typically to resolve issues of intersection and detection problems. COARSE is considered to be the maximum amount the program can

modify the mesh, to avert some disaster, before that mesh is considered to have been significantly modified. COARSE will commonly be used to perturb nodes in a mesh, where a glancing intersect makes it difficult to determine with certainty on which of two adjacent triangles a particular ray intersected.

#### 9.2.4 Mesh Definitions

The mesh topology used in this work is based on tetrahedral volumes. There are four types of geometric simplex referred to in this chapter, these simplices have an analogous concept when used in a finite element mesh indicated in brackets. The definitions of these types are as follows:

- Point (node) – a single coordinate in 3D space;
- Segment (edge) – all points along the shortest line that connects two points (nodes);
- Ray – a straight line that emits from some defined origin, that travels along some defined unit direction, for a given distance;
- Facet (face) – a plane bounded by three lines that form a closed loop; and
- Tetrahedral (tetrahedral element) – a four faced, four node, 3D geometric shape that represents some discrete volume.

Throughout this chapter, there are references to various types of mesh artefacts, including wedges, needles, caps and clusters. These features are defined here and shown in figures 9-2 and 9-3.

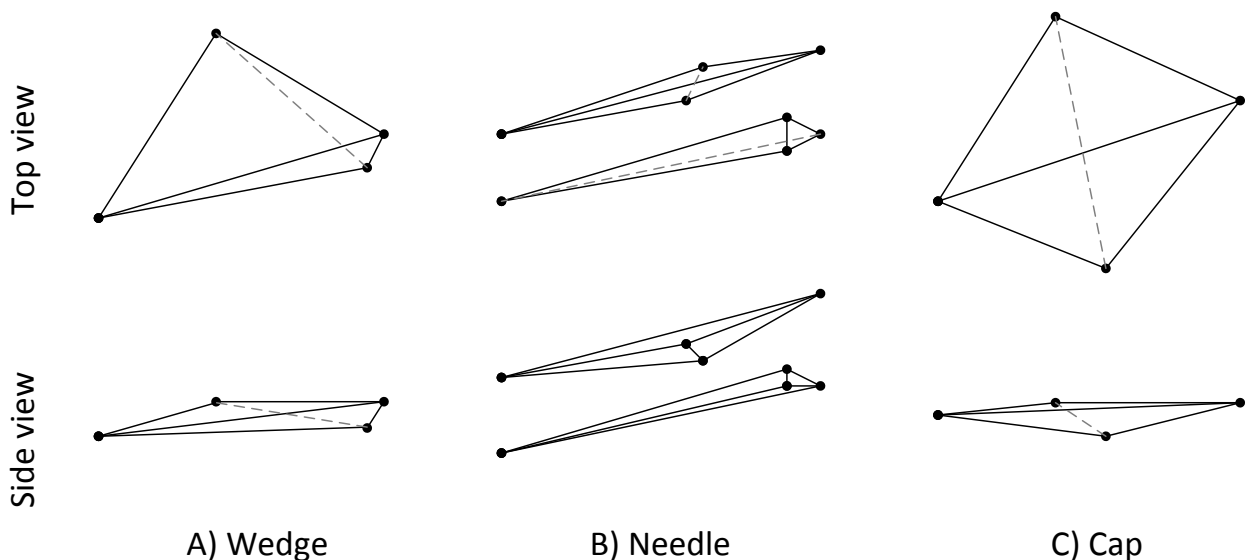
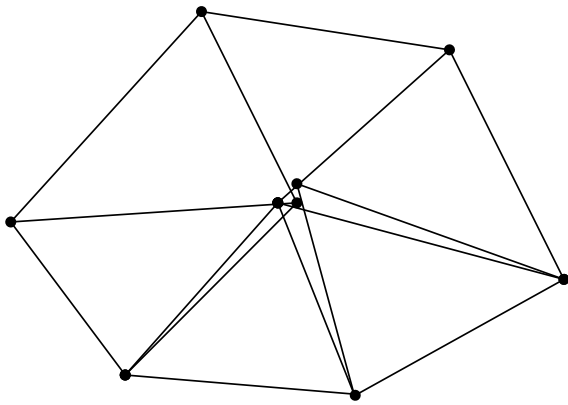


Figure 9-2 – Mesh artefact types

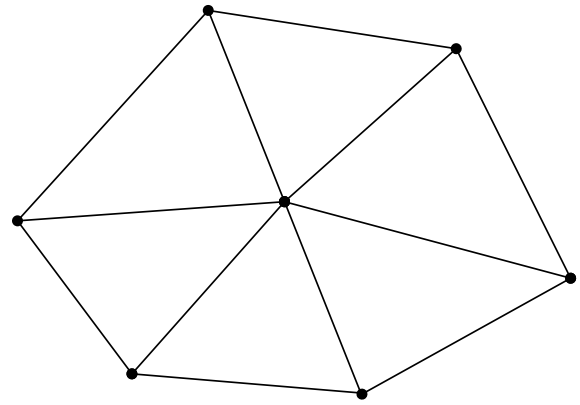
The common characteristic of these artefacts is that all nodes that make them up are approximately on the same plane (this is an important observation, core to eliminating them during mesh simplification).

When taken to their extremes, these artefacts represent poor quality elements not suited to finite element equations. These artefacts are particularly hazardous, as they can often be Delaunay and meet other criteria required to form naturally during volume meshing.

**Clusters** are small groups of nodes in close proximity. They can form as a result of floating point error during mesh splitting and if not resolved, they will typically lead to the formation of intersecting elements. Fortunately, clusters are easily dealt with by ‘sweeping’ – a process where nearby nodes are merged. The sweep algorithm is discussed later in section 9.7.6.



A) Cluster



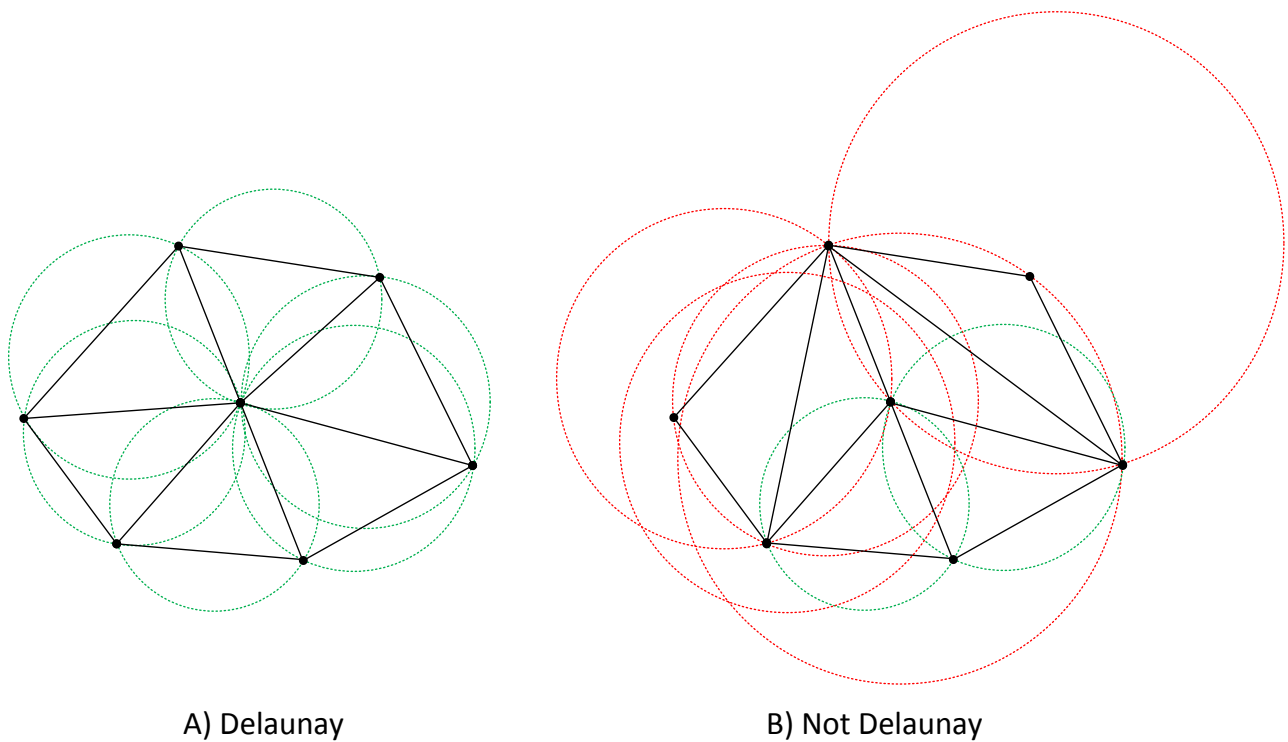
B) Cluster (after Sweeping)

*Figure 9-3 – Node cluster, A) before and B) after sweeping*

The **Delaunay** property of a mesh element applies in both 2D and 3D and refers to a core theory underpinning mesh generation and judging mesh quality.

An element is Delaunay if its circumcircle (2D) or circumsphere (3D) contains no nodes in its interior and only its own nodes on its radius (strongly Delaunay) or its own nodes and other nodes on its radius (weakly Delaunay).

Figure 9-4 illustrates the difference between a A) Delaunay and B) non-Delaunay mesh for the same set of nodes. Aside from simply being more aesthetically pleasing, Delaunay meshes carry essential properties. Firstly, they form elements with superior aspect ratios to randomised methods, which are much better suited for finite element simulations. Secondly, and importantly for this section of this work, the gift wrapping techniques used for meshing surfaces (described in detail in section 9.9.1) and volumes (section 9.9.2) have a strong affinity for forming Delaunay triangles and tetrahedrals (Shewchuk, 2002). That being the case, the gift wrapping techniques used will more likely create a mesh that agrees with edges and facets used to bound the mesh, if those edges and facets can be made Delaunay in advance. This is an important observation for improving the probability of a successful mesh generation.



*Figure 9-4 – Circumscribed Delaunay and non-Delaunay triangulations of the same set of nodes*

### 9.2.5 Siding

Both segments and facets can have a side. This is important when choosing which side of a segment or facet to generate a mesh element on. Since the concept of sided segments and facets is used in various places throughout the Fortran program, common definitions must be established.

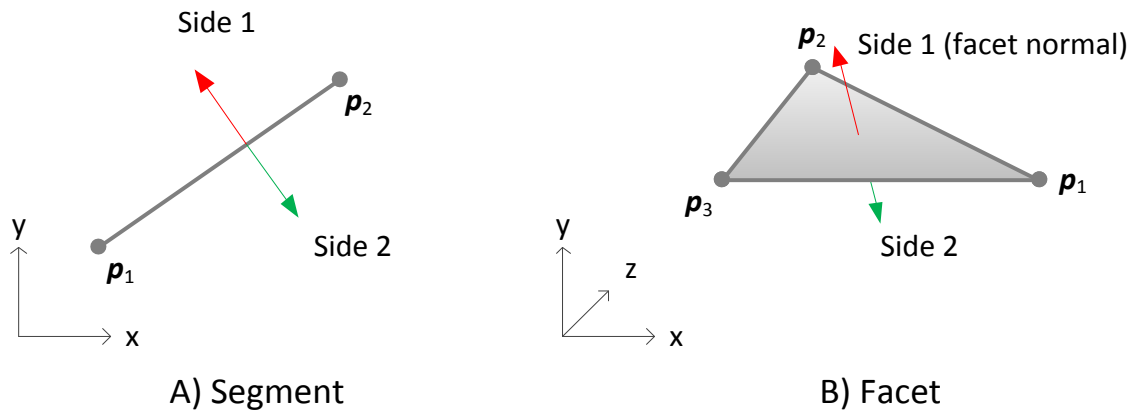


Figure 9-5 – Segment and facet side definition diagrams

For segments similar to that shown in figure 9-5, A), the vector that results from the cross product,  $\mathbf{n}$ , of the plane normal vector,  $\mathbf{N}$ , and segment vector,  $\mathbf{e}$ , defines side 1, as shown in equation 9-5, where  $\mathbf{e}$  is defined according to equation 9-3, where  $\mathbf{p}_1$  and  $\mathbf{p}_2$  are the segment points. Equation 9-4 shows the plane normal vector for the example given in figure 9-5, A).

$$\mathbf{e} = \mathbf{p}_2 - \mathbf{p}_1 \quad (9-3)$$

$$\mathbf{N} = \langle 0, 0, 1 \rangle \quad (9-4)$$

$$\mathbf{n} = \mathbf{N} \times \mathbf{e} \quad (9-5)$$

For facets similar to that shown in Figure 9-5, B), the facet normal vector,  $\mathbf{n}$ , is used to denote side 1, according to equation 9-8, where  $\mathbf{e}_{12}$  and  $\mathbf{e}_{13}$  are the edge vectors defined in equations 9-6 and 9-7, where  $\mathbf{p}_1$ ,  $\mathbf{p}_2$  and  $\mathbf{p}_3$  are the corner points shown in figure 9-5, B).

$$\mathbf{e}_{12} = \mathbf{p}_2 - \mathbf{p}_1 \quad (9-6)$$

$$\mathbf{e}_{13} = \mathbf{p}_3 - \mathbf{p}_1 \quad (9-7)$$

$$\mathbf{n} = \mathbf{e}_{12} \times \mathbf{e}_{13} \quad (9-8)$$

## 9.3 Development Environment

This section gives an introduction to the development software used, including its configuration and prerequisite packages for compiling and running the Fortran program. This section also discusses the relationship between Marc and the Fortran program, including how code in the Fortran program is invoked by Marc, and how information is retrieved from Marc's databases. Finally, this section shows some development and debugging methods used in this project.

### 9.3.1 Development Software

Marc is designed in such a way that user subroutines run at a low level, in line with the Marc solver itself. User subroutines in Marc are compiled from Fortran source code. This approach provides a much greater potential for speed, at the cost of longer development times.

The following software is used to set up the development environment and test the user subroutines. For end users, only Marc is required and the compiled program can be supplied as a standalone executable package:

- Microsoft Windows 7 (x64)
- MSC Marc 2016
- Visual Studio Professional 2012 (or above)
- Intel Fortran Compiler (Parallel Studio XE 2015 – Update 6) (ifort)

Marc acquires the path to the Intel Fortran compiler and Visual Studio linker from the path environmental variable, therefore the following path variables must be present in Windows.

```
C:\Program Files (x86)\Intel\Composer XE 2015\bin\intel64\ifort.exe  
C:\Program Files (x86)\Microsoft Visual Studio 11.0\VC\bin\link.exe
```

Furthermore, the following common library folders must be declared in the LIB environmental variable for use by the compiler.

```
C:\Program Files (x86)\Microsoft Visual Studio 11.0\VC\lib  
C:\Program Files\Microsoft SDKs\Windows\v7.1\Lib\x64  
C:\Program Files (x86)\Intel\Composer XE 2015\compiler\lib\intel64
```

In order to prepare the environment for compiling the Fortran user subroutine, the Marc \*.mud is opened from a Windows batch file which first executes *vcvarsall.bat*, located in the Visual Studio installation folder. The most important function of *vcvarsall.bat* is to declare the library path environmental variables to ensure the compiler links libraries for the correct processor architecture.

```
call "C:\Program Files (x86)\Microsoft Visual Studio 11.0\VC\vcvarsall.bat" x64
```

### 9.3.2 The Relationship Between the Fortran Program and Marc

The user subroutines developed for this project are called by Marc. They do not have control of the root process and must yield the execution pointer back to Marc at the end of every user subroutine. The Fortran program cannot interrupt Marc, nor can it execute code in parallel with Marc's internal execution. In practice, this execution pattern is not a major hindrance. User subroutines cannot guarantee the persistence of data between calls, but with a number of programming techniques, data can be persistently stored across increments by defining a number of variables inside a Fortran module and including that module in user subroutines using Fortran's **use** statement.

Marc offers a wide range of user subroutines, the subroutines used in this work are given in table 9-1.

User Subroutine	Purpose
<b>FORCEM</b>	Applies a reaction force to the workpiece.
<b>MOTION</b>	Sets the rotational and feed velocity of the cutters.
<b>UMAKNET</b>	Outputs a new mesh every increment to replace the workpiece mesh.
<b>PLOTV</b>	Sets element scalars for viewing in the post file.
<b>UPSTNO</b>	Sets nodal scalars and nodal vectors for viewing in the post file.
<b>UBGINC</b>	Called at the beginning of each increment, used to call <b>Setup</b> on increment 0 and test the output of UMAKNET every increment thereafter.
<b>UEDINC</b>	Called at the end of each increment, used to reset the statistics module (not discussed in this chapter).

*Table 9-1 – User subroutine headers used in this program*

The headers for these subroutines are fixed, the definitions for each can be found in the Marc user subroutine programming guide (MSC Software, 2016b) and at the end of Appendix B.

During simulation, Marc must keep track of the simulation state. Marc does this by storing data in a vast array of variables that correspond to things such as simulation time, node and element counts, nodal loads, material data, NURBS surface information, element connectivity data and so on.

A surface level query of Marc's internal databases can be made using some predefined functions, such as **e1nodes** (to get a list of nodes associated with an element) or **getbodyid** (to retrieve the contact body ID of an element). For more detailed information, Marc common blocks must be included in the Fortran program, using the Fortran **include** keyword. Marc's predefined functions and some common block variable definitions are defined in Marc's programming guide (MSC Software, 2016b), however the vast majority have no publicly available documentation. Many of these undocumented common blocks have been used in this work, with their meanings interpreted by inspecting the common block headers, memory analysis and trial and error.



### 9.3.3 Debugging Tools and Methods

Microsoft Visual Studio is a very well developed programming environment. Whilst it would be completely possible to develop the entire program from a simple text editor such as Notepad++, thanks to Marc's ability to invoke the Fortran Compiler directly and compile the source at runtime, it is infinitely more useful to use Visual Studio thanks to its comprehensive debugging capabilities. Breakpoints are special instructions that Visual Studio can seamlessly embed in the compiled machine code at runtime, that when hit by the processor execution pointer, 'raise' that breakpoint. This causes the program to temporarily pause. During this pause, Visual Studio inspects the program memory and populates variable names with their values. For example, figure 9-6 below shows how the *origin* property of object **splitRay** can be inspected in the moment before the program raises the highlighted exception. Equally as useful, the execution pointer can be moved to a different section of code and the stack trace can be inspected and traversed.

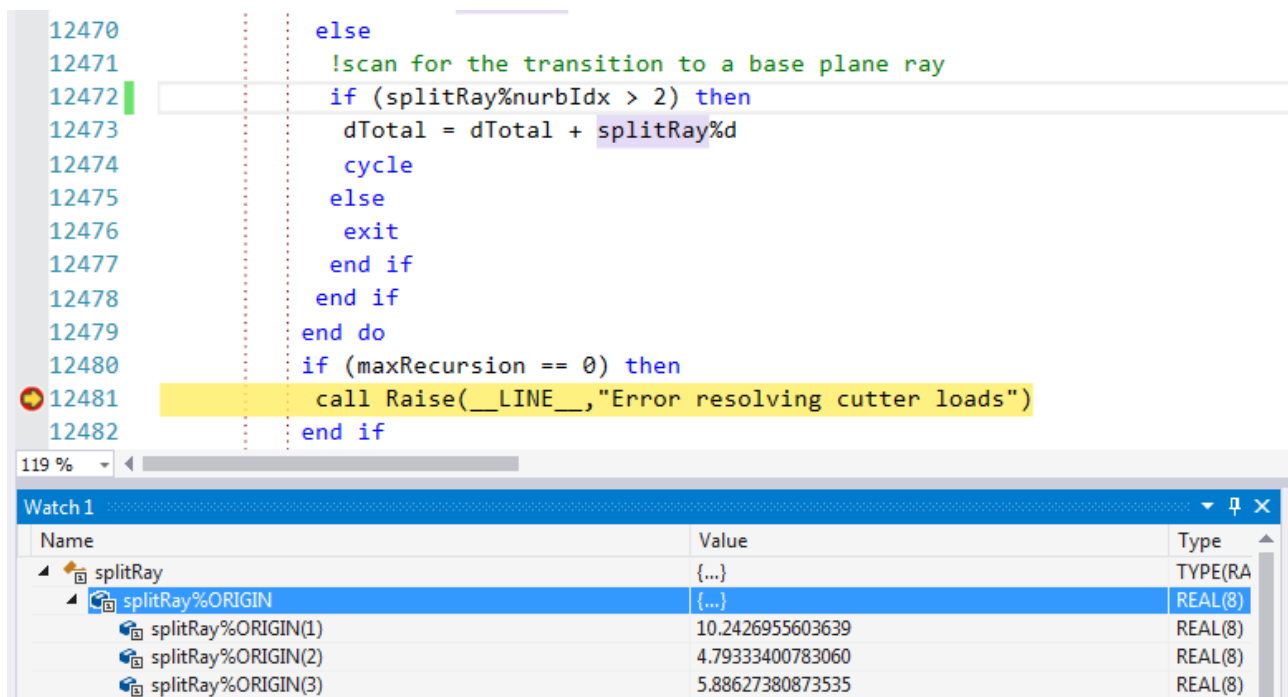


Figure 9-6 – Breakpoint example in Microsoft Visual Studio

The following ifort compiler switches are required to activate the inclusion of debugging symbols in the compiled program:

```
/traceback /debug:full
```

Although Visual Studio has no problem working with executables developed in Fortran, compiled with Intel's Fortran compiler, it was never designed to work with Marc. To make this functionality work, a series of bespoke tools were written which help to interface Visual Studio with Marc and seamlessly run the simulation when selecting 'Build and Run' in Visual Studio.

At the time of writing, there appears to be no evidence in literature that anyone else has configured Marc to work with Visual Studio in this way.

Aside from the debugging symbols previously introduced, it is essential to have some facility to log data to a file and monitor the data being written to the file in real time. It is often the case that by the time the

program hits an exception breakpoint, it is already too late to inspect the state of the program that led to that condition. For this reason, key variables can be persistently written to a log file as and when their state changes. For example, the local coordinate system of a particular cutter or the progress of mesh generation.

At runtime, Marc and the solver each create a log file named after the simulation and end in the extensions, \*.log and \*.out respectively. A script (named 'tail' after a similar utility built into Linux) was written in Python to monitor these files in real time. The output file can be written to by the Fortran program using unit file number 6. A typical output from this log is shown in figure 9-7.

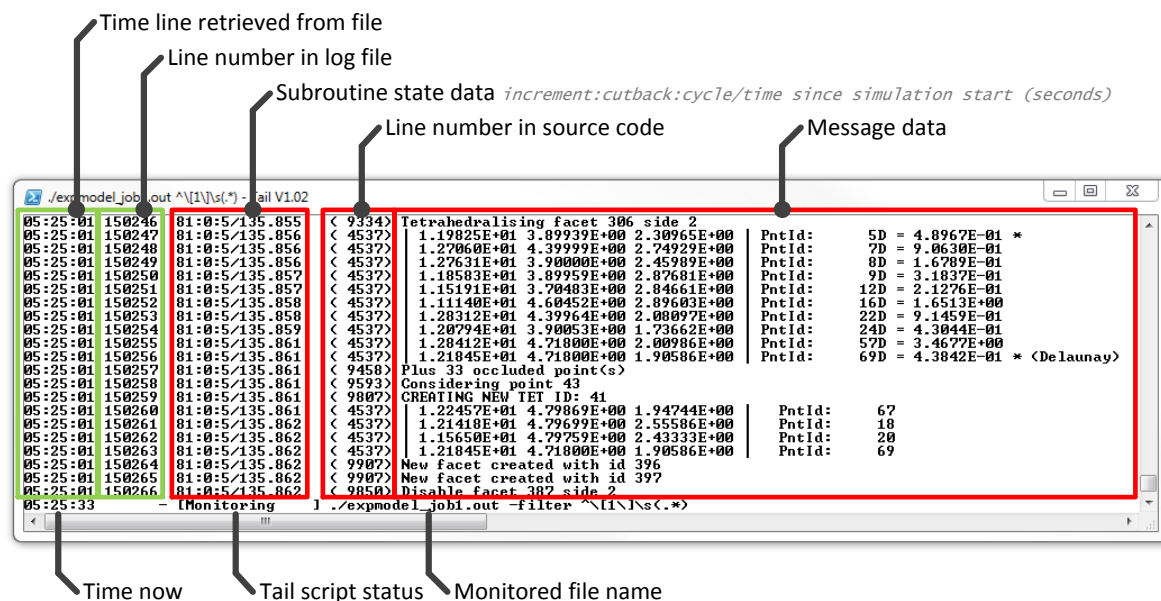


Figure 9-7 – Real-time log viewer support program

In this output, only messages generated by the Fortran program are shown. Lines added by Marc are filtered out automatically for clarity.

The first column shows the time the line was retrieved from the output file. It is not necessarily the time the data was written to the file by Marc, since the operating system will often delay file writes to better suit the mechanical demands of the hard disk. The second column shows the line number in the output file. These columns, shown in green, are generated by the tail script itself. The third column is the time in seconds since execution started in the Fortran program. The fourth column is the line number in the source code that generated the output and the remainder of the line is the message. These components in red are generated by the Fortran program.

In addition to these log tracking features, a number of functions have been written within the Fortran program itself to aid with debugging and performance tuning. Figure 9-8 below, shows an example of a summary table which is generated at the end of every increment. The summary table contains important information regarding the time taken to execute each section of code. The data is presented in a tree format so that key information can be inferred at a glance (this is important when scrolling through many increments worth of summary data).

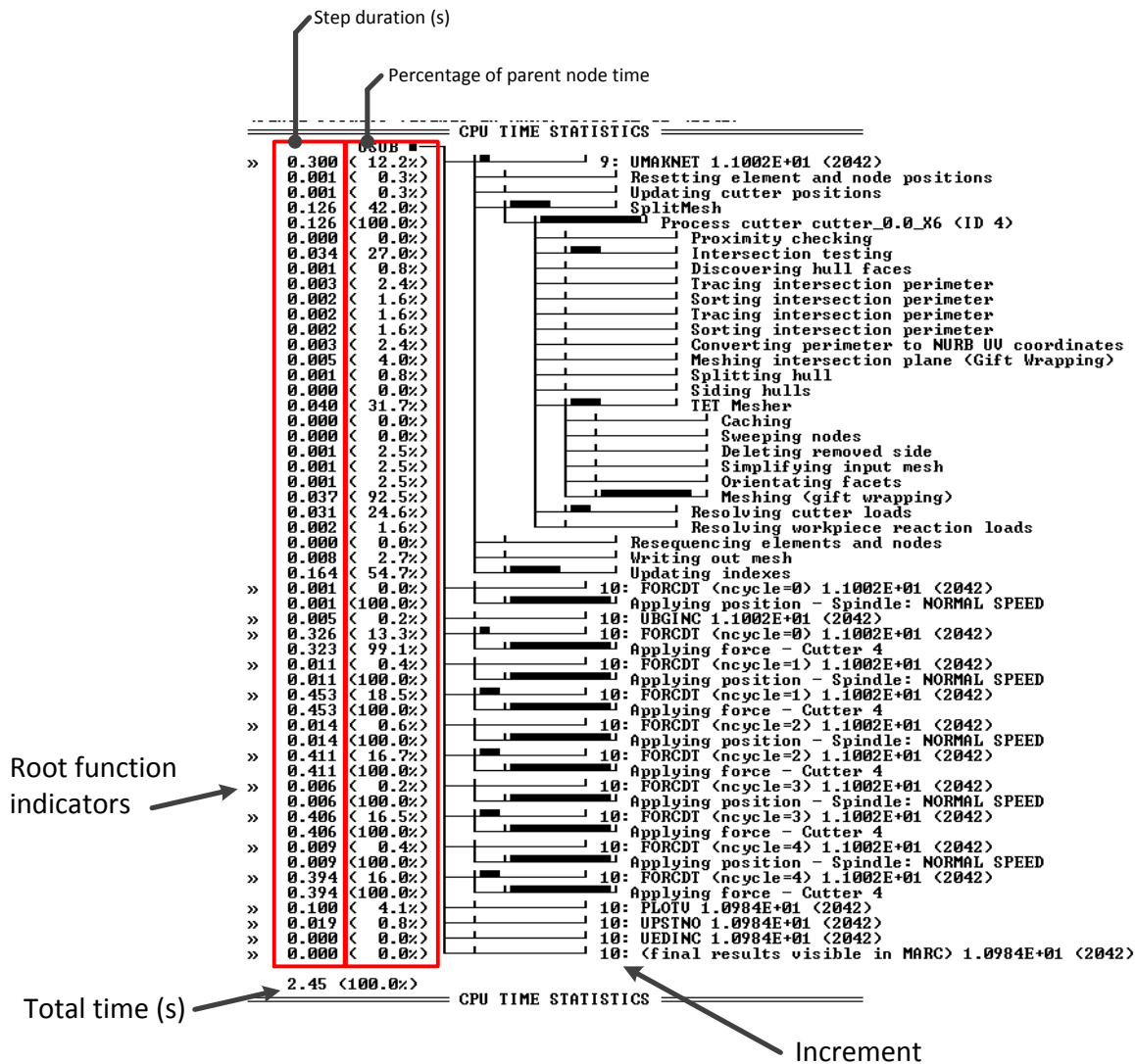


Figure 9-8 – Hierarchical CPU time tree view

## 9.4 Program Theory

This section deals with the broad theory underpinning the program, including how it is laid out in source code, how it initialises at runtime, and the procedural steps it follows at runtime. This section has been distilled down to a high level explanation with definitions and mathematical concepts dealt with separately in sections 9.2 and 9.8 respectively.

The primary functions of the program are to:

- detect cutting events that occur between cutters and the workpiece, by computing the proximity between cutter and workpiece and using an intersection algorithm (developed in 9.7.1) to compute intersections;
- generate new workpiece geometry that approximates what the true cut geometry would look like based on the principle that once a cutter has moved through space occupied by the workpiece, any workpiece material in that region is subsequently removed; and
- calculate the forces that develop during the cutting event for the cutter, spindle and workpiece based on the cutting force prediction model developed in Chapter Seven.

It must achieve these goals using the following inputs:

- the mesh geometry of the workpiece;
- the geometry, position and velocity of the cutting edges; and
- two specific force equations that output the rake and feed cutting forces based on a given depth and width of cut (according to the cutting force prediction model developed in Chapter Seven).

The thesis of this project is that cutting simulations such as these can be run in 3D in vastly reduced times over that of traditional cutting simulations. In order to assist with this aim, it is necessary to reduce computational effort wherever possible. This is especially important in parts of the code that are executed many thousands, in some cases hundreds of thousands of times per increment. Several efficiency strategies have been applied to improve execution times, such as:

- in loops that are likely to end early due to disqualifying criteria, expensive tasks are delayed until absolutely necessary to avoid wasted computational effort;
- simplification of equations, for example:
  - pre-calculating all parts of an equation that can be calculated before compiling;
  - layout equations to reduce the number of divisions and square root operations;
  - normalise direction vectors when setting, rather than every time the direction is used;
  - avoid rigidly following an equation if just the relationship will do (e.g. no need to compute the radius from circle diameters, if the radii will only be used to compare sizes); and
- the use of caches, memory maps and lookup-tables to avoid repeated calculations and database queries.

### 9.4.1 Types and Objects

Fortran is not traditionally an object orientated programming (OOP) language, however Fortran has long supported user defined types, and since Fortran 2003, type-bound procedures are also supported. These features of Fortran types can be used to implement many OOP techniques which suit the objectives of this project well.

User defined types are primarily used in this work to improve readability, reduce repeated code and establish rigid specifications for exchanging data between functions. They are not used for speed or memory efficiency (and often decrease the speed and memory efficiency potential).

Wherever an object is instantiated, Fortran always allocates enough memory to contain the entire object (the combined memory required to contain all of its properties). This often leads to cases where an object supports multiple contexts but, whilst being used in one context, variables defined for another context are not being used, thus reducing memory efficiency.

Steps must be taken to reduce the memory footprint of objects, especially when those objects are being used in lists of many hundreds or thousands of instances.

An unfortunate feature of the way ifort allocates memory for logical (Boolean) values means that 64 bits are reserved by the compiler to store a single logical value<sup>1</sup>. In the majority of cases, this isn't a problem, however, when the logical value forms part of a frequently instantiated type, this can amount to considerable memory waste. To overcome this waste, logical values within the **Node**, **Element** and **Ray** types are encoded in a single integer value, `status`, using bitwise Boolean logic. This allows the storage of 64 Boolean values in the memory space of a single integer.

A brief summary of the user defined types developed for the Fortran program is given as follows. For more detail about the properties and type bound procedures associated with these types, please see Appendix B.

- **Element**: Generated for each element entity in Marc and supports elements with up to 8 nodes. This support is required as the workpiece bulk mesh and initial workpiece mesh is hexahedral before Marc generates the initial tetrahedral mesh.
- **Node**: Generated for each node entity in the simulation. **Node** objects are key to maintaining the 'elements at node' cache commonly used throughout this work. This functionality is implemented using an allocatable array, `elementsInt`, within the **Node** object itself. The array is allocatable to conserve space at the cost of longer initialisation times (due to reallocations). However, the time saved throughout the program by using the 'elements at node' cache is worth this small upfront cost.
- **Ray**: A fundamental and very frequently used type to represent geometric rays, lines and segments. **Ray** objects, as described in section 9.8.2, are defined by an origin, unit direction vector and distance magnitude.
- **NURBSSurface**: NURBS surface entities are used in several contexts throughout the program, including contact detection, mesh splitting, 2D meshing and force distribution. It is useful therefore, to have a common structure to package the numerous properties associated with NURBS surface entities. For an explanation of how NURBS surface coordinates are calculated, see section 9.8.3.
- **Cutter**: Created for each cutter entity in the model, like **NURBSSurface** objects, **Cutter** objects are used in various places throughout the code, often as the subject of a loop. It is therefore helpful to have access to properties associated with **Cutter** objects in a common structure.

The Fortran program maintains two important indexes in the global scope. These are `nodeIndex` and `elementIndex`. As the names imply, these are lists consisting of **Node** type objects and **Element** type objects respectively. At the start of the simulation, these indexes are populated with objects to represent all

---

<sup>1</sup> This behaviour is flexible, the length of a logical value can easily be reduced to 8-bit, however this creates compatibility problems with built-in functions that accept logical inputs. The alternative used in this project is significantly more memory efficient.

elements and nodes in the mesh. Both lists are largely redundant, since Marc's database can be queried directly in most subroutines. However, there are many properties of node and element entities that are required by this program, for example, nodes associated with elements, elements associated with nodes, the position of a node, the volume of an element, the contact body ID of an element and so on. The method used to recover each piece of data from Marc's internal database is surprisingly variable, obfuscated and in some cases, slow. For this reason alone, the bulk collection and storage of all properties in one batch function, justifies the memory redundancy of the `elementIndex` and `nodeIndex`.

Furthermore, the indexes allow the storage of custom data against a particular **Node** or **Element**. For example marking a **Node** as near a particular **Cutter**, or storing the cutting pressure applied to a particular **Element**.

Aside from the memory inefficiency of keeping these indexes, there is also the burden of keeping them updated. The offset of an **Element** or **Node** object within the index is also its Marc internal ID. For example, the **Element** found at offset 32 in `elementIndex`, represents the element entity with Marc ID 32. This is an important feature of the lists which is used extensively throughout the program. In order to preserve this property across increments, after adding and removing node and element entities during meshing, code is required to anticipate where Marc will reorganise node and element entities within its own internal lists. To understand Marc's indexes, the following rules are established:

- **UMAKNET** (as an agent of this program) can only add and remove **workpiece** element and node entities;
- Element entities can never be shared between two or more contact bodies, however, node entities can; and
- Marc will not shrink its internal element or node entity lists, (the lengths of which can be determined by the Marc global variables, `numel` and `numnp`, respectively).

With these constraints in mind, Marc will organise its internal element list according to the following rules:

An element entities place in the list should be thought of as occupying a 'slot'.

Element entities:

- Non-workpiece elements are never moved to another slot;
- If an element is killed, its slot is marked as vacant;
- After the removal of elements, existing elements are redistributed to fill vacant slots; and
- New elements are added sequentially into vacant slots after redistribution of existing elements, if no vacant slots remain, the list will expand to accommodate new elements.

Node entities:

- Non-workpiece nodes, or nodes that are shared between contact bodies are never moved to another slot;
- If a node was only ever associated with a workpiece element, but now no longer has any references to it, its slot will be marked as vacant;
- After the removal of nodes, remaining workpiece-only nodes are redistributed to fill vacant slots;
- New nodes are added sequentially into vacant slots after redistribution of existing nodes, if no vacant slots remain, the list will expand to accommodate new nodes.

At the end of **UMAKNET**, the Fortran program will reorganise its own `nodeIndex` and `elementIndex` lists according to these rules, with the intention that at the start of the next increment, Marc will have done the same and all node entities will be represented by a **Node** object in `nodeIndex` and likewise, element entities will be represented by an **Element** object in `elementIndex`. The object offsets in their respective indexes will match the Marc ID of their corresponding entities.

#### 9.4.2 User Subroutine Call Order

This section is intended to show the calling order of user subroutines within this program. User subroutines are called by Marc sequentially at the appropriate points during simulation. As described in the programming definitions section, 9.2.1 earlier, the headers and calling order of user subroutines is a hard coded feature of Marc. However, the programmer is free to add additional functions and subroutines, providing they are called from within one of Marc's user subroutines. Marc cannot be made to call a custom function or subroutine without first going through a user subroutine.

Increments of this program are not neatly aligned with the increment progression of the simulation. To help describe this misalignment, the concept of a session is introduced. Sessions can be thought of as a period in time in which some bulk of data is delivered, worked on and then returned. Sessions end with the discarding of almost all associated data except for the return values and certain pieces of meta-data (such as the time required to complete the session). Figure 9-9 shows the boundary lines of sessions from the point of view of Marc (red) and the Fortran program (blue).

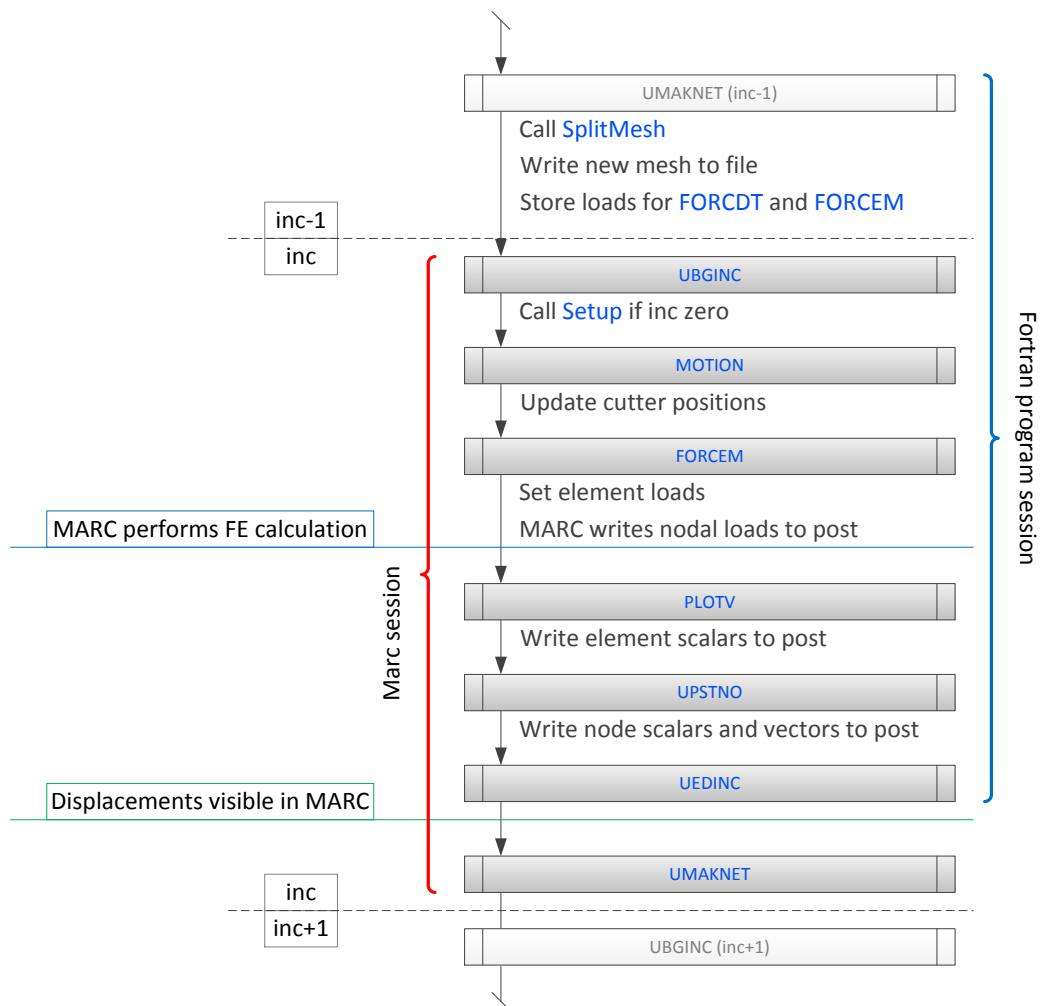


Figure 9-9 – Session overlap and subroutine call order

Marc calls the functions shown sequentially until the simulation terminates. **SplitMesh**, the core function within this project is called from within **UMAKNET**. **SplitMesh** is responsible for calculating three key outputs, the new (cut) mesh geometry, the cutter and workpiece loads and various nodal scalar and vector values. After **SplitMesh** returns, **UMAKNET** immediately writes out the new workpiece mesh to a file and stores its other outputs, such as the calculated cutter reaction load and workpiece load, in memory (this remains accessible until the session ends). At some point before Marc calls **FORCEM**, it reads in the new mesh and replaces the workpiece contact body mesh. Portions of the data calculated in **SplitMesh** are referenced by Marc as Marc steps through the subroutines.

### 9.4.3 Configuration File

The Fortran program uses a configuration file to allow the user to tweak simulation parameters, turn features on or off and specify the cutting force equations. Table 9-2 lists the available configuration file parameters and their meanings.

Parameter Name	Description	Units	Required
<i>ProximityRadius</i>	The search cylinder radius centred around the cutter proximity <b>Ray</b> used to determine which elements are near.	mm	Yes
<i>OutputPhysicsData</i>	Switch to enable or disable outputting of incremental physics data for each contact body. Physics data includes information such as the contact body forces, spindle angle, contact status etc. When enabled, physics data for each contact body will be output to a Comma Separated Values (CSV) file in the model directory.	(Boolean)	No (default = Yes)
<i>OutputCutterFaceProfile</i>	Switch to enable or disable outputting of incremental cutter face profile information for each cutter. This information is used by a Python script to plot the contact ‘heat-map’ over a diagram of the cutter face.	(Boolean)	No (default = Yes)
<i>NoReactionForce</i>	Switch to enable or disable the application of reaction force to the Workpiece. If workpiece reaction force is not required, enabling this option can speed up the simulation slightly.	(Boolean)	No (default = False)
<i>Feed rate</i>	The feed rate of the cutters.	mm / rev	Yes
<i>RPM</i>	The spindle RPM.	RPM	Yes
<i>RapidMultiplier</i>	How many times faster than normal speed the cutters will move if not near the workpiece.	(Multiplier)	Yes (set to 1 to disable rapid)
<i>FlexibleSpindle</i>	Legacy option to enable flexible spindle mode <sup>2</sup> .	(Boolean)	Not (default = False)
$F_r$	The rake force equation (see Chapter Seven).		Yes
$F_f$	The feed force equation (see Chapter Seven).		Yes

Table 9-2 – Configuration file parameters

<sup>2</sup> The Fortran program has full support for a flexible spindle mode. However this mode is not discussed in this chapter since experimental data for the spindle flexibility could not be acquired.



## 9.5 Procedure

This section gives a high level overview of the main subroutines discussed in this chapter, they are; the built-in user subroutine headers listed earlier in table 9-1, as well as [Setup](#) and [SplitMesh](#). The call order relationship between these subroutines during simulation is given in the master procedure diagram in figure 9-10.

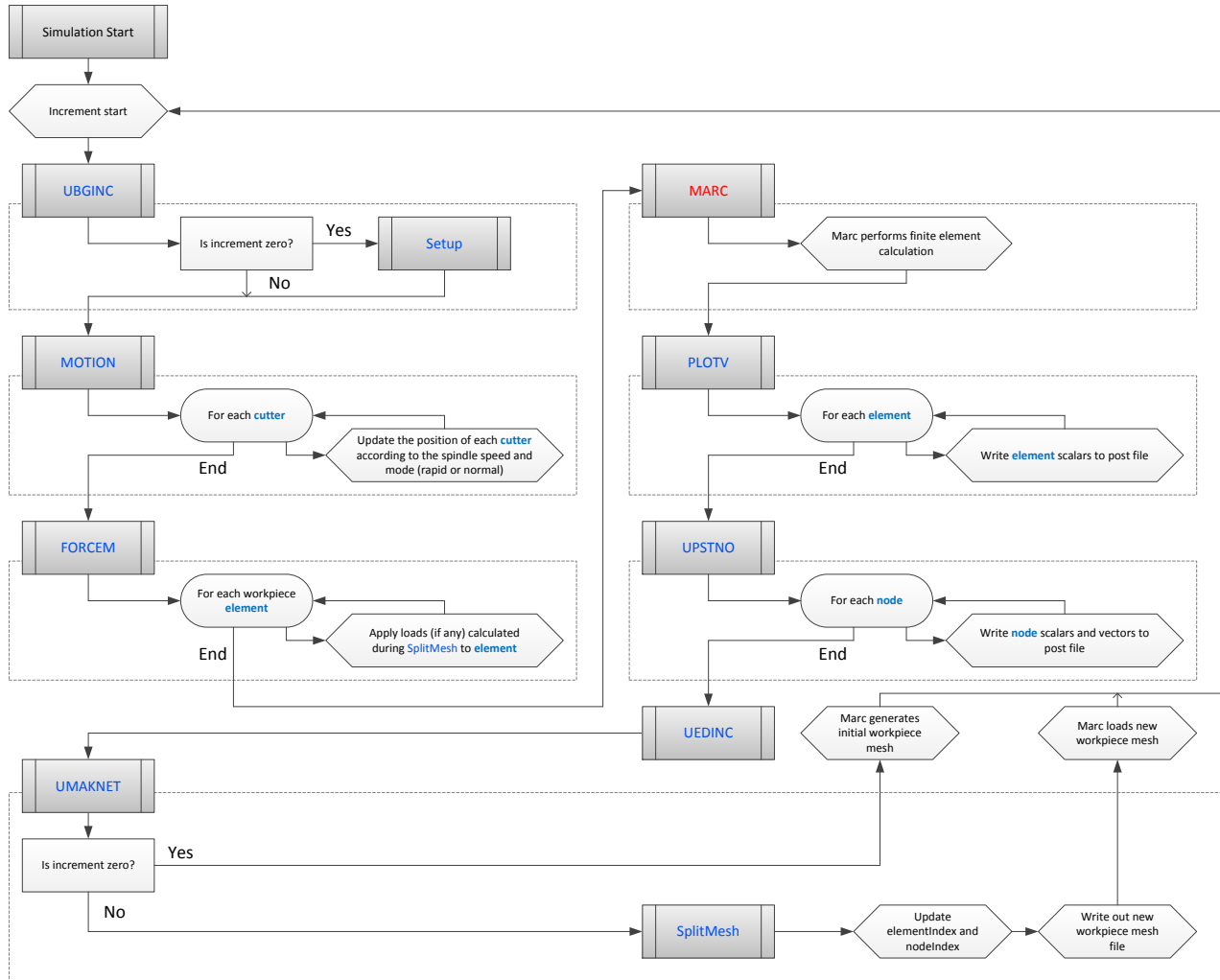


Figure 9-10 – Master procedure diagram

The basic function of each subroutine referenced in figure 9-10 is given as follows:

- **UBGINC**: Called at the beginning of each increment. On the first increment, this subroutine calls [Setup](#). On all other increments, if the `DEBUGMODE` pre-processor definition was set when compiling, this subroutine compares `nodeIndex` and `elementIndex` to Marc's internal database to make sure **Node** and **Element** objects are configured correctly.
- **MOTION**: Called by Marc for every contact body in the model when processing geometry updates. If called for a non-cutter contact body, the Fortran program dismisses the call. If called for a cutter contact body, the Fortran program returns the spindle speed and feed rate (or rapid spindle speed and feed rate) specified in the configuration file.

Due to the exceptionally high call counts of **FORCEM**, **PLOTV** and **UPSTNO**, great care is taken to reduce their computational demand. Therefore, unlike most other functions, these functions contain no statistics tracking code.

- **FORCEM**: Used to set element reaction pressures calculated at the end of the previous increment by **SplitMesh**. **FORCEM** is called for every integration point, in every element that was assigned the **FORCEM** volume pressure boundary condition.
- **PLOTV**: Used to assign element scalar post variables. **PLOTV** is called for every element post scalar selected in the simulation job, for every element in the model.
- **UPSTNO**: Used to assign node scalar and node vector post variables. **UPSTNO** is called for every node scalar and vector selected in the simulation job, for every node in the model.
- **UMAKNET**, is called once at the start of the Fortran program session, or at the end of each increment (with the exception of increment 1, where Marc will call it at the start, or increment 0). For increment 0, **UMAKNET** instructs Marc to use its own advancing-front tetrahedral mesher to generate a mesh for the entire workpiece, after which **UMAKNET** returns. For all other increments, **UMAKNET** performs the following functions:
  - resets all **Element** and **Node** states;
  - calls **SplitMesh**;
  - writes out the new mesh generated by **SplitMesh** (Marc requires the new mesh to be written to a file so it can import it from that file);
  - anticipates how the removal and addition of elements and nodes will affect Marc's internal indexes and updates `elementIndex` and `nodeIndex` to match Marc's internal indexes; and
  - updates the positions of cutters ready for the next increment.
- **SplitMesh**, performs the following tasks for each **Cutter** in the model:
  - identifies **Element** objects that must be split based on their proximity to a **Cutter**, and the intersection of their edges with a **Cutter**;
  - generates an external hull of facets from the near by / intersected **Elements**;
  - generates an intersection loop of edges and uses it to:
    - seed an interface mesh for the surface occupied by both the **Cutter** and workpiece, generated by calling **Giftwrap**; and
    - split external facets, again, by calling **Giftwrap**.
  - identifies the side of the split hull to keep by calling **Facetwalker**;
  - simplifies the remaining hull mesh;
  - calls **TetMesh** to generate a 3D mesh within the interior of the hull;
  - appends the new **Element** and **Node** objects generated by **TetMesh** to the end of `elementIndex` and `nodeIndex` respectively; and
  - calculates the cutting loads and stores them for later assignment by **FORCEM**.

## 9.6 Runtime Setup

This section describes the setup phase of the Fortran program. These are tasks that are executed just once during simulation. The `Setup` function is responsible for the following tasks:

- loading the configuration file;
- discovering the workpiece body ID and making sure there is only one workpiece;
- discovering cutter body IDs and making sure there is at least one cutter;
- configuring the cutters (discovering associated direction nodes and setting the local coordinate system of each cutter);
- discovering the spindle direction nodes; and
- populating up `elementIndex` and `nodeIndex`.

The procedure diagram for the `Setup` function is shown below in figure 9-11. The `Setup` function is called from `UMAKNET` on increment zero.

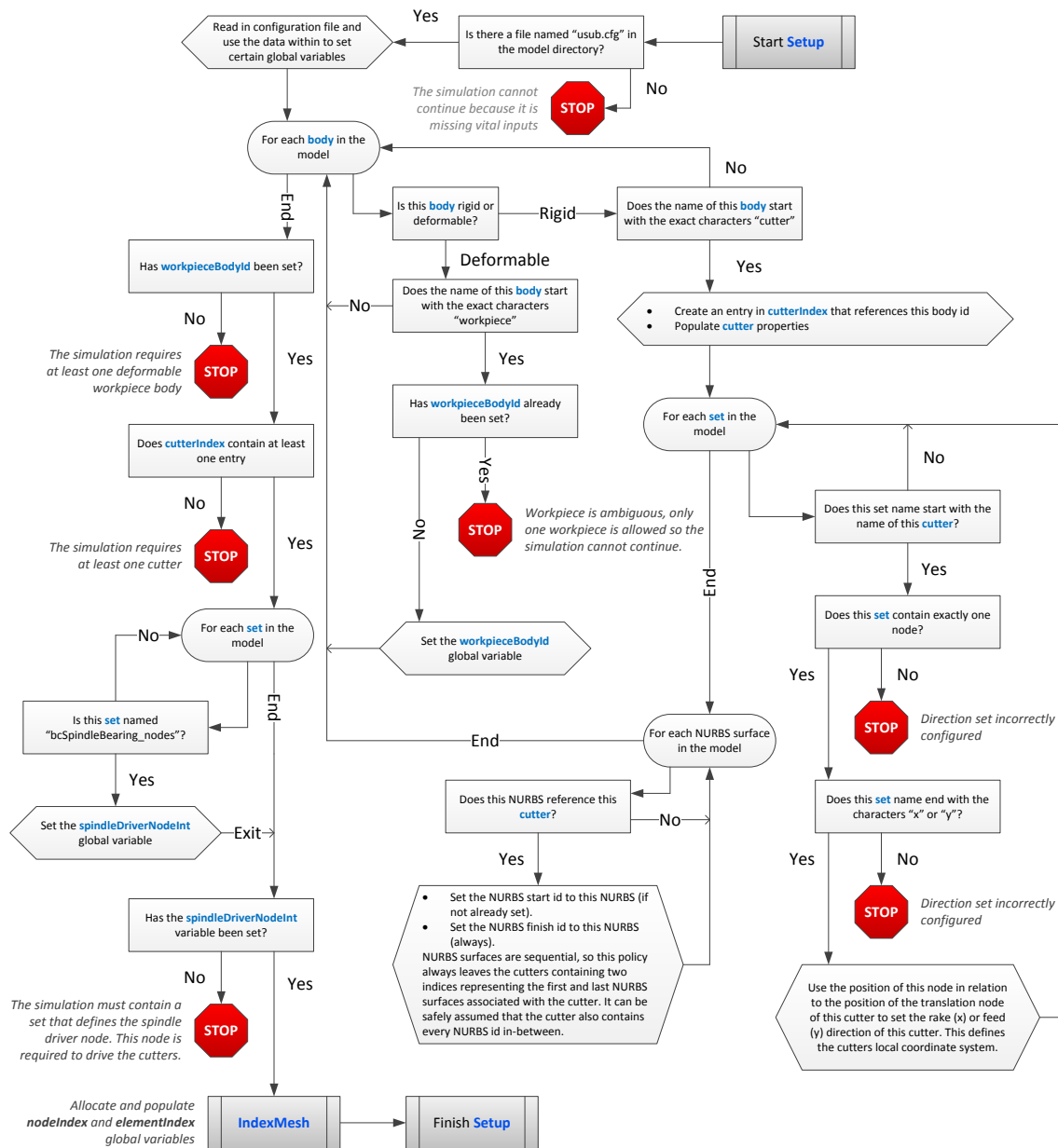


Figure 9-11 – `Setup` procedure diagram

## 9.7 SplitMesh

**SplitMesh** is a very long and complex function. The responsibilities of **SplitMesh** are carried out in just one subroutine as opposed to a set of subroutines due to the overlapping contexts between its responsibilities. For example, split **Rays** are used in remeshing the intersection face and then again much later when allocating forces. This helps to reduce the programming effort required to write additional subroutine headers and duplicate variable definitions for single use subroutines.

**SplitMesh** is laid out analogously to a production line where value is added at each step. Each block of code within **SplitMesh** systematically changes something about the mesh. This approach makes **SplitMesh** highly modular, certain blocks can be ‘turned off’ and the blocks following can still attempt their duties without modification (though often with a reduced chance of success). Figure 9-12 shows a high level procedure diagram for **SplitMesh**, giving reference where necessary to sections within this chapter describing the detailed methodology underpinning each block.

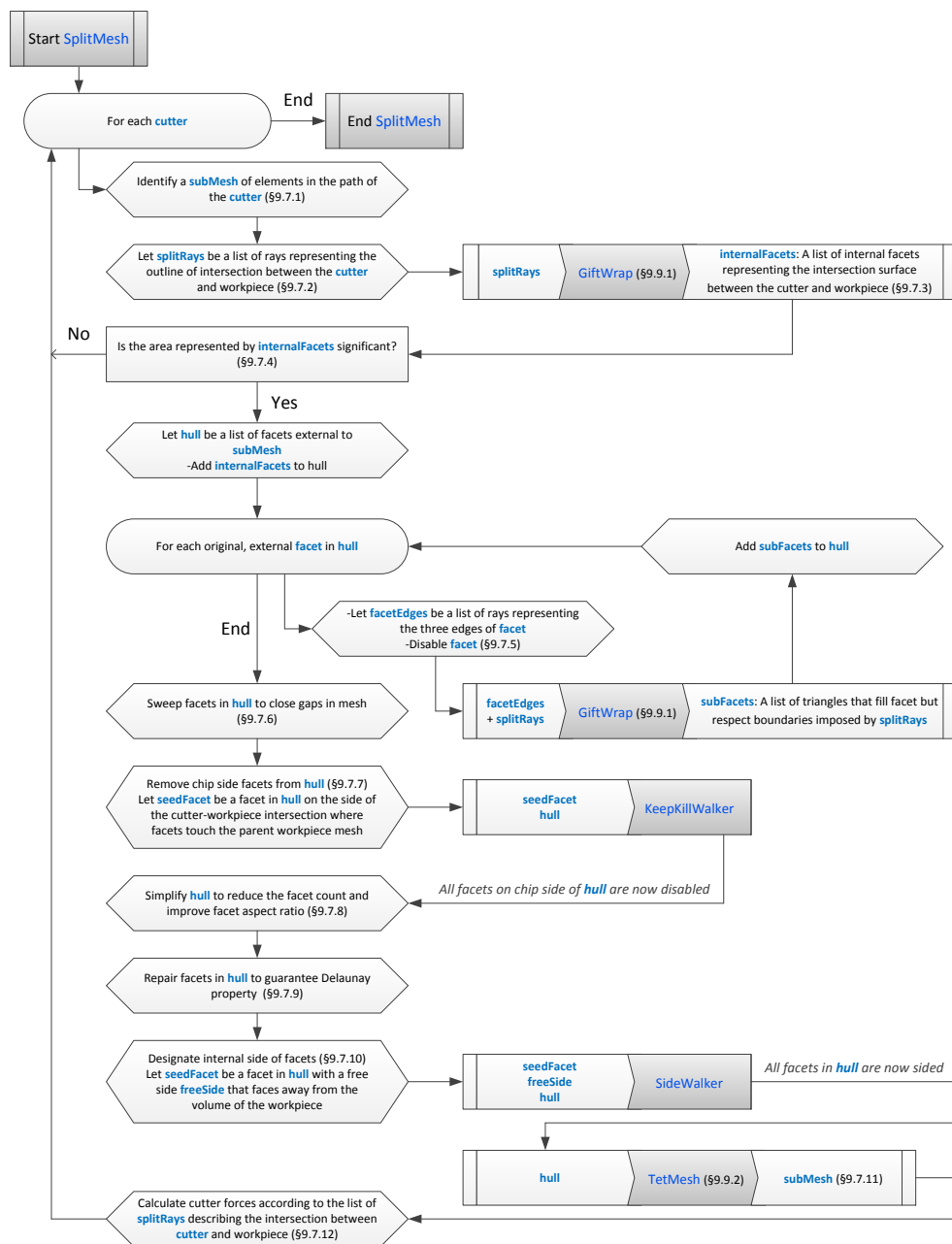


Figure 9-12 – **SplitMesh** procedure diagram

The remainder of this section walks through this procedure, explaining in detail how each step is accomplished with reference to the example workpiece-cutter intersection shown in figure 9-13.

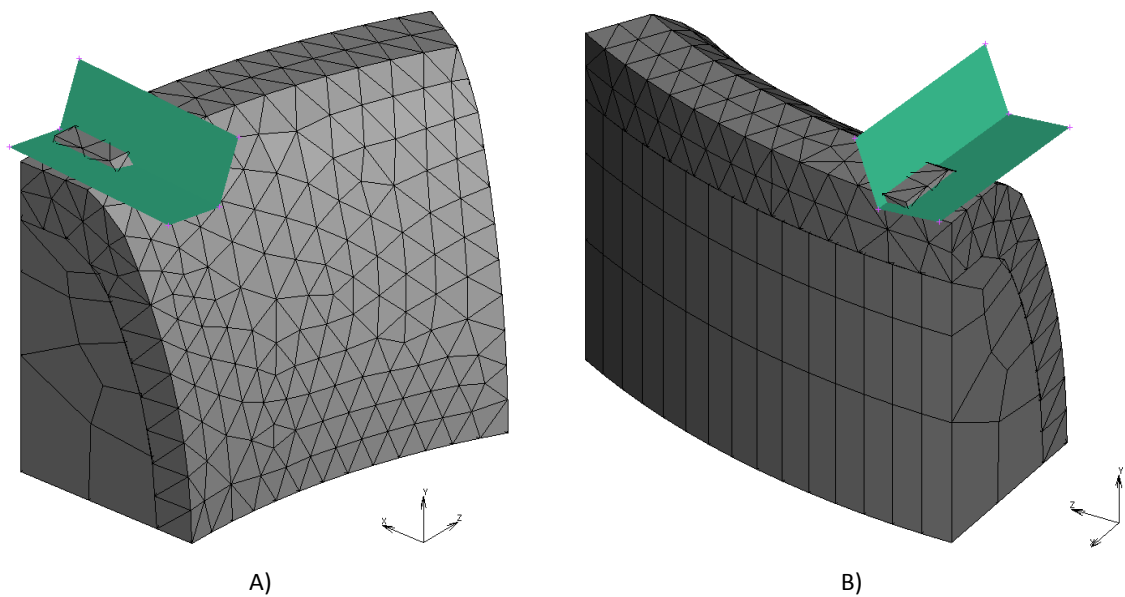


Figure 9-13 – Cutter-workpiece intersection

As the figure shows, the cutter is forming a full intersection, in other words, there is no path from a node on one side of the cutter, to the other side. This is an essential characteristic of intersections. [SplitMesh](#) will fail if the cutter only forms a partial intersection.

#### 9.7.1 Intersection Detection

Once the cutter is moved to its new location at the start of an increment, [SplitMesh](#) must identify the ‘sub-mesh’ or local portion of the workpiece mesh to remesh. The following steps are performed to narrow down the selection of sub-mesh elements. These steps are as follows:

- Let  $N$  be a set containing every **Node** in the model that carries the `F_workpiece` flag (assigned by [IndexMesh](#) to all workpiece **Node** objects).
- Let  $P$  be a subset of  $N$  containing only **Node** objects that are within some proximity distance of any point on a **Ray** that runs parallel to the cutters local radial direction, that lies on the maximum rake, maximum feed edge of a bounding box that surrounds the **Cutter**, as shown in figure 9-14. The proximity radius is set in the configuration file. This test is reasonably fast, however, it does require one square root operation per **Node**.

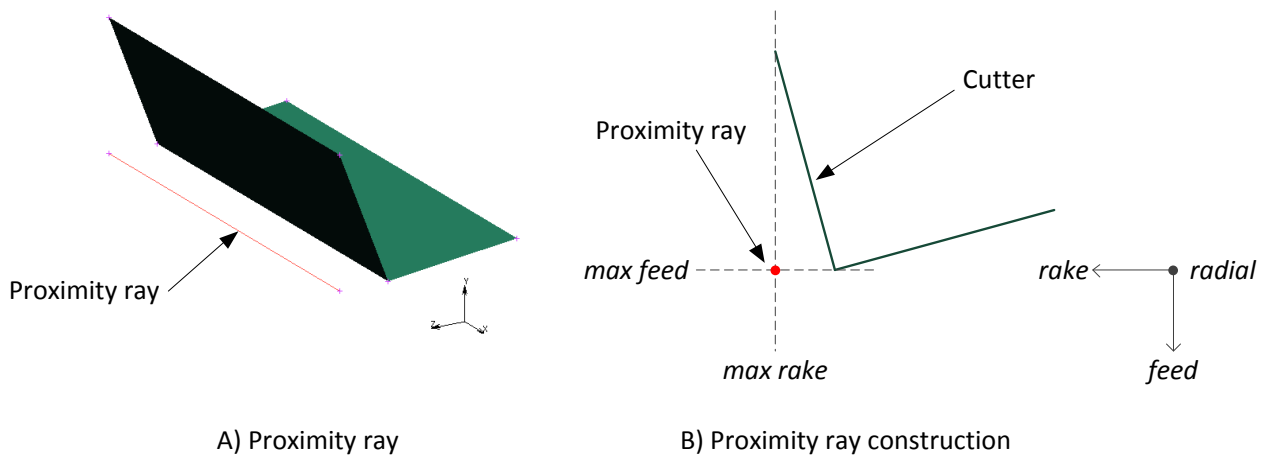


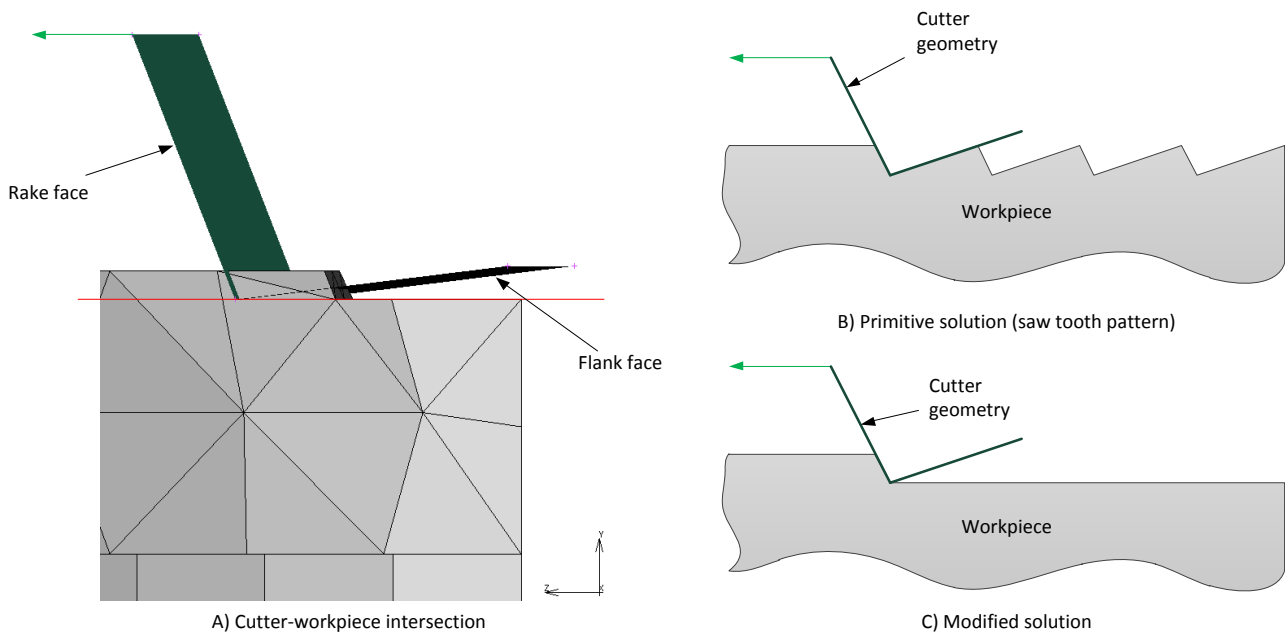
Figure 9-14 – Cutter proximity ray relative to cutter geometry

- Let  $E$  be a set of **Element** objects, that have **Node** objects in  $P$ . This test is very fast thanks to the precomputed elements-at-node cache.
- Let  $I$  be a subset of  $E$ , whose **Element** objects all have at least one edge intersected by the **Cutter**. This test is much slower than the others, since intersection testing is very expensive. Intersection testing is performed using the `NURBSurface%RayIntersect` method described later in section 9.8.10. **Rays** are generated for **Element** edges so that they can be used with `NURBSurface%RayIntersect`. Since edges are shared, whenever an edge is tested it is entered into a cache with the result of the intersection test recorded alongside it. This greatly reduces the number of intersection tests that must be performed.

The set in  $I$  is flagged as `F_Remesh`. These **Elements** fit together to form the *sub-mesh*, or part of the workpiece mesh that will be removed and replaced by a mesh of the newly cut geometry. The external facets of  $I$  form the *hull*, which will be split, simplified and volume meshed in the following steps.

### 9.7.2 Hull Splitting

After the sub-mesh external hull has been identified, it is split according to the cutter geometry. Figure 9-15, shows a discretisation problem that arises when attempting to split the hull, due to the way the cutter traverses the mesh.



*Figure 9-15 – Approximations of removed material*

The cutter is shown in A) from a side view of the workpiece. The cutter has just traversed the vector shown in green. The rake and flank faces of the cutter are defined according to the cutter geometry, invariably the flank face is not parallel with the cut surface. In reality, the cutter has moved through the workpiece material, leading to its removal, however, since this is a finite simulation, the cutter can only jump from one location to another. If the cutter were used as a tool to split the mesh in each location and remove the chip side, then this would lead to the saw tooth pattern shown in B). To achieve the desired solution, shown in C) several further steps are required.

When **UMAKNET** updates the positions of the **Cutter** objects for the next increment, it attaches two invisible NURBS surfaces to the lowest point of the **Cutter** to act as additional tool surfaces. The first of these tool surfaces is the extrusion of the lowest edge of the cutter through the inverse of the cutters affine transform (which describes its displacement from the last increment to the current). This method takes into account the curvature and descent path of the **Cutter** as it rotates. The second tool surface attaches to the rearmost end of the previous tool surface and projects up at approximately a 45 degree angle. This surface only intersects the workpiece due to floating point rounding error, and is intended to guarantee a full intersection is formed. The vast majority of increments will not require this surface. All non-front facing parts of the **Cutter** NURBS surfaces are disabled for the remainder of the increment. The algorithm developed for this work performs this non-front facing removal on a sub-NURBS basis, meaning that it is capable of subdividing a curved NURBS and only removing part of it to satisfy the front facing criteria. Figure 9-16 shows a side view of the cutter with these additional tool surfaces.

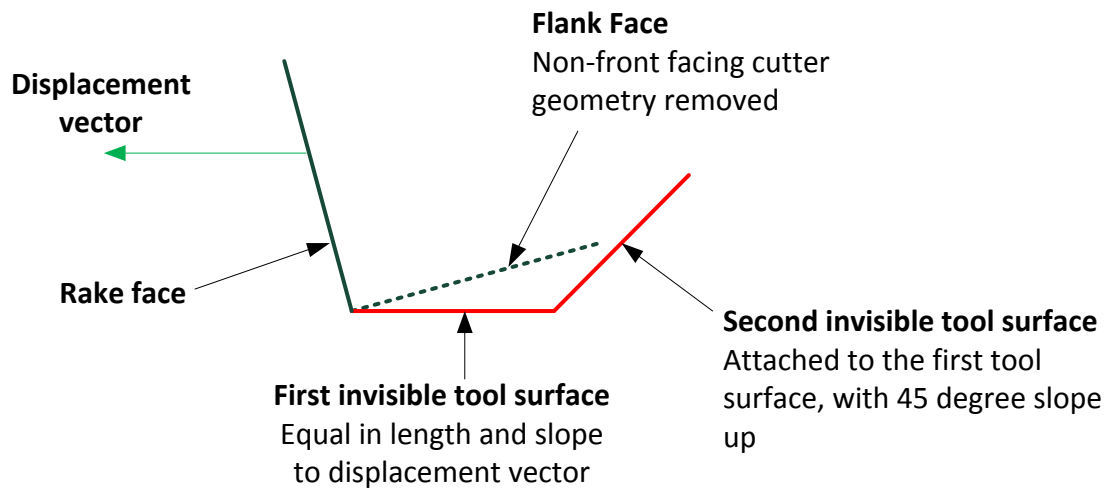


Figure 9-16 – Invisible tool surfaces

The new tool surfaces are created as **NURBSSurface** objects allowing them to seamlessly interface with other parts of the program that operate on NURBS. This removes the requirement to write special routines to handle invisible tool surface NURBS.

Figure 9-17 A) shows several cutters in the simulation domain (purple) with the tool surface NURBS shown (green). B) Shows several angles of one cutter with both tool surface NURBS. In this figure, the sweep angle of the tool surface NURBS is doubled for better visualisation of the curvature.

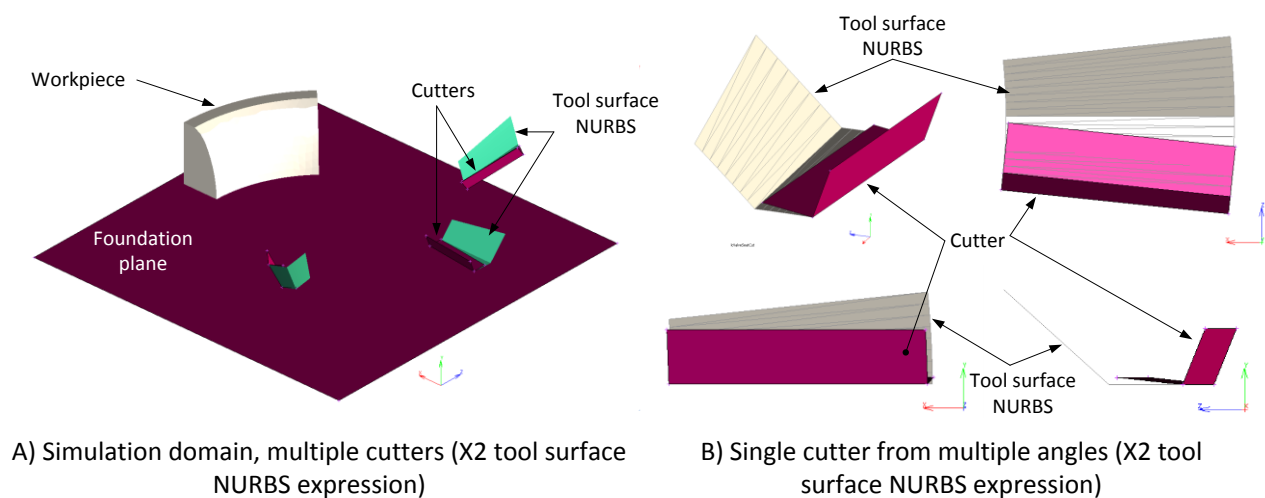


Figure 9-17 – Cutters and tool surface NURBS (x2 expression of new tool surface NURBS)

All hull facets are submitted to the **NURBSSurface%TriIntersect** algorithm (described in section 9.8.11) for each **Cutter** NURBS, including the two newly formed tool surface NURBS. For each call, if an intersection is found, a list of split **Ray** objects is returned.



All split **Ray** objects are assembled into a single list. This list is ordered so that the **Ray** objects run end to end. Figure 9-18 below shows an example of the split **Ray** objects plotted in green over the sub mesh region. The parent mesh shape is shown in light grey in the background.

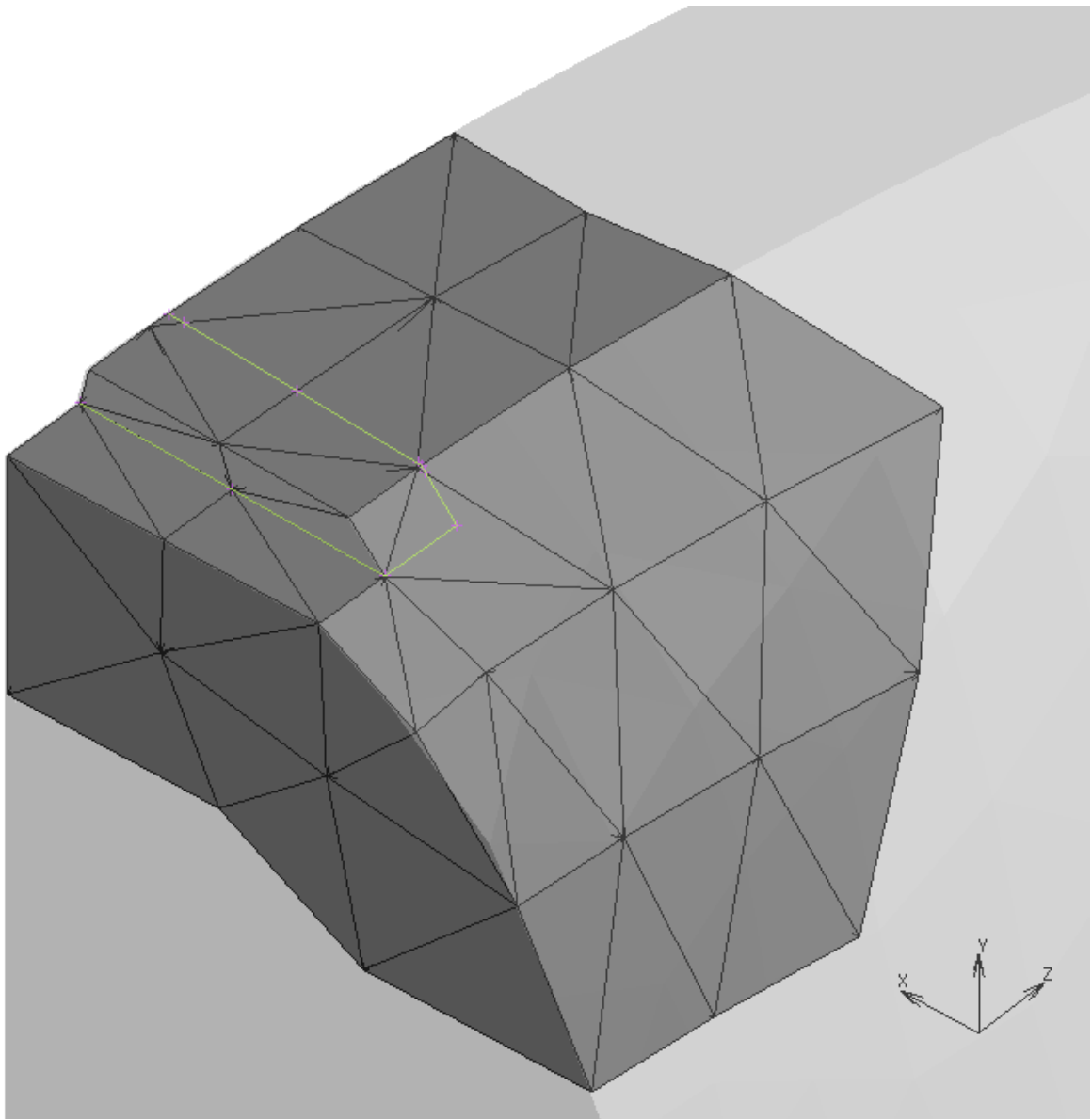


Figure 9-18 – Split **Ray** objects overlaid on workpiece

It will often be the case that a split **Ray** falls almost exactly on a pre-existing mesh segment. This is because the cutter must begin almost exactly where it last finished in order to preserve the continuity of the cut. This creates a problem as the more exact the overlap, the more ambiguous it becomes as to which facet the split **Ray** splits.

To resolve this issue, two steps are taken. Firstly, the triangle-triangle intersection algorithm will not report an intersection for triangles that intersect on their edges. This causes the broader cutter/workpiece intersection routine to fail to generate a **Ray** for that part of the intersection loop. Once all **Rays** are generated, the gap between **Rays** is tested to make sure they join up, this gives the algorithm an opportunity to detect the missing **Ray**. If a missing **Ray** is detected, the algorithm will create a small perturbation in near by **Nodes** and repeat the intersection detection. This process will repeat until a fully closed loop is formed or a recursion limit is reached.

### 9.7.3 2D Meshing – Cut Face

Before the hull mesh can be split according to the split **Rays** generated in the previous step, a new surface mesh must be produced to cap the void that would be exposed by removing part of the hull. This surface is equal to the region of space occupied by both the **Cutter** tool surfaces and workpiece volume.

Although **Giftwrap** (the two dimensional mesher developed for this project described in section 9.9.1) works in R3, it requires the calling function to guarantee all point and edge seeds passed to it are on the same plane (the orientation of the plane is irrelevant). For this reason, the 3D intersection **Rays** must be flattened.

The method used here takes advantage of the fact that all NURBS surfaces can be represented easily in a 2D parametric domain, and that all intersection mesh facets will lie on a NURBS surface.

A map is created in parametric space that contains all the adjacent **Cutter** NURBS surfaces in relation to one-another. The edge **Rays** are aware of which NURBS surface they were generated on and are subsequently mapped to the parametric domain with the appropriate offsets so that they maintain their positions relative to one-another. Figure 9-19 below shows an example of this mapping, A) shows the intersection **Rays** in 3D space. The green line indicates the transition between two NURBS surfaces. B) shows the split rays mapped to the parametric domain. This is achieved using the **NURBSurface%GetUV** function discussed in section 9.8.4.

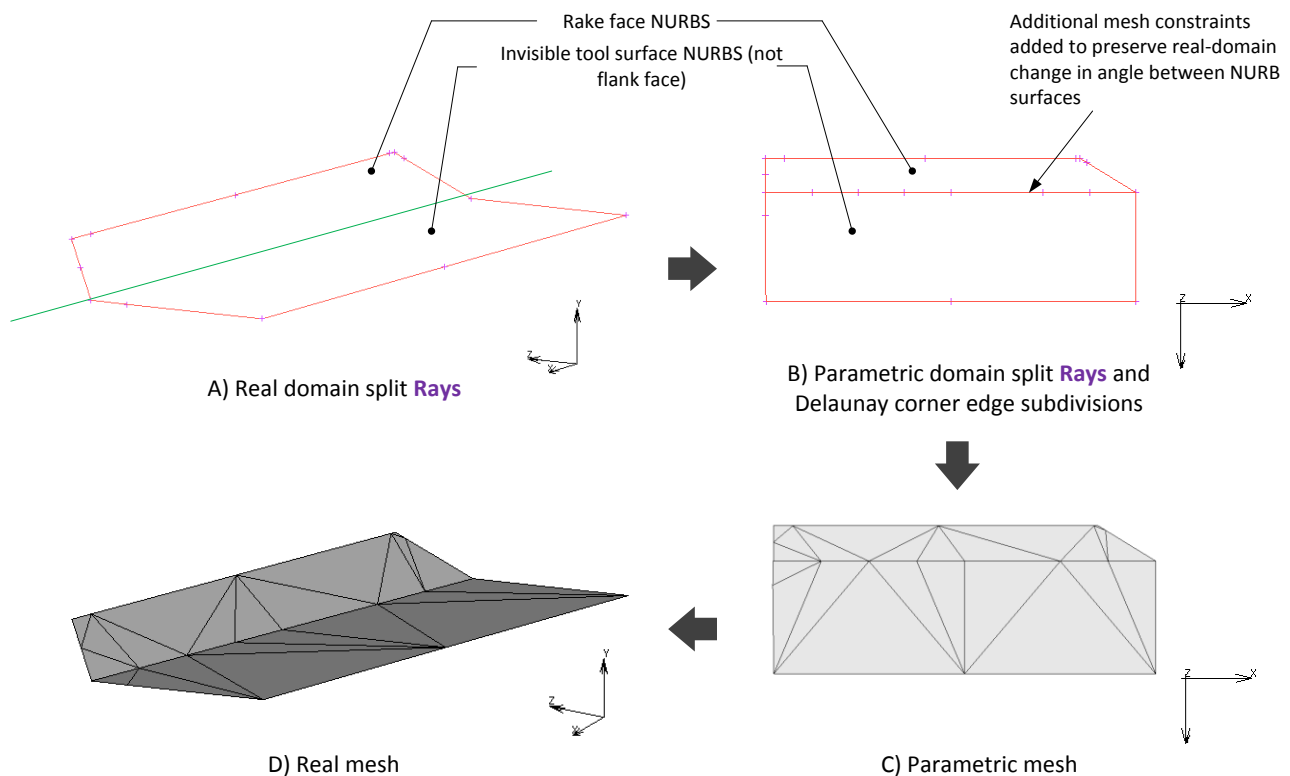


Figure 9-19 – Progression of cutter face mesh

Additional edge constraints are added at the interfaces between NURBS surfaces if they form a steep angle to one another. This prevents the mesher from placing a facet that overlaps this fold, as such a facet would create a web when mapped back to the real domain.

After a parametric domain mesh has been created as shown in C), the connectivity data is directly copied and the mesh is rendered using the list of real coordinates (as they were before conversion to parametric

coordinates). This results in the final real mesh shown in D). This set of mesh facets is simply inserted in the same list as the general hull facets.

#### 9.7.4 Significance Check

Once **SplitMesh** has reached this stage, it performs a check to make sure this cut should be processed. **SplitMesh** will stop processing this **Cutter** and move to the next one, if:

- the **Cutter** was not in contact with the workpiece in the previous increment; and
- the cut face area is below some minimum threshold.

This check is performed to prevent **SplitMesh** subdividing the mesh for grazing intersections that commonly occur when running the simulation in intermittent cutting mode. These intersections can be processed, but they can unnecessarily increase the complexity of the wider mesh, and increase the risk of a failed mesh.

#### 9.7.5 2D Meshing – Split Hull Facets

Next, the hull facets that are split by the intersection of the **Cutter** and hull must be meshed. These facets are meshed individually due to the requirement of **Giftwrap** that all points are on one plane. During splitting, an ordered list of closed loop split **Rays** are generated. Due to the way Triangle-Triangle intersection testing works, **Rays** in this list are guaranteed to originate and terminate only on facet boundaries or corners.

Figure 9-20, A) shows a group of facets, A through E, split by **Rays** 1 through 7.

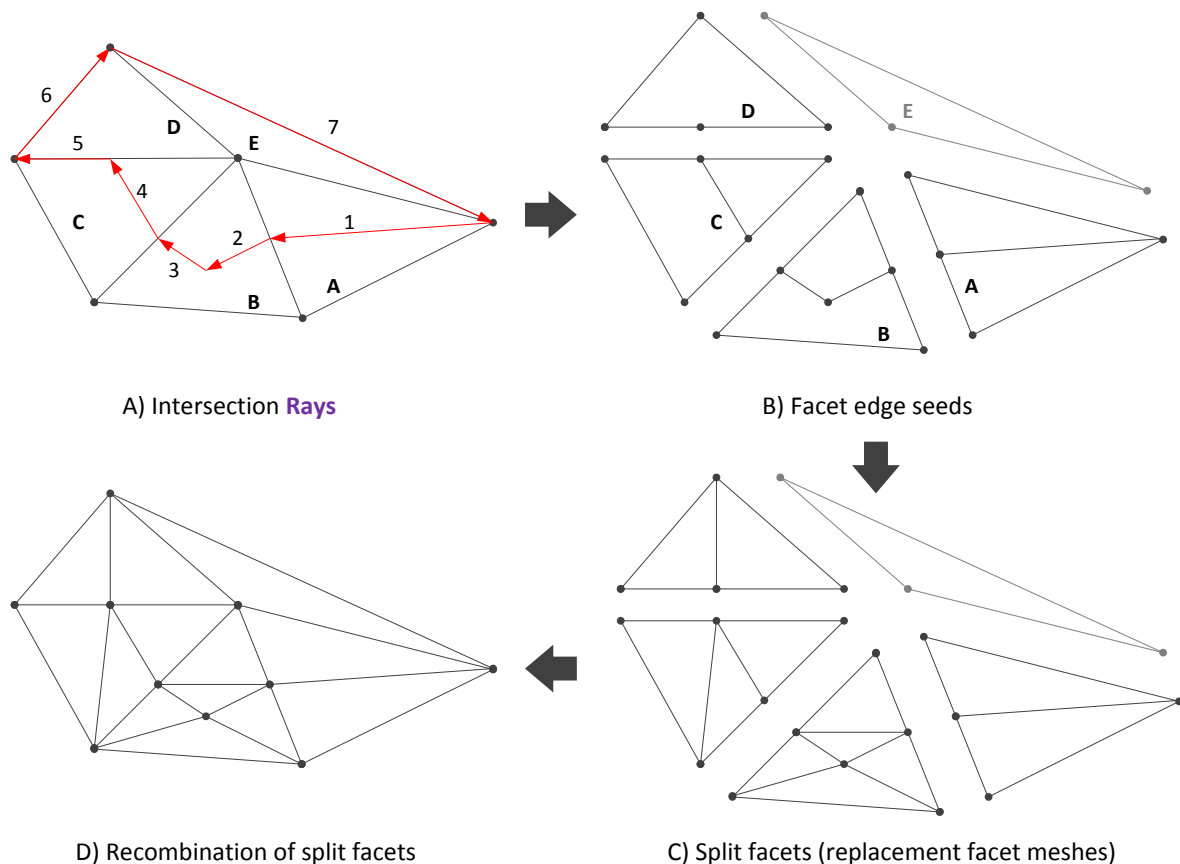


Figure 9-20 – Progression of hull facet meshing

Each of the facets in A) contains a different type of split that can occur, handled as described in table 9-3, below.

Facet	Ray(s)	Features & special handling steps
A	1	<b>Rays</b> start on a point and end on an edge. A new node is added on the edge, but the corner node can be shared.
B	2,3	More than one <b>Ray</b> . Nodes are added everywhere a <b>Ray</b> starts or ends.
C	4,5	One ray runs partially along an edge. The partially intersected edge is split where the <b>Ray</b> starts, but the <b>Ray</b> itself is discarded to avoid an overlap.
D	5,6	One <b>Ray</b> runs partially along an edge and another runs entirely along an edge. Although nothing disrupts the middle of this facet, it must still be split because one of its edges were split as described when processing facet C.
E	7	One <b>Ray</b> entirely on one edge. Despite being associated with a <b>Ray</b> , the <b>Ray</b> starts and ends on pre-existing corners of this facet and therefore nothing needs to be done, since the result of a split would be a facet identical to the root facet.

*Table 9-3 – Facet characteristics*

All facets associated with a split **Ray** are submitted to the routine described in figure 9-21. **Ray** deactivations in this routine are temporary until the next facet is checked.

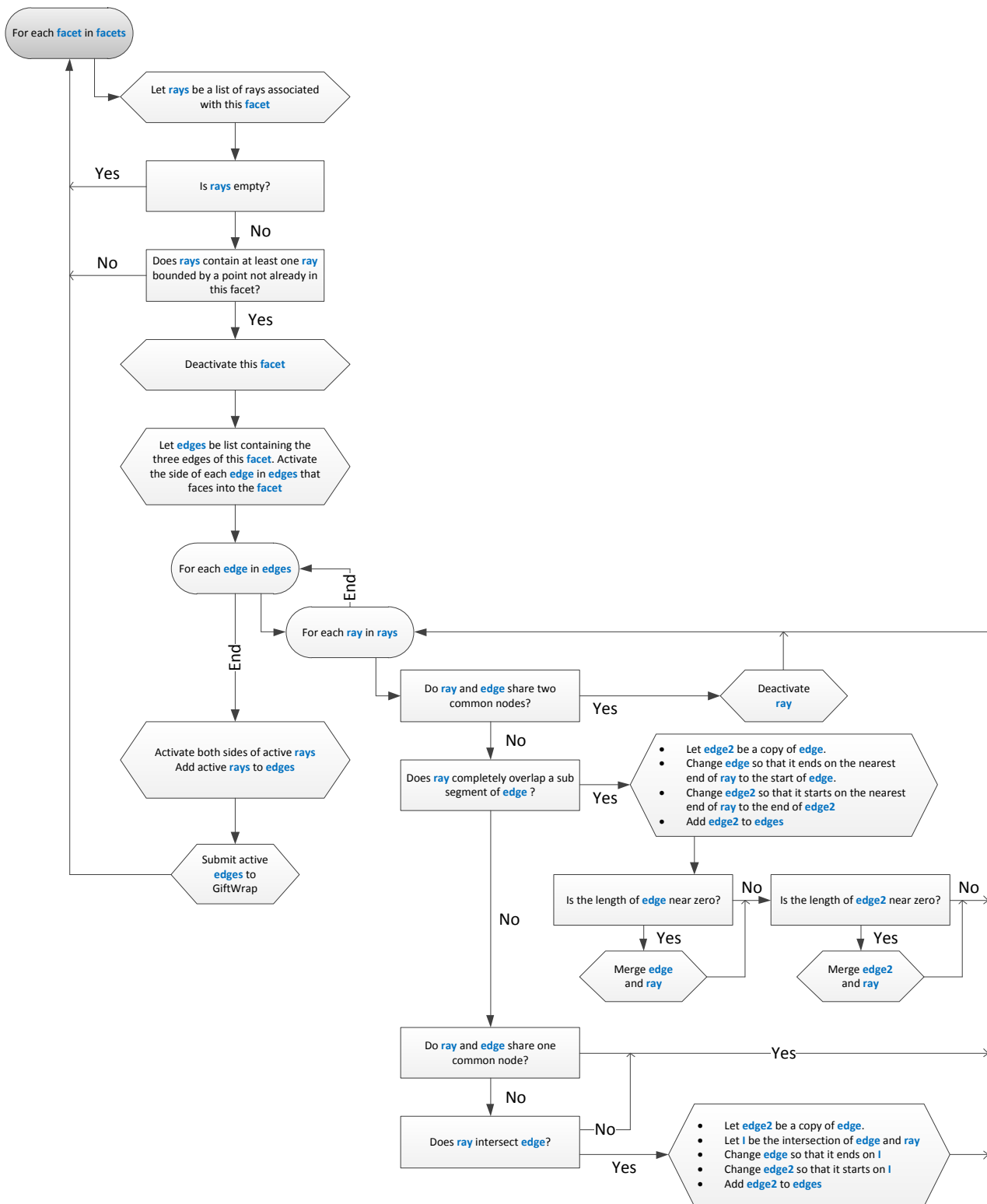
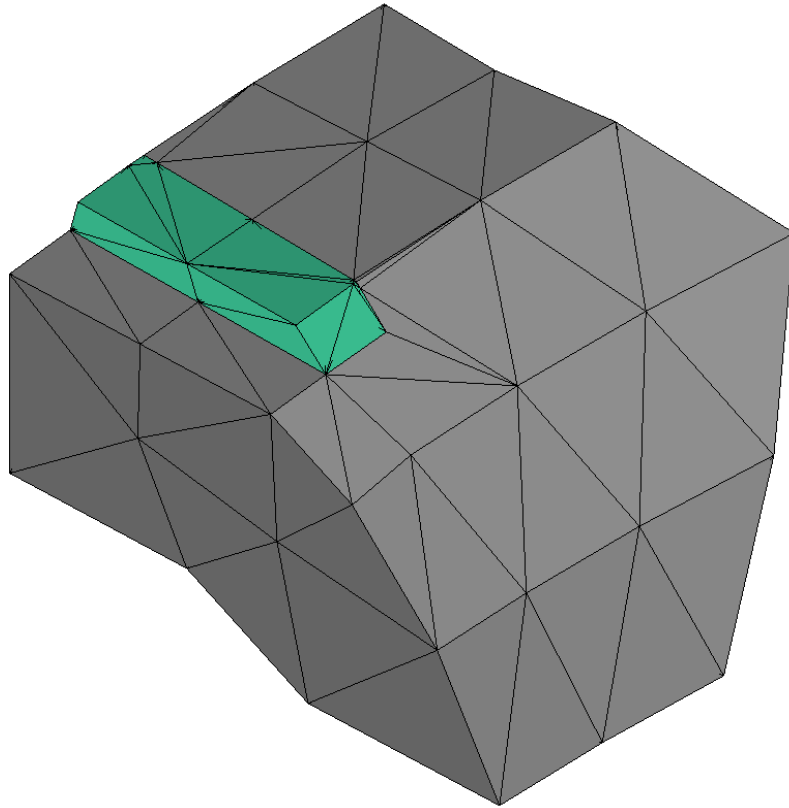


Figure 9-21 – Facet mesh procedure diagram

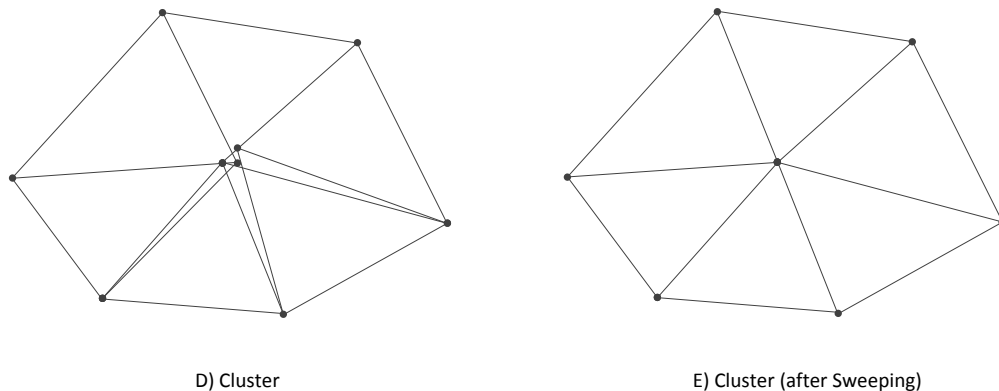
Figure 9-22 shows the state of the hull facet mesh up to this point. Although the hull is now split and the interior cutter face intersection mesh is generated, the facets are not optimised for volume meshing. There are needles, sharp changes in mesh density and unnecessarily non-Delaunay triangles.



*Figure 9-22 – Result of hull splitting, where the highlighted elements are on the opposite side of the split to non-highlighted elements*

#### **9.7.6 Simplification I – Sweep**

Sweep, so named after the similar tool in the Marc user interface, is a very simple routine that merges tight clusters of nodes. Such clusters arise in a number of scenarios, usually as a result of the cutter splitting a facet very near the tip of that facet. If not eliminated, these clusters can create a number of hazards, for example, needle elements, point elements and inside-out elements, all of which are highly likely to result in a failed mesh. Figure 9-23 shows an example of a node cluster before and after sweeping.



*Figure 9-23 – Facets before and after sweeping*

Sweep will typically generate a new coordinate for the sweep node from an average of coordinates within the cluster. An exception to this rule is that if the cluster contains one or more legacy nodes (nodes that existed in the previous increment) then sweep will only use the legacy nodes to generate an average for the new coordinate. This helps to preserve the current workpiece shape rather than performing actions which may ‘dent’ or ‘pinch’ the external geometry when merging a cluster.

### 9.7.7 Removal of Unwanted Hull Side

Sweep (described previously) and General Complexity Reduction (described next) are both simplification strategies. It may seem odd therefore to remove the unwanted side between executing the simplification routines. However this is done for the following important reasons:

- Sweep must run before the removal of the unwanted side because small clusters will often confuse `keepkillwalker` (the function used to ‘paint’ which sides to keep and which to remove) due to the small and often overlapping facet edges that occur in clusters, thus causing it generate an undesired result without failing; and
- General complexity reduction should run after the removal of the unwanted side because it will have more freedom to make simplifications if not constrained by the chip side which will not be present in the final mesh anyway.

As mentioned, `keepkillwalker` is the name given to a function which paints facets as ‘keep’ or ‘kill’. `keepkillwalker` is a simple recursive function that follows the procedure shown in figure 9-24.

`keepkillwalker` is seeded with one facet that will appear in the final mesh. This seed facet is selected based on its nodes, if all of its nodes are shared by an element which is not marked as near the cutter. In other words, an interface element that the new mesh will mate with.

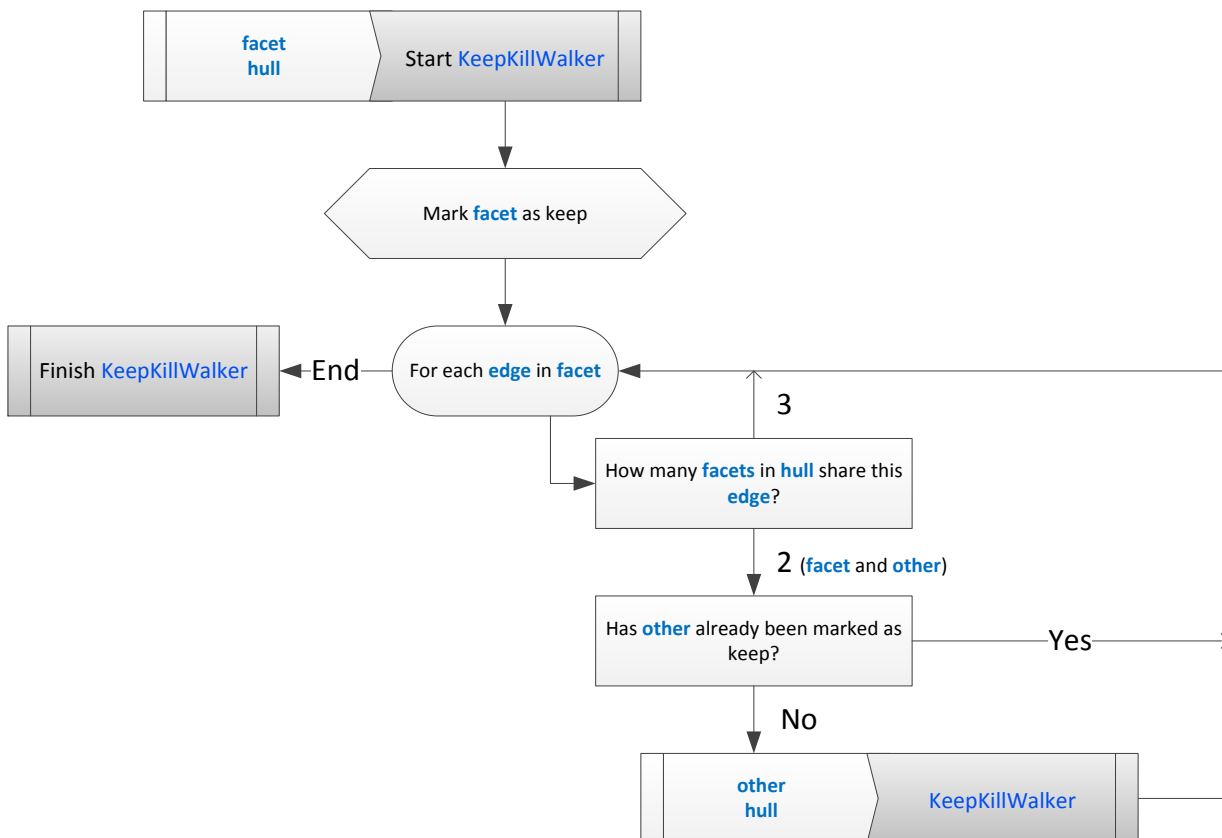


Figure 9-24 – `keepkillwalker` procedure diagram

This method works because the cutter face intersection mesh is inserted inside the hollow hull. Therefore, the external edges of the cutter face mesh will always be shared by three facets, whereas every other edge in the hull will only be shared by exactly two facets. This property of the cut plane mesh external boundary edges is exploited to determine the side to keep by preventing [keepkillwalker](#) from crossing edges shared by three facets. Figure 9-25 shows a cross section of the hull showing the external hull facets, split plane facets and edges that are shared by three facets.

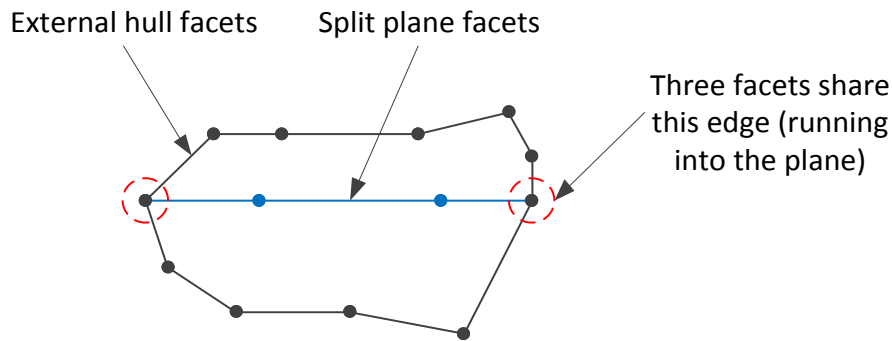
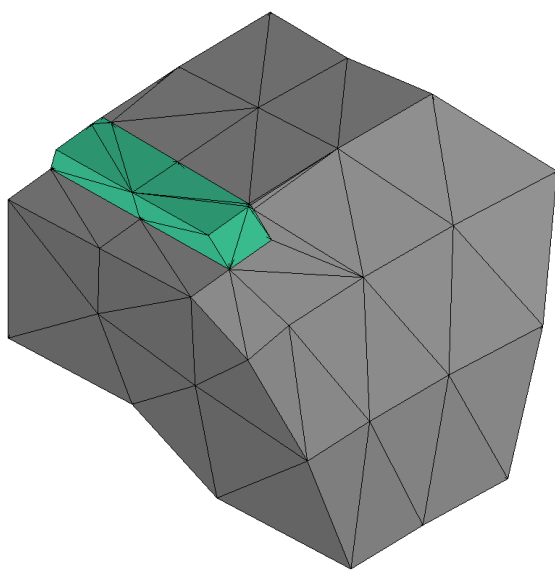
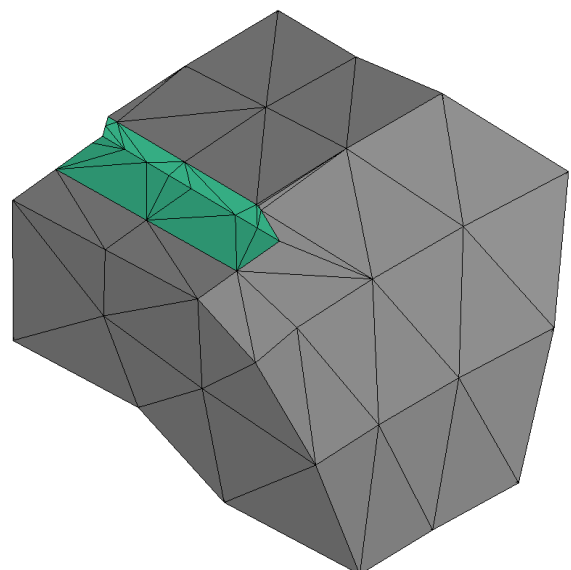


Figure 9-25 – Hull cross-section showing internal cut face mesh facets

[keepkillwalker](#) is called with the known 'keep' facet. Once it returns, all facets (excluding split plane facets) not flagged as 'keep' are disabled. Figure 9-26 A) shows the facets marked for removal, and B) shows the resulting mesh with the newly exposed cutter face facets highlighted.



A) Kill facets



B) Chip side removed, exposing cutter face facets

Figure 9-26 – Hull facets A) before, and B) after, removal of unwanted side



### 9.7.8 Simplification II – General Complexity Reduction

The success of TetMesh is never guaranteed. Some hull meshes are impossible to tetrahedralise, however, it is NP-hard to determine before trying, whether or not a mesh will generate successfully. Although it is not impossible for TetMesh to create a mesh of the raw hull facets generated up to this point, the chances of success are drastically increased by first simplifying the mesh.

General complexity reduction attempts to reduce complexity by merging coplanar facets that share collinear edges. General complexity reduction can easily identify candidates for simplification, but is complicated by the large number of checks that must be performed to prevent a potential simplification from breaking the mesh. Some of the challenges that can arise are as follows.

Firstly, general complexity reduction must detect the case where it is about to create a ‘fold over’ facet. Figure 9-27 shows how this type of case can arise. Consider the four facets, A through D in A) in the figure. The algorithm has identified that facets that share the red node, A, B and C are all coplanar. Furthermore the lower edges of A and B are collinear. The common node is identified as a suitable candidate for the simplification shown by the red arrow. However, as the figure shows, B would be eliminated (which is a desired outcome), but C would fold over D. The combined area of C and D would also be partially overlapped by A. In this scenario, the general complexity reduction algorithm caches the normal vector of each facet (according to the global side definitions given in section 9.2.5). If the new direction vector is opposite to the cached normal vector, then the facet (C) is eliminated as are any facets that share all of the fold over facets points (D). In this example, only A would remain as shown in B) in the figure, as would one orphaned point which is removed in a subsequent step.

A) Before proposed simplification

B) After proposed simplification

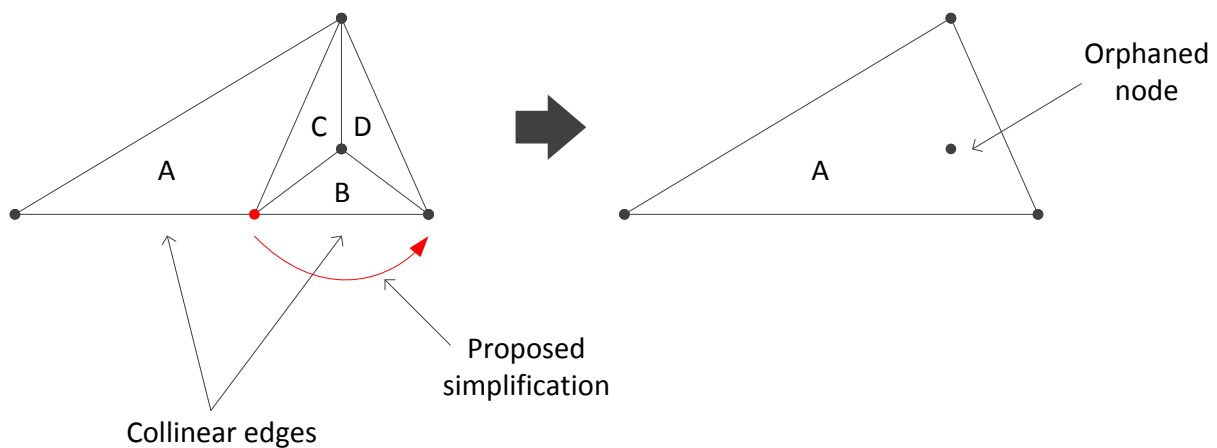


Figure 9-27 – Mesh simplification that would lead to overlap

Simplifications that cause significant changes in external geometry are not allowed. In figure 9-28, every external edge depicted in blue has a facet that extends into the plane. Figure 9-28 A) shows a proposed simplification that would eliminate facets C and D, however, it would also significantly change the external geometry (including the facets extending into the plane). To check if a simplification will significantly change the external geometry, the new position of the common node (red) must be within COARSE of all affected facets as they are in their pre-simplification state, measured along each facet's normal vector. Figure 9-28 B) however, shows a legal simplification. In this example, no external edges are disturbed.

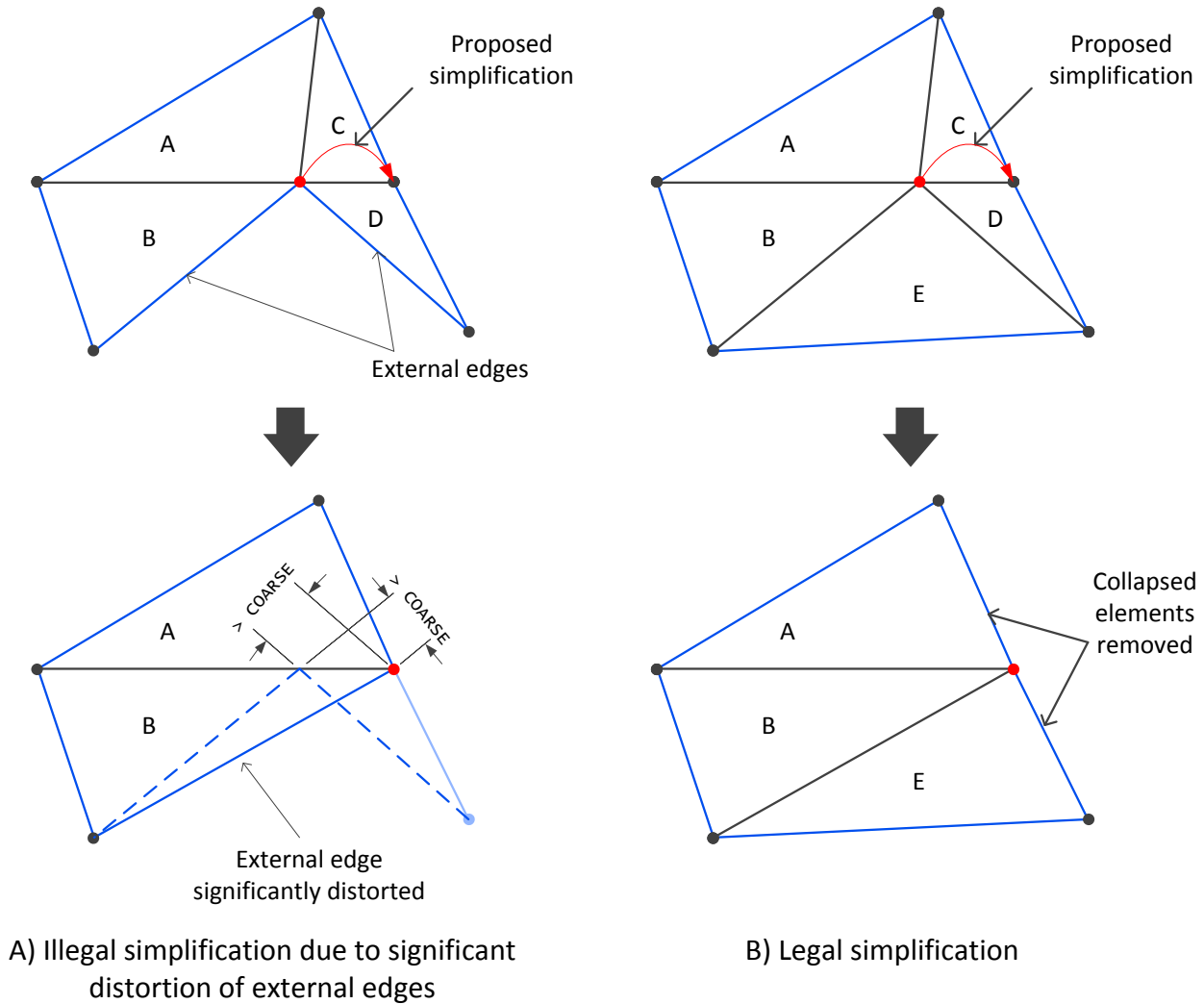
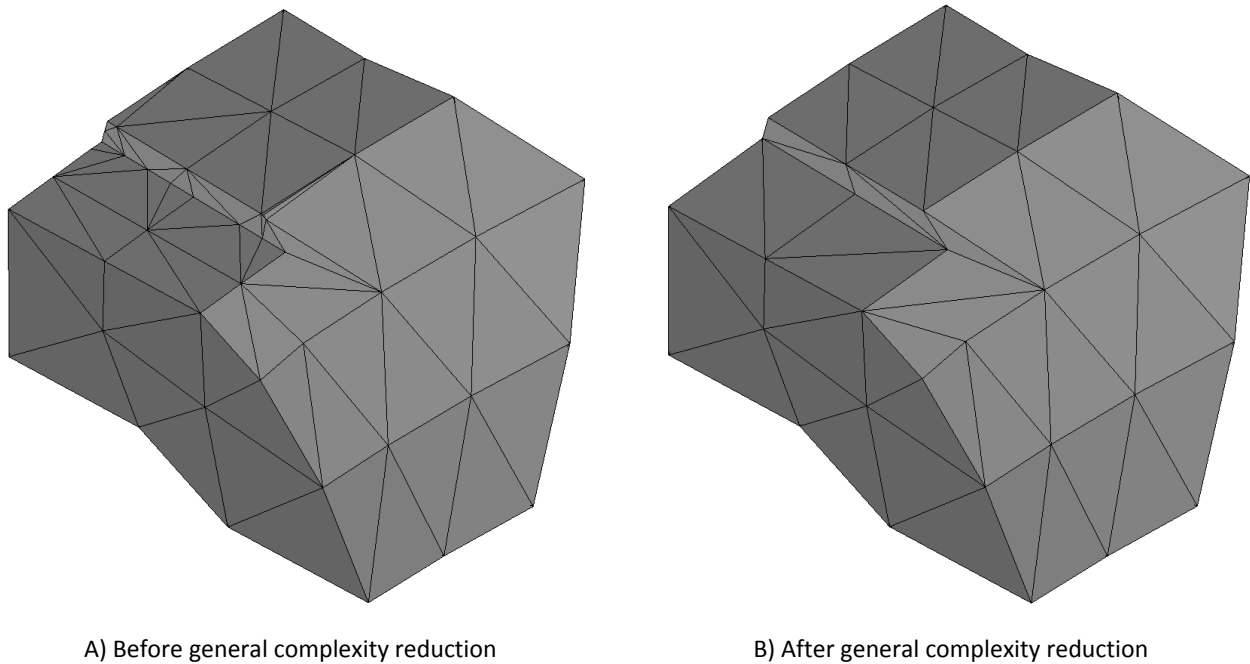


Figure 9-28 – Facet simplification that would significantly change the external mesh shape

Nodes are free to move so long as they remain on the planes of all facets associated with the node. This method is very robust at preserving the sharp corners of the mesh whilst still allowing the smoother edges and interior to be simplified.

Figure 9-29 shows the hull facets before and after general complexity reduction.



*Figure 9-29 – Hull facets A) before, and B) after, general complexity reduction*

#### 9.7.9 Mesh Optimisation

The gift wrapping algorithm used within [TetMesh](#) will naturally try to generate Delaunay elements and Delaunay facets. However, since the algorithm is being used in this work to generate a sub mesh that mates with a parent mesh, it must respect the boundaries of the parent mesh.

Conflicts can occur when the gift wrapping algorithm attempts to generate a tetrahedral with a facet that conflicts with another facet occupying the same space. Mesh optimisation attempts to resolve this problem with two strategies.

**Prevention** - Before meshing, all facets are checked to see if they are free, or if they mate with the surrounding mesh. Those that are free are subjected to an edge flip algorithm. The edge flip algorithm checks all pairs of facets that share an edge, to see if they are Delaunay. Those that are not, are replaced with the same quadrilateral, but split along the counter-segment instead (making them Delaunay). There are some exceptions where it is not possible to use the counter-segment, for example, pairs of facets that form concave quadrilaterals, however these are always Delaunay anyway.

Figure 9-30, A) shows the state of the sub-mesh before edge flipping. The particular triangulation of quadrilateral facet pairs depends entirely on which simplifications were made in the previous step. Simplifications are considered even if they would result in non-Delaunay facets. After the edge flipping routine, non-Delaunay facets are adjusted where possible producing the mesh seen in B).

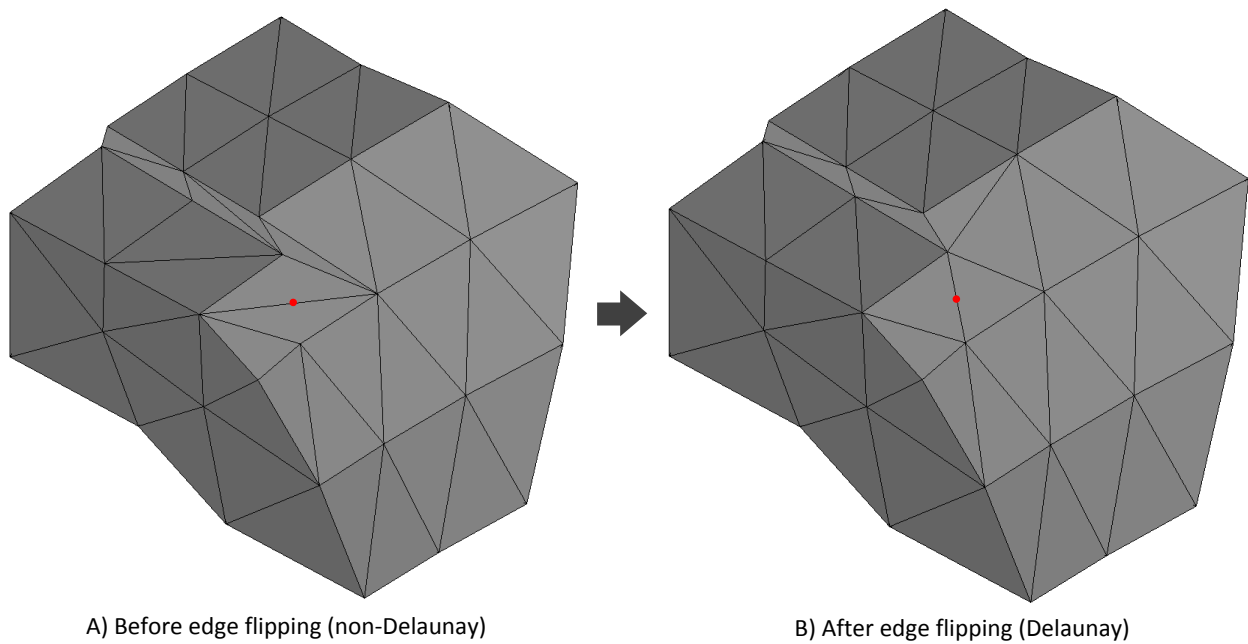


Figure 9-30 – Hull facets before and after edge flipping (red dot indicates one example of a flipped edge)

**Restriction** - Restriction uses a list of *illegal segments*, to flag certain segments that **TetMesh** maybe tempted to create, that should not be present in the final volume mesh.

Figure 9-31 shows A) shows the external hull triangle that must be tetrahedralised. **TetMesh** identifies  $p_1$  as the best candidate to complete the tetrahedral (see section 9.9.2 on **TetMesh** for an explanation of how this point is selected).

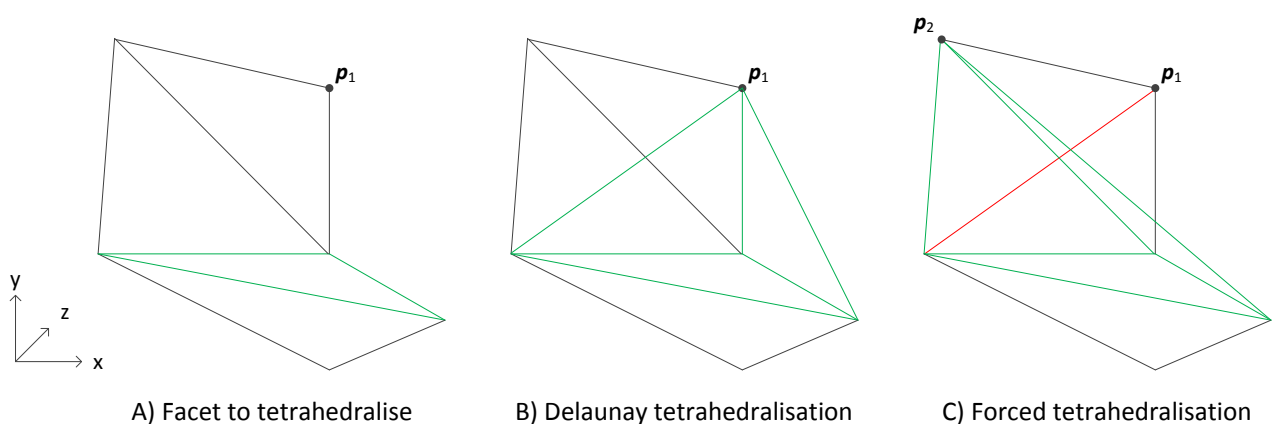


Figure 9-31 – Delaunay segment vs. interface segment conflict

$p_1$  would generate a tetrahedral that has four Delaunay facets, however, as part B) shows, one of the facets conflicts with a pre-existing hull segment. During tetrahedral meshing, if **TetMesh** attempts to make a tetrahedral that contains an illegal segment, then the next best point will be considered to avoid creating

the illegal counter-segment. This is illustrated in C), where the illegal segment is shown in red. The presence of the illegal segment forces the algorithm to consider  $p_2$  - the next best point, and so on to  $p_n$  until the tetrahedron is resolved, or the number of points has been exhausted.

There are two types of illegal segments, *mandatory* and *exclusive*. Mandatory illegal segments must not appear in the mesh under any circumstances. These are the segments that would prevent the sub mesh from mating perfectly with the parent mesh. Exclusive illegal segments are not preferred by TetMesh, but will be considered as a last resort providing their counter-segment does not already exist in the mesh. These are the external segments that may or may not share a glue contact with the workpiece bulk.

All counter-segments of segments that share two coplanar facets are illegal. Counter-segments, of segments that exist in the non-remeshed portion of the workpiece, are mandatory illegal segments.

Figure 9-32 shows an example of mandatory illegal counter segments (light blue) overlaid on top the mating facets (blue) of the sub mesh hull facets. There are three mandatory illegal counter segments in total in this mesh. The counter-segments of all non-mating segments that share coplanar facets are exclusive illegal segments (not-shown). The counter-segments of non-coplanar facets are extremely unlikely to form and do not need to be registered as an illegal segment.

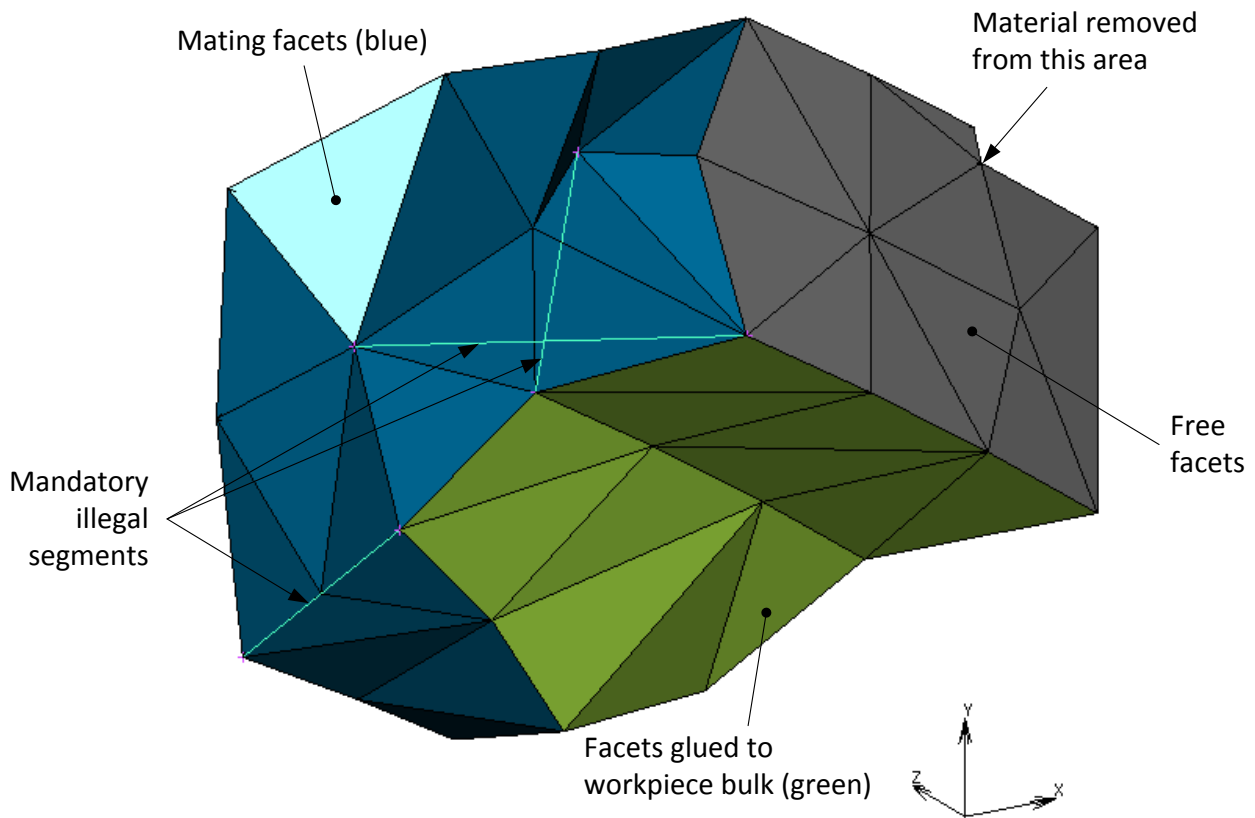


Figure 9-32 – Illegal segments overlaid on immovable mating facets

Restriction is the least desirable control, because once a tetrahedron has been forced in this way, the mesh becomes unstable and the probability of failure is slightly increased. The presence of the non-Delaunay element, may nucleate other non-Delaunay elements around it. Furthermore, it should be noted that the alternative shown in figure 9-31 is generous. Many real-world next-best points create distorted tetrahedrons. For this reason, the mesher creates as many Delaunay elements as possible before it resorts to generating non-Delaunay elements.

### 9.7.10 Facet Siding

The final step before submitting the hull facets to [TetMesh](#) is to flag free sides. Facets have two sides, and all hull facets defined up to this point have just one free side (the internal side). [TetMesh](#) will only create tetrahedrals on free sides. All internal facets that may be created by the formation of new tetrahedrals will have their side which faces away from the new tetrahedral, set to free.

Siding of facets is critical for [TetMesh](#) to know which side of the hull to start constructing the tetrahedral mesh on. Siding works by first finding any facet that has a definite obvious external and internal side. Once this nucleating facet is found, [Sidewalker](#) traverses all facets in the hull, assigning their free side relative to the last facet side assigned. This process uses the common side definitions as defined in section 9.2.5.

The nucleating facet is the first hull facet which has one side that can see at least one other facet and another side that can see no other facets. This test must be prepared to iterate through all potential nucleating hull facets as concave hulls can contain external facet sides that can see other hull facets. This is a very expensive test which is performed by testing a [Ray](#) that originates at the centre of the potential nucleating facet, against every other facet in the hull, using [Ray%TriIntersect](#).

Once the nucleating facet is found, its free side is set according to the convention established in section 9.2.5, where the free side is the one that can see any other facet. This facet is passed to the recursive function, [Sidewalker](#) which propagates the siding throughout the hull.

Facets are recorded in terms of their corner point IDs. [Sidewalker](#) uses the rule that, if an adjacent facet defines the common edge in the same order as the root facet (clockwise or anti-clockwise) then the free side of the adjacent facet is opposite to that of the root facet. Figure 9-33 shows two adjacent triangles,  $i$  and  $j$ . In case A), the node ID loop definitions for each facet specify the shared edge in the same order. Using the right hand rule, the reader can observe that the normal directions of each facet oppose one another. Therefore, whichever side facet  $i$  has free, facet  $j$  must have the opposing side free. In the second case B), each facet has an opposing definition for the common edge. The normal direction of these facets is the same, therefore, whichever side  $i$  has free,  $j$  has the same side free.

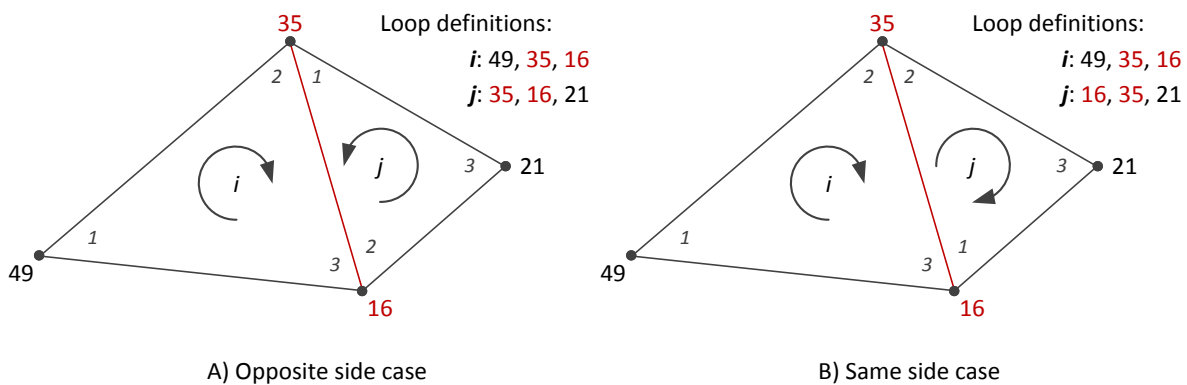


Figure 9-33 – Facet siding propagation rule

Figure 9-34 shows the procedural flow diagram of *SideWalker*.

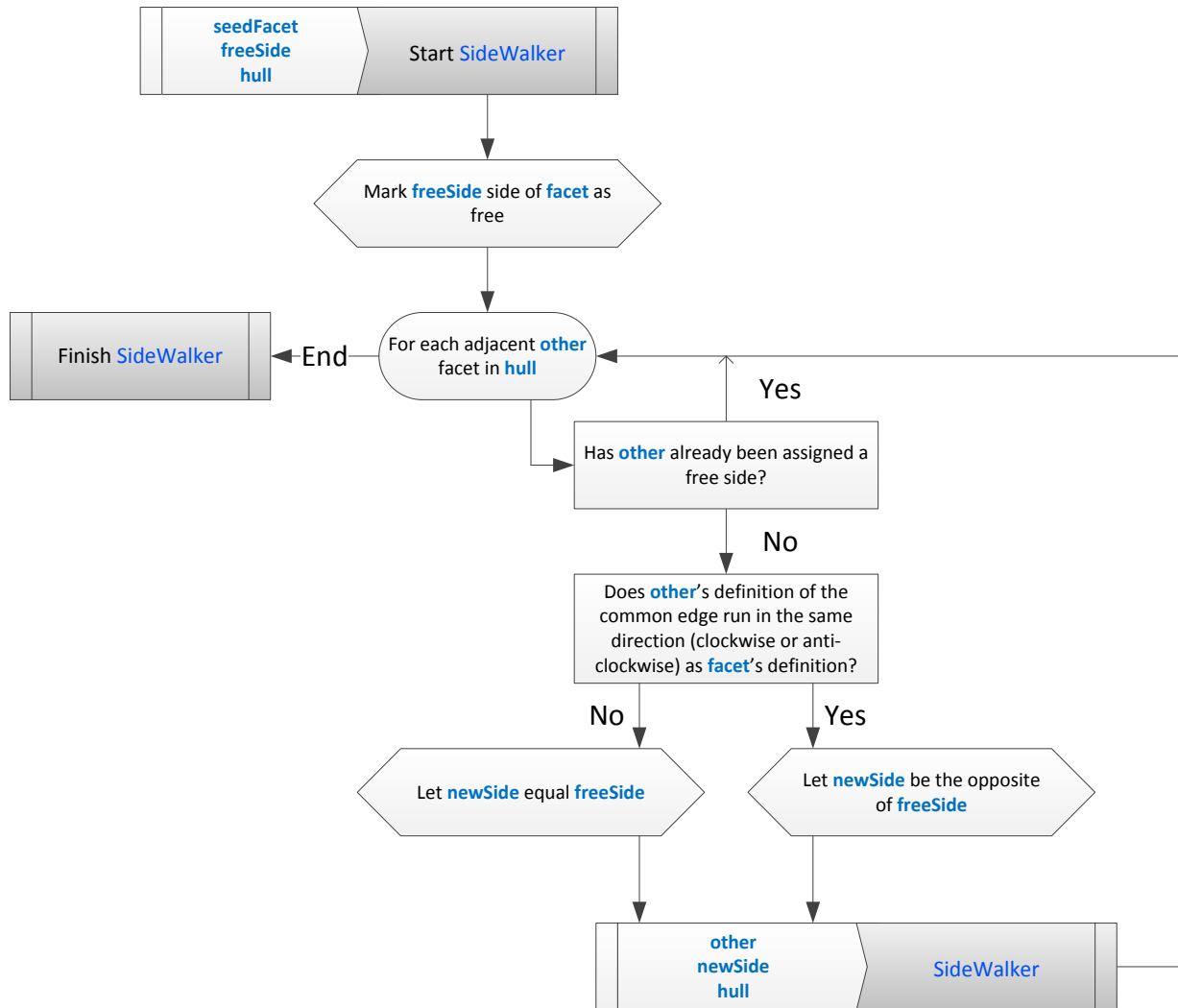


Figure 9-34 – *SideWalker* procedure diagram

### 9.7.11 3D Volume Meshing

After identification, simplification, optimisation, siding of the facets and the definition of certain constraints (such as the illegal segments list) the facet list is passed to *TetMesh*. *TetMesh* is a very complex function based on the gift wrapping tetrahedral meshing algorithm. *TetMesh* is discussed in detail in section 9.9.2.

*TetMesh* returns a list of tetrahedrals and points. These lists are parsed by *SplitMesh*, such that all active tetrahedrals returned are appended to a newElements list as *Element* type objects. Likewise, all points referenced by at least one active tetrahedral are added to a newNodes list as *Node* type objects.

*SplitMesh* delays appending these new *Elements* and *Nodes* to elementIndex and nodeIndex until all *Cutters* have been processed. This is because doing so would require the reallocation of nodeIndex and elementIndex. *Node* types contain an allocatable array of element IDs that are associated with the *Node*. Reallocation of the nodeIndex causes all allocatable arrays within *Nodes* to become unallocated. The elements at node caches are used extensively by *SplitMesh* and functions *SplitMesh* calls. To update the nodeIndex after every *Cutter* would require updating these internal caches, adding unnecessary

computational burden. It is safe to delay adding new **Elements** and **Nodes** to the master indexes because of the assumption that if a **Node** or **Element** is near one **Cutter**, it is not near another.

### 9.7.12 Force Recovery and Allocation

In Chapter Seven, cutting forces at different depths of cut were measured directly using a force dynamometer. The data gathered was used to produce a cutting force prediction model that allows cutter rake and feed force to be extracted from a known depth of cut and width of cut. Interestingly, that chapter also found that velocity has very little effect on cutting forces within the range of spindle speeds and depths of cut used during the real cutting operation. This section deals with implementing the force model (developed in Chapter Seven) in code for use in the simulation.

Figure 9-35, A) below shows a set of intersection **Rays** overlaid in purple on the cutter geometry. These intersection **Rays** are the same **Rays** generated earlier during mesh splitting. All **Rays** are projected to the **Cutter's** local Feed-Radial plane as shown in red, in B).

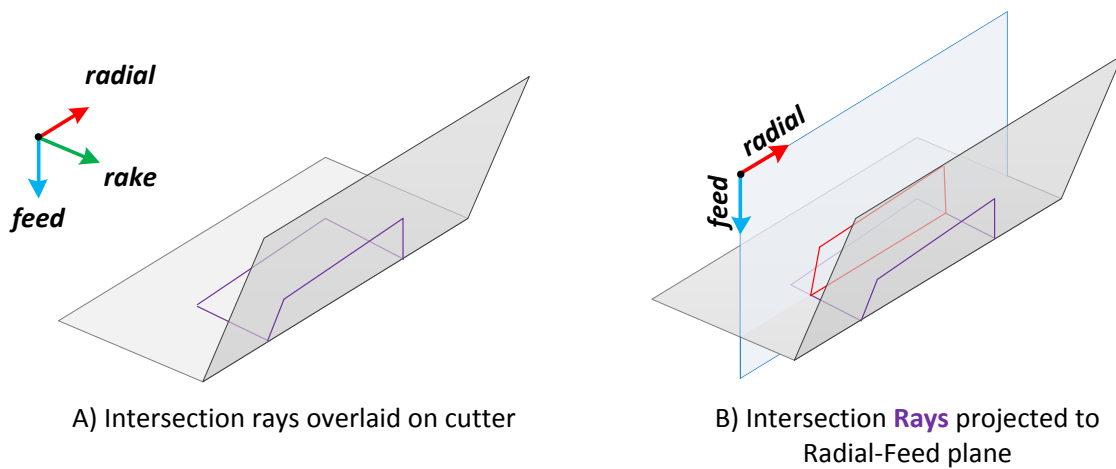


Figure 9-35 – Intersection ray projection for force recovery

Projection works by updating the split **Ray** boundary coordinates with their equivalent radial and feed components, relative to the **Cutter's** local coordinate system. Figure 9-36 shows an example local coordinate system for a **Cutter**. The red, green and blue vectors denote the **Cutter's** local radial, rake and feed coordinate system.  $p$  is some **Ray** boundary point in the global coordinate system that must be expressed in terms of the **Cutter's** local rake-feed plane,  $R$  and  $F$ . Projection this results in a vector to  $p'$ .

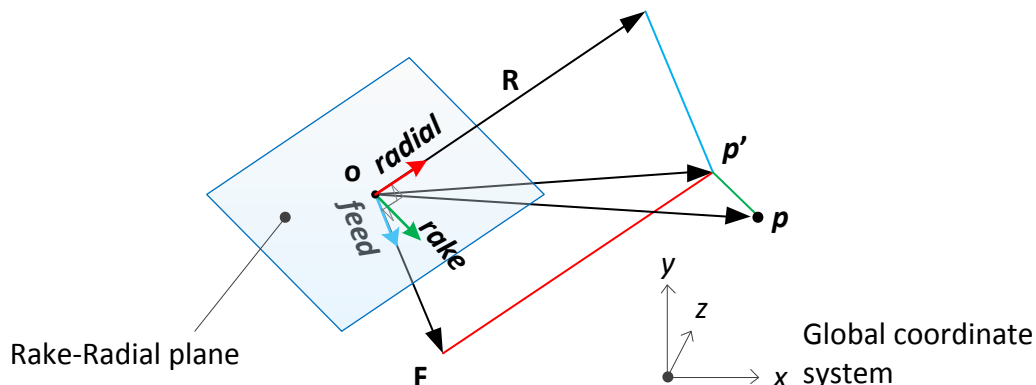


Figure 9-36 – Intersection point projection in terms of local feed-radial plane



First, the vector,  $V_{op}$  from the origin,  $o$ , to  $p$  is calculated according to equation 9-9.

$$V_{op} = p - o \quad (9-9)$$

Next, the radial and feed components,  $R$  and  $F$ , are calculated according to equations 9-10 and 9-11 respectively, where  $\hat{r}$  and  $\hat{f}$  are the radial and feed unit direction vectors respectively.

$$R = \hat{r} \cdot V_{op} \quad (9-10)$$

$$F = \hat{f} \cdot V_{op} \quad (9-11)$$

Figure 9-37 shows the result of projection to the cutters local radial-feed plane. The area below the projected intersection **Rays** is subdivided into vertical strips. The width,  $w$ , and height,  $h$ , of each strip is fed into the constitutive equations shown in equations 9-12 and 9-13 to recover the rake and feed forces.

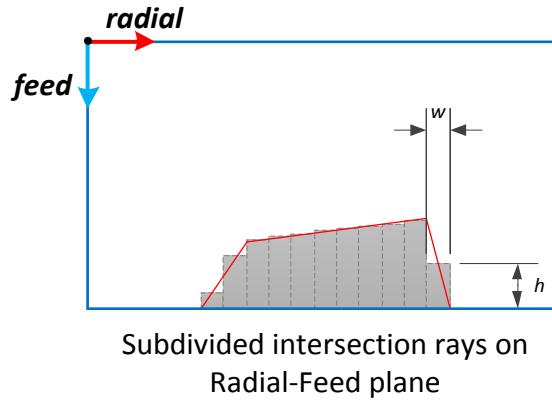


Figure 9-37 – Discretised strips

$$F_r = w \times (1.758E^8 \times h^{7.103E^{-1}}) \quad (9-12)$$

$$F_f = w \times (1.004E^7 \times h^{4.228E^{-1}}) \quad (9-13)$$

Equations 9-12 and 9-13, for the rake and feed cutting loads are specific to the cutting system characterised in Chapter Seven. These equations are passed to the Fortran program via the configuration file lines shown as follows.

```
Fr = v1 * (1.758E+08 * v2^7.103E-01)
Ff = v1 * (1.004E+07 * v2^4.228E-01)
```

These textual representations of equations 9-12 and 9-13 respectively are compiled to a list of instructions using the **Compile** function once per simulation. The compiled equations are evaluated during simulation by the **Evaluate** function. The keywords **v1** and **v2** are special designations interpreted by **Evaluate**. Whenever a call to **Evaluate** is made, **v1** and **v2** are substituted with width,  $w$ , and height,  $h$ , respectively, as defined in figure 9-37.

The sum of forces for each strip is computed and stored in the first three degrees of freedom in the  $dp$  property of the **Cutter**. This value is later recovered from the post file when checking the cutter loads.

To calculate the **Cutter** moment, the centre of area of each strip, shown in figure 9-38 A), is projected to the front face of the **Cutter** by generating a **Ray** that originates from the centre of area and travels in the direction of the cutters affine transform (provided by Marc to describe the change in position of the **Cutter**).

Each infinite **Ray** is tested against the front face of the **Cutter** using the **NURBSurface%RayIntersect** method described in section 9.8.10. A vector is created from the **Cutter's** control node to the intersection point, as shown in blue, in B).

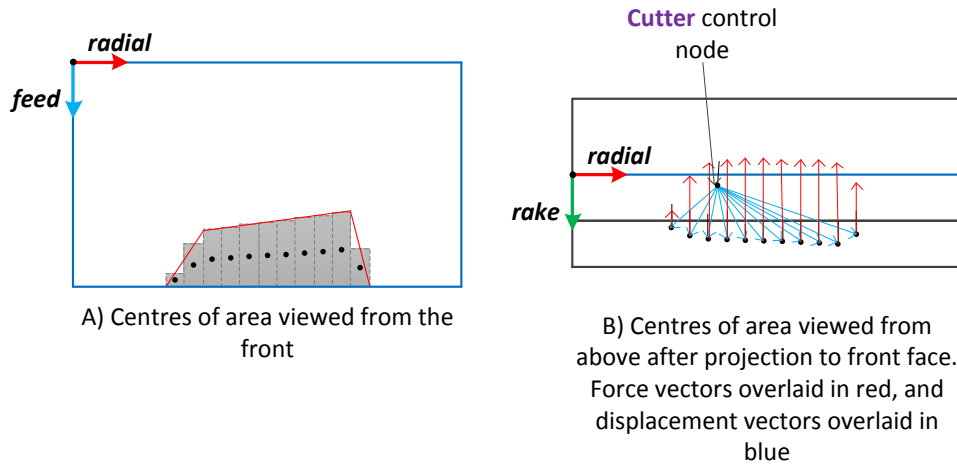


Figure 9-38 – Centres of area for calculating cutter moment

The cross products of the force vectors calculated in the previous step, and shown in red in figure 9-38 B) and the displacement vectors is taken for each strip. The sum of which is applied to degrees of freedom 4 through 6 in the **dp** property of the **Cutter**.

Next, force recovery must assign a reaction load to the workpiece elements. Fortunately, this is a much simpler procedure. Loads are assigned to elements via Marc's **FORCEM** user subroutine. This subroutine assigns element loads based on a pressure and unit direction vector.

The load is divided based on the crude approximation that all cut face elements will more or less share the cutting load. This is of course not exactly how workpiece loads are applied in reality. However, it is regarded as a fair simplification since, the main loads of interest are those on the cutter and, the workpiece does not undergo any significant deformation other than the removal of volume.

The pressure and direction vector must be calculated. The first step is to sum the volume of all **F\_CutFace** elements,  $V_{total}$ , according to the formulation given in section 9.8.1. Pressure,  $P$ , is thus resolved from equation 9-14, where  $\mathbf{F}$  is the **Cutter** force vector (degrees of freedom 1 through 3 in **Cutter%dp**).

$$P = \frac{|\mathbf{F}|}{V_{total}} \quad (9-14)$$

The unit direction vector is simply  $\hat{\mathbf{F}}$ .  $P$  and  $\hat{\mathbf{F}}$  for a given cutter are stored on all **F\_CutFace** elements near that cutter. These loads are distributed later when Marc calls **FORCEM**.

## 9.8 Low-Level Functions and Subroutines

This section describes the mathematical basis underpinning the bespoke functions and subroutines written to solve certain mathematical problems encountered during the higher level functions of the wider Fortran program.

### 9.8.1 Volume of a Tetrahedron

Tetrahedrons are the only simplex used to represent the volume of material. The difference between the volume of the pre-cut and post-cut mesh is used to calculate the volume of material removed, and the elemental cutting pressures.

The volume,  $V$ , of an  $n$ -dimension simplex, with vertices  $(v_0, v_1 \dots v_n)$  can be recovered from the Cayley-Menger determinant as shown in equation 9-15, where  $d_{ij}$  is the distance between vertices  $i$  and  $j$ . A proof for this formulation is given in literature (Blumenthal, 1970).

$$V^2 = \frac{(-1)^{n-1}}{n!^2 2^n} \det \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & d_{01}^2 & d_{02}^2 & \dots & d_{0n}^2 \\ 1 & d_{10}^2 & 0 & d_{12}^2 & \dots & d_{1n}^2 \\ 1 & d_{20}^2 & d_{21}^2 & 0 & \dots & d_{2n}^2 \\ 1 & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & d_{n0}^2 & d_{n1}^2 & d_{n2}^2 & \dots & 0 \end{bmatrix} \quad (9-15)$$

Figure 9-39 shows a schematic of a Marc type 157 element, represented by four nodal points,  $p_1$ ,  $p_2$ ,  $p_3$  and  $p_4$ , where  $d_{ij}$  is the distance between node points  $i$  and  $j$ . Equation 9-15 can be used to derive equation 9-16, from which the volume of the element,  $V$ , can be recovered.

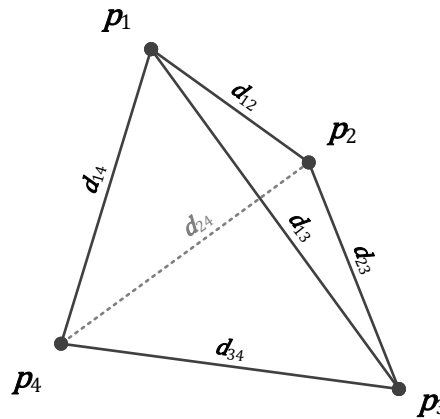


Figure 9-39 – Tetrahedral element

$$288 V^2 = \det \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & d_{12}^2 & d_{13}^2 & d_{14}^2 \\ 1 & d_{21}^2 & 0 & d_{23}^2 & d_{24}^2 \\ 1 & d_{31}^2 & d_{32}^2 & 0 & d_{34}^2 \\ 1 & d_{41}^2 & d_{42}^2 & d_{43}^2 & 0 \end{bmatrix} \quad (9-16)$$

### 9.8.2 Ray Construction

**Ray** objects can be either infinite (mathematical rays) or finite (mathematical segments). They contain properties **Ray%origin**, and **Ray%unitDirection**. For finite **Rays**, a **Ray%d** value is also prescribed that describes the distance along the **Ray** to some significant point. **Rays** are typically constructed from a call to **Ray%Make** which takes a start point and end point (or any point defining the direction). **Ray** parameters are defined according to equations 9-17, 9-18 and 9-19, where  $\mathbf{p}_1$  is the start point,  $\mathbf{p}_2$  is the end point,  $\mathbf{o}$  is the origin of the **Ray**,  $d$  is the distance along the direction vector and  $\hat{\mathbf{D}}$  is the direction vector (with magnitude 1).

$$\mathbf{V}_{12} = \mathbf{p}_2 - \mathbf{p}_1 = \mathbf{p}_2 - \mathbf{o} \quad (9-17)$$

$$d = |\mathbf{V}_{12}| \quad (9-18)$$

$$\hat{\mathbf{D}} = \frac{\mathbf{V}_{12}}{d} \quad (9-19)$$

### 9.8.3 NURBS Surface Formula (**NURBSurface%S**, **CDBR**)

Rigid body surfaces in Marc are represented by Non-Uniform Rational B-Spline Surfaces (NURBS Surfaces). NURBS Surfaces are a two dimensional extension of NURBS curves. Their strength lies in their ability to represent smooth and varied surface curvature, but they can easily be used to represent flat surfaces.

A NURBS surface is defined by a series of weighted control points and knot vectors. Control points can be thought of as magnets that attract the path of the NURBS, knot vectors determine the limits of effect for each control point and weights control the intensity of the attraction.

Like general NURBS curves, the coordinates of a given point on a NURBS surface are calculated using basis functions. Basis functions can have an order depending on the required complexity of curves or surfaces.

Points on the NURBS surface are calculated by **NURBSurface%S**, which is based on equation 9-20, where  $u$  and  $v$  are parametric points on a grid that fits within the upper and lower bounds of the knot vectors  $\mathbf{k}_u$  and  $\mathbf{k}_v$ , for the  $u$  and  $v$  axes respectively.  $B$  is the basis function,  $w$  is the weight for the homogeneous control point coordinate,  $\mathbf{p}$ .  $d_u$  and  $d_v$  are the degree in the  $u$  and  $v$  axes respectively.

$$S_{(u,v)} = \frac{\sum_{i=1}^m \sum_{j=1}^n B_{i,d_u,k_u}(u) B_{j,d_v,k_v}(v) w_{i,j} \mathbf{p}_{i,j}}{\sum_{i=1}^m \sum_{j=1}^n B_{i,d_u,k_u}(u) B_{j,d_v,k_v}(v) w_{i,j}} \quad (9-20)$$

The basis functions are blended using equations 9-21 and 9-22 which are slightly adapted versions of the Cox-de Boor recursion formula. These functions are processed by the **CDBR** function.

$$B_{i,d,k}(x) = \frac{x - k_i}{k_{i+d} - k_i} B_{i,d-1,k}(x) + \frac{k_{i+d+1} - x}{k_{i+d+1} - k_{i+1}} B_{i+1,d-1,k}(x) \quad (9-21)$$

$$B_{i,d=0,k}(x) = \begin{cases} 1: & k_i \leq x < k_{i+1} \\ 0: & \text{otherwise} \end{cases} \quad (9-22)$$

Since there is significant overlap within the [CDBR](#) recursion function, a cache is used to store the results of the recursion function. Whenever the recursion function is called, the cache is checked, if a value is present, that value is returned, otherwise the recursion function will evaluate as normal and record its result in the cache before returning a value.

#### 9.8.4 Reverse NURBS Surface Formula ([NURBSurface%GetUV](#))

Occasionally the parametric coordinates of a known real point on a NURBS surface are required, for example, when meshing the NURBS surfaces of cutters during hull splitting in [SplitMesh](#).

As can be seen in the previous section, the recursive nature of the NURBS equations mean that parametric coordinates are not easily recovered from real coordinates. A converging trial and error based method is used in this work. The method used converges extremely quickly (one increment for flat NURBS) due to the proportional approach used to adjust the parametric coordinate in search of the real coordinate.

In this section, a bar is used to differentiate parametric coordinates from their equivalent real coordinates, equation 9-23 shows an example, where  $S$  is the NURBS surface function described in the previous section.

$$\mathbf{p} = S(\bar{\mathbf{p}}) \quad (9-23)$$

Let  $\mathbf{p}$  be some real coordinate known to exist approximately on the surface of a NURBS surface. The parametric coordinate,  $\bar{\mathbf{p}}$ , is to be found, such that when fed into the NURBS Equation described in section 9.8.3 previously, yield a coordinate,  $\mathbf{p}'$  which is less than SWEEP\_NURBS away from  $\mathbf{p}$ .

Let  $\mathbf{N}$  be a set of real points on the NURBS surface generated from a 2 X 2 array of parametric coordinates,  $\bar{\mathbf{N}}$ , between the maximum and minimum  $u$  and  $v$  values, using the recommended step size prescribed by Marc. This step size changes depending on the complexity of the NURBS based on some internal formula within Marc.

Let  $\mathbf{n} \in \mathbf{N}$ , with equivalent parametric coordinate  $\bar{\mathbf{n}} \in \bar{\mathbf{N}}$ , be the closest coordinate to  $\mathbf{p}$ . Let  $\mathbf{V}_{np}$  be the real vector from  $\mathbf{n}$  to  $\mathbf{p}$ . Let  $\bar{\mathbf{V}}_{\delta u}$  and  $\bar{\mathbf{V}}_{\delta v}$  be [rays](#), both centred around  $\bar{\mathbf{n}}$ , representing some small perturbation in the parametric  $u$  and  $v$  axes respectively. Figure 9-40 below, shows  $\mathbf{V}_{\delta u}$  and  $\mathbf{V}_{\delta v}$  after conversion to real coordinates.

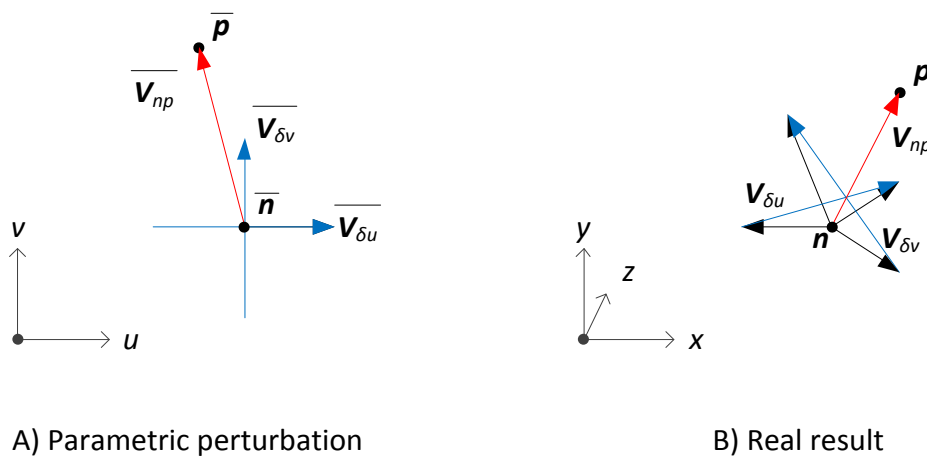


Figure 9-40 – Parametric perturbation and real result

In the figure, the unknown vector  $\overline{V_{np}}$  can be expressed in terms of a parametric error,  $d_u$  and  $d_v$  as shown in equation 9-24.

$$\overline{V_{np}} = d_u \overline{V_{\delta u}} + d_v \overline{V_{\delta v}} \quad (9-24)$$

Likewise, the real equivalent is defined as shown in equation 9-25, where  $e$  is some very small error introduced to make the equivalence solvable in the real domain and  $V_{norm}$  is the cross product of real vectors as shown in equation 9-26.

$$V_{np} = d_u V_{\delta u} + d_v V_{\delta v} + e \hat{V}_{norm} \quad (9-25)$$

$$V_{norm} = V_{\delta u} \times V_{\delta v} \quad (9-26)$$

This relationship is shown in figure 9-41.

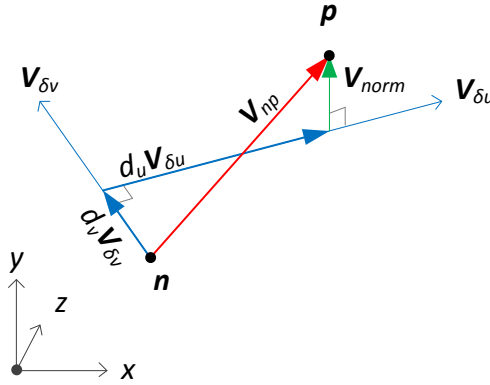


Figure 9-41 – Real vectors to  $p$  through  $V_{\delta u}$ ,  $V_{\delta v}$  and  $V_{norm}$

Equation 9-14 can be expanded to form the matrix equation shown in equation 9-27.

$$\begin{bmatrix} d_u \\ d_v \\ e \end{bmatrix} = \begin{bmatrix} V_{np,x} \\ V_{np,y} \\ V_{np,z} \end{bmatrix} \begin{bmatrix} V_{\delta u,x} & V_{\delta v,x} & \hat{V}_{norm,x} \\ V_{\delta u,y} & V_{\delta v,y} & \hat{V}_{norm,y} \\ V_{\delta u,z} & V_{\delta v,z} & \hat{V}_{norm,z} \end{bmatrix}^{-1} \quad (9-27)$$

After solving equation 9-27,  $d_u$  and  $d_v$  can be fed into equation 9-24 to recover  $\overline{V_{np}}$  which can be used to determine  $\overline{p}$  from  $\overline{n}$ , and thus  $p'$  by using the NURBS surface equation presented in the previous section. The convergence error is computed according to equation 9-28.

$$error = |p - p'| \quad (9-28)$$

If  $error$  is less than `SWEEP_NURBS`, the function returns  $\overline{p}$ , else the recursive portion of the routine is repeated up to twenty times, after which the code will raise a divergence exception, and execution will cease. Each time the routine repeats,  $\overline{n}$  takes the value of  $\overline{p}$ .

### 9.8.5 Circumscribed Triangle in R3 (Circumcircle)

During 2D meshing, it is necessary to determine whether or not a triangle in R3 is Delaunay or not. A Delaunay triangle is a triangle whose circumcentre contains no other points in the mesh. A strongly-Delaunay triangle only has its own points on the circumcircle radius and a weakly Delaunay triangle may contain other mesh points on the circumcircle radius.

A circumscribed triangle, as shown in figure 9-42, is characterised by having all points at an equal radius from the circumcentre. All un-collapsed triangles have a circumcentre and radius.

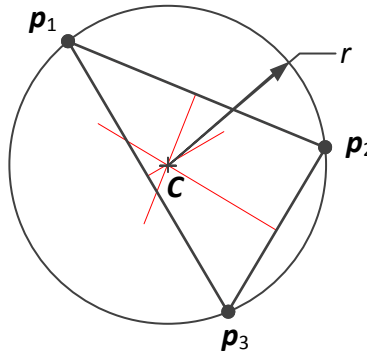


Figure 9-42 – Circumscribed triangle

Calculating the circumcentre and radius of a triangle in R3 is a straightforward process. Firstly the normal vector,  $\mathbf{n}$ , of the plane containing the triangle is determined from the cross product of edge vectors  $\mathbf{e}_{12}$  and  $\mathbf{e}_{13}$ , as shown in equation 9-31, where the edge vectors are given in equations 9-29 and 9-30, where  $\mathbf{p}_1$ ,  $\mathbf{p}_2$  &  $\mathbf{p}_3$  are the triangle corner points.

$$\mathbf{e}_{12} = \mathbf{p}_2 - \mathbf{p}_1 \quad (9-29)$$

$$\mathbf{e}_{13} = \mathbf{p}_3 - \mathbf{p}_1 \quad (9-30)$$

$$\mathbf{n} = \mathbf{e}_{12} \times \mathbf{e}_{13} \quad (9-31)$$

Next, the vector to the centre of the circumcircle from  $\mathbf{p}_1$ ,  $\mathbf{V}_{1C}$ , can be computed from equation 9-32.

$$\mathbf{V}_{1C} = \frac{(\mathbf{n} \times \mathbf{e}_{12})|\mathbf{e}_{13}|^2 \times (\mathbf{e}_{13} \times \mathbf{n})|\mathbf{e}_{12}|^2}{2|\mathbf{n}|^2} \quad (9-32)$$

Finally the centre coordinate,  $\mathbf{C}$ , and radius,  $r$ , can be calculated from equations 9-33 and 9-34 respectively.

$$\mathbf{C} = \mathbf{p}_1 + \mathbf{V}_{1C} \quad (9-33)$$

$$r = |\mathbf{V}_{1C}| \quad (9-34)$$

### 9.8.6 Circumscribed Tetrahedron (Circumsphere)

Calculating the centre coordinate and radius of a circumscribed tetrahedron is a little more challenging than its triangular counterpart. The method used in this work is based on that described by Weisstein, 2017, where the centre,  $\mathbf{C}$ , of a circumscribed tetrahedron can be found from equation 9-35, where  $\mathbf{p}_1$ ,  $\mathbf{p}_2$ ,  $\mathbf{p}_3$  and  $\mathbf{p}_4$  are the tetrahedron vertices.

$$\begin{vmatrix} C_x^2 + C_y^2 + C_z^2 & C_x & C_y & C_z & 1 \\ p_{1x}^2 + p_{1y}^2 + p_{1z}^2 & p_{1x} & p_{1y} & p_{1z} & 1 \\ p_{2x}^2 + p_{2y}^2 + p_{2z}^2 & p_{2x} & p_{2y} & p_{2z} & 1 \\ p_{3x}^2 + p_{3y}^2 + p_{3z}^2 & p_{3x} & p_{3y} & p_{3z} & 1 \\ p_{4x}^2 + p_{4y}^2 + p_{4z}^2 & p_{4x} & p_{4y} & p_{4z} & 1 \end{vmatrix} = 0 \quad (9-35)$$

Solving for  $\mathbf{C}$  begins by expanding the determinate according to the form given in equation 9-36.

$$a(C_x^2 + C_y^2 + C_z^2) - (D_x + D_y + D_z) + c = 0 \quad (9-36)$$

This expansion yields expressions for,  $a$ ,  $D_x$ ,  $D_y$ ,  $D_z$  and  $c$  as shown in equations 9-37, 9-38, 9-39, 9-40 and 9-41.

$$a = \begin{vmatrix} p_{1x} & p_{1y} & p_{1z} & 1 \\ p_{2x} & p_{2y} & p_{2z} & 1 \\ p_{3x} & p_{3y} & p_{3z} & 1 \\ p_{4x} & p_{4y} & p_{4z} & 1 \end{vmatrix} \quad (9-37)$$

$$D_x = \begin{vmatrix} p_{1x}^2 + p_{1y}^2 + p_{1z}^2 & p_{1y} & p_{1z} & 1 \\ p_{2x}^2 + p_{2y}^2 + p_{2z}^2 & p_{2y} & p_{2z} & 1 \\ p_{3x}^2 + p_{3y}^2 + p_{3z}^2 & p_{3y} & p_{3z} & 1 \\ p_{4x}^2 + p_{4y}^2 + p_{4z}^2 & p_{4y} & p_{4z} & 1 \end{vmatrix} \quad (9-38)$$

$$D_y = - \begin{vmatrix} p_{1x}^2 + p_{1y}^2 + p_{1z}^2 & p_{1x} & p_{1z} & 1 \\ p_{2x}^2 + p_{2y}^2 + p_{2z}^2 & p_{2x} & p_{2z} & 1 \\ p_{3x}^2 + p_{3y}^2 + p_{3z}^2 & p_{3x} & p_{3z} & 1 \\ p_{4x}^2 + p_{4y}^2 + p_{4z}^2 & p_{4x} & p_{4z} & 1 \end{vmatrix} \quad (9-39)$$

$$D_z = \begin{vmatrix} p_{1x}^2 + p_{1y}^2 + p_{1z}^2 & p_{1x} & p_{1y} & 1 \\ p_{2x}^2 + p_{2y}^2 + p_{2z}^2 & p_{2x} & p_{2y} & 1 \\ p_{3x}^2 + p_{3y}^2 + p_{3z}^2 & p_{3x} & p_{3y} & 1 \\ p_{4x}^2 + p_{4y}^2 + p_{4z}^2 & p_{4x} & p_{4y} & 1 \end{vmatrix} \quad (9-40)$$

$$c = \begin{vmatrix} p_{1x}^2 + p_{1y}^2 + p_{1z}^2 & p_{1x} & p_{1y} & p_{1z} \\ p_{2x}^2 + p_{2y}^2 + p_{2z}^2 & p_{2x} & p_{2y} & p_{2z} \\ p_{3x}^2 + p_{3y}^2 + p_{3z}^2 & p_{3x} & p_{3y} & p_{3z} \\ p_{4x}^2 + p_{4y}^2 + p_{4z}^2 & p_{4x} & p_{4y} & p_{4z} \end{vmatrix} \quad (9-41)$$



Completing the square yields equation 9-42.

$$a \left( C_x - \frac{D_x}{2a} \right)^2 + a \left( C_y - \frac{D_y}{2a} \right)^2 + a \left( C_z - \frac{D_z}{2a} \right)^2 - \frac{D_x^2 + D_y^2 + D_z^2}{4a} + c = 0 \quad (9-42)$$

Equation 9-42 is the equation of the circumsphere with the form given in equation 9-43.

$$(x - C_x)^2 + (y - C_y)^2 + (z - C_z)^2 = r^2 \quad (9-43)$$

Therefore, the circumcentre coordinates are given in equation 9-44 as follows.

$$C_x = \frac{D_x}{2a}, \quad C_y = \frac{D_y}{2a}, \quad C_z = \frac{D_z}{2a} \quad (9-44)$$

Finally, the radius can be determined from equation 9-45.

$$r = \frac{\sqrt{D_x^2 + D_y^2 + D_z^2 - 4ac}}{2|a|} \quad (9-45)$$

### 9.8.7 Ray-Ray Intersection ([Ray%RayIntersect](#))

The intersection between two [Rays](#) in 3D space is a surprisingly difficult problem compared to its 2D counterpart. This is largely due to the property of 3D space that lines can glance one another forming an intersection in all practical purposes, but differing by some floating point error sufficient to suggest they do not intersect.

Figure 9-43 shows two finite [Rays](#), 1 and 2, in R3. The [Rays](#) make a glancing pass, separated at their mutual closest point by the perpendicular vector,  $\mathbf{V}$ , indicated in green. If the magnitude of  $\mathbf{V}$  is sufficiently small, the infinite [Rays](#) can be said to have intersected.

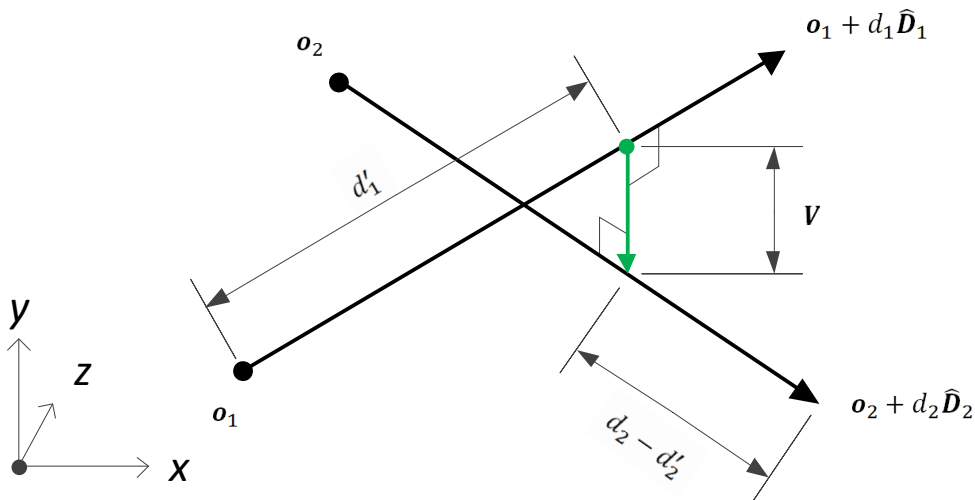


Figure 9-43 – Ray-Ray intersection diagram showing perpendicular vector

If the two **Rays** are parallel, this function terminates early, indicating that there is no single intersection point. For all other cases, the shortest distance between the two **Rays**, is a vector,  $\mathbf{V}$ , that forms a right angle to both **Rays**. The direction of this vector is given by the cross product of the **Ray** direction vectors as shown in equation 9-46.

$$\hat{\mathbf{V}} = \frac{\hat{\mathbf{D}}_1 \times \hat{\mathbf{D}}_2}{|\hat{\mathbf{D}}_1 \times \hat{\mathbf{D}}_2|} \quad (9-46)$$

$\mathbf{V}$  can be expressed in terms of its length, the error,  $e$ , and its unit direction  $\hat{\mathbf{V}}$  as shown in equation 9-47.

$$\mathbf{V} = e\hat{\mathbf{V}} \quad (9-47)$$

In 9-43, it can be observed that the equivalence given in equation 9-48 is true.

$$d'_1\hat{\mathbf{D}}_1 + (d_2 - d'_2)\hat{\mathbf{D}}_2 + e\hat{\mathbf{V}} = \mathbf{o}_2 + d_2\hat{\mathbf{D}}_2 - \mathbf{o}_1 \quad (9-48)$$

Equation 9-48 can be rearranged to give equation 9-49, which can be solved to give the two **Ray** magnitudes corresponding to their respective nearest points to one another, and  $e$ , the error or distance between the two points.

$$\begin{bmatrix} d'_1 \\ d_2 - d'_2 \\ e \end{bmatrix}_{1 \times 3} = \begin{bmatrix} o_{2x} + d_2\hat{D}_{2x} - o_{1x} \\ o_{2y} + d_2\hat{D}_{2y} - o_{1y} \\ o_{2z} + d_2\hat{D}_{2z} - o_{1z} \end{bmatrix}_{1 \times 3} \begin{bmatrix} \hat{D}_{1x} & \hat{D}_{2x} & \hat{V}_x \\ \hat{D}_{1y} & \hat{D}_{2y} & \hat{V}_y \\ \hat{D}_{1z} & \hat{D}_{2z} & \hat{V}_z \end{bmatrix}_{3 \times 3}^{-1} \quad (9-49)$$

If  $|e| > \text{EPSILON}$  (or **SWEEP** in some cases depending on the desired accuracy of the calling function) then the **Rays** do not intersect and the function returns **False**.

Finally the  $d'$  values along each **Ray** can be tested against their limits and the appropriate result returned according to equation 9-50.

$$\begin{cases} \text{Does not intersect:} & \begin{aligned} d'_1 &< -\text{EPSILON} \\ d'_2 &< -\text{EPSILON} \\ d'_1 &> d_1 + \text{EPSILON} \\ d'_2 &> d_2 + \text{EPSILON} \end{aligned} \\ \text{Does intersect:} & \text{otherwise} \end{cases} \quad (9-50)$$

### 9.8.8 Ray-Triangle Intersection (Ray%TriIntersect)

Ray triangle intersection testing is used in a variety of contexts, for example, during visibility testing in TetMesh (9.9.2), NURBSurface%RayIntersect (9.8.10) and facet siding during SplitMesh (9.7.10).

The algorithm developed for this program can be used in several modes. Its principle function is to determine whether or not a finite Ray intersects a triangle and return the intersection coordinate and Ray  $d$  value. Ray%TriIntersect can be made to work in infinite Ray mode and in which case will return True if the Ray is on target, even if that Ray does not pass through the triangle.

Ray%TriIntersect is based on a modified fast intersection detection algorithm developed by Moller and Trumbore, 1998, in which triangular facets are used to represent surfaces. Aside from the high performance of their algorithm, it is not sensitive to the sidedness of the triangle like other algorithms.

Points on a triangle,  $T$ , as a function of the barycentric coordinates  $u$  &  $v$  are given by equation 9-51, where  $p_1$ ,  $p_2$  and  $p_3$  are the corner points of the triangle, and where  $u$  and  $v$ , must satisfy the criteria given in equations 9-52 and 9-53.

$$T(u, v) = (1 - u - v)p_1 + up_2 + vp_3 \quad (9-51)$$

$$u + v \leq 1 \quad (9-52)$$

$$u \geq 0, v \geq 0 \quad (9-53)$$

A Ray is defined by its origin,  $o$ , unit direction,  $\hat{D}$ , and direction magnitude,  $d$ . The intersection point between a Ray and triangle is therefore given by equation 9-54, where  $d'$  is some distance along the infinite Ray.

$$o + d'\hat{D} = (1 - u - v)p_1 + up_2 + vp_3 \quad (9-54)$$

Equation 9-54 can be rearranged to give equation 9-55, Where  $e_1$  &  $e_2$  are the triangle edge vectors given in equations 9-56 and 9-57 respectively.

$$[-\hat{D} \quad e_1 \quad e_2] \begin{bmatrix} d' \\ u \\ v \end{bmatrix} = o - p_1 \quad (9-55)$$

$$e_1 = p_2 - p_1 \quad (9-56)$$

$$e_2 = p_3 - p_1 \quad (9-57)$$

Before this formulation can be applied, the parallel case must be ruled out in which the **Ray** is parallel with the plane of the triangle. This is done by taking the cross product of edge vectors to find a plane normal vector,  $\mathbf{n}$ , as shown in equation 9-58.

$$\mathbf{n} = \mathbf{e}_1 \times \mathbf{e}_2 \quad (9-58)$$

If the **Ray** is parallel to the triangle plane, then the angle between this normal vector and the unit direction,  $\hat{\mathbf{D}}$ , of the **Ray** will be approximately 90 degrees. The angle,  $\alpha$ , is found according to the dot product relationship given in equation 9-59.

$$\cos \alpha = \frac{\mathbf{n} \cdot \hat{\mathbf{D}}}{|\mathbf{n}| |\hat{\mathbf{D}}|} \quad (9-59)$$

The parallel relationship can be established without solving for  $\alpha$ , since as  $\alpha$  approaches 90 degrees,  $\cos \alpha$  approaches zero. Thus, the condition given in equation 9-60 can be used to determine if the **Ray** is parallel to the plane of the triangle.

$$\begin{cases} \text{Possible intersection:} & |\cos \alpha| > EPSILON \\ \text{Parallel (does not intersect):} & \text{otherwise} \end{cases} \quad (9-60)$$

If intersection is possible according to equation 9-60 then the matrix equation 9-55 is solved, yielding  $d$ ,  $u$  and  $v$ . To make sure that the intersection point is within the boundary of the triangle,  $u$  and  $v$  are checked against the criteria given in equations 9-52 and 9-53.

When called in infinite ray mode, **Ray%TriIntersect** terminates at this point and returns the result of the boundary check. However, if intersection of the finite **Ray** is to be determined,  $d'$  must be tested against the criteria given in equations 9-61 and to make sure it lies on the finite **Ray**.

$$\begin{cases} \text{Does not intersect:} & d' < -EPSILON \\ & d' > d + EPSILON \\ \text{Does intersect:} & \text{otherwise} \end{cases} \quad (9-61)$$

### 9.8.9 Triangle-Triangle Intersection (TriTriIntersect)

Triangle-Triangle intersection is used by the `NURBSurface%TriIntersect` intersection algorithm given in section 9.8.11. Triangle-Triangle intersection takes as its inputs, two triangles defined by their corner coordinates, and outputs a logical value, where `True` indicates the two triangles intersect. If an intersection occurs, the function also outputs the finite intersection `Ray`.

Triangle-Triangle intersection detection is performed in two steps, firstly, by finding the infinite intersection `Ray, I`, between the two triangles, and secondly, by generating a collinear finite intersection `Ray, I'`, by finding the boundaries of intersection, limited at each end by the most restrictive triangle.

Before intersection testing, the coplanar case must be ruled out. This can be done efficiently by calculating the angle between the triangle normal vectors,  $\mathbf{n}_1$  and  $\mathbf{n}_2$ , according to equation 9-62, where  $\mathbf{n}_1$  and  $\mathbf{n}_2$  are calculated for triangles 1 and 2 respectively, according to equation 9-63.

$$\cos \alpha = \frac{\mathbf{n}_1 \cdot \mathbf{n}_2}{|\mathbf{n}_1||\mathbf{n}_2|} \quad (9-62)$$

$$\mathbf{n} = \mathbf{e}_1 \times \mathbf{e}_2 \quad (9-63)$$

Edge vectors for each triangle,  $\mathbf{e}_1$  and  $\mathbf{e}_2$  are calculated according to equations 9-64 and 9-65 respectively, where  $\mathbf{p}_1$ ,  $\mathbf{p}_2$  and  $\mathbf{p}_3$  are the corner points.

$$\mathbf{e}_1 = \mathbf{p}_2 - \mathbf{p}_1 \quad (9-64)$$

$$\mathbf{e}_2 = \mathbf{p}_3 - \mathbf{p}_1 \quad (9-65)$$

As  $\alpha$  approaches 90 degrees,  $\cos \alpha$  approaches 0. Thus, the condition given in equation 9-66 can be used to determine if the triangles are coplanar.

$$\begin{cases} \text{Possible intersection:} & |\cos \alpha| > EPSILON \\ \text{Coplanar (no, or infinite intersection):} & \text{otherwise} \end{cases} \quad (9-66)$$

If no finite intersection is possible, the function simply returns `False` at this point. Determining the direction vector  $\mathbf{D}_I$  of the infinite intersection `Ray, I`, is performed by taking the cross product of the normal vectors of each triangle as shown in equation 9-67.

$$\mathbf{D}_I = \mathbf{n}_1 \times \mathbf{n}_2 \quad (9-67)$$

The origin of  $\mathbf{I}$ , (simply any point on  $\mathbf{I}$ ) must be found. This is done by constructing two spur `Rays`,  $\mathbf{S}_1$  and  $\mathbf{S}_2$ , one for each triangle, coplanar with their respective triangles.

The origin of  $\mathbf{S}$  is  $\mathbf{p}_1$  (any known point on the triangle will suffice), and the direction,  $\mathbf{D}_S$  is the cross product of  $\mathbf{D}_I$  and the triangle normal vector,  $\mathbf{n}$ , as shown in equation 9-68.

$$\mathbf{D}_S = \mathbf{D}_I \times \mathbf{n} \quad (9-68)$$

Both spur **Rays** intersect **I**, but do not necessarily intersect one another, however, since both spur **Rays** are perpendicular to **I**, their closest points to one another are both on **I**. Therefore, either point on **I** can be determined according to **Ray%RayIntersect** described in section 9.8.7. By testing an intersection between **S**<sub>1</sub> and **S**<sub>2</sub>, two  $d'_s$  values will be returned. Either  $d'_s$  value can be used to determine an origin for **I**, **o**<sub>I</sub>, for example,  $d_{s1}$  as shown in equation 9-69.

$$\mathbf{o}_I = \mathbf{o}_{S1} + d'_{s1} \hat{\mathbf{D}}_{S1} \quad (9-69)$$

The next phase is to find the finite intersection **Ray**, **I'** from **I**. Figure 9-44 shows some triangle intersection examples. This figure omits the case where one or both triangles do not intersect **I** at all. A) shows two triangles that both intersect **I** in the portions shaded blue. However, there is no overlap between the blue spans, and so no Triangle-Triangle intersection. B) shows an example where **I'** (red) is bounded partially by one triangle and partially by the other, and finally C) shows an example where **I'** is fully bounded by just one triangle.

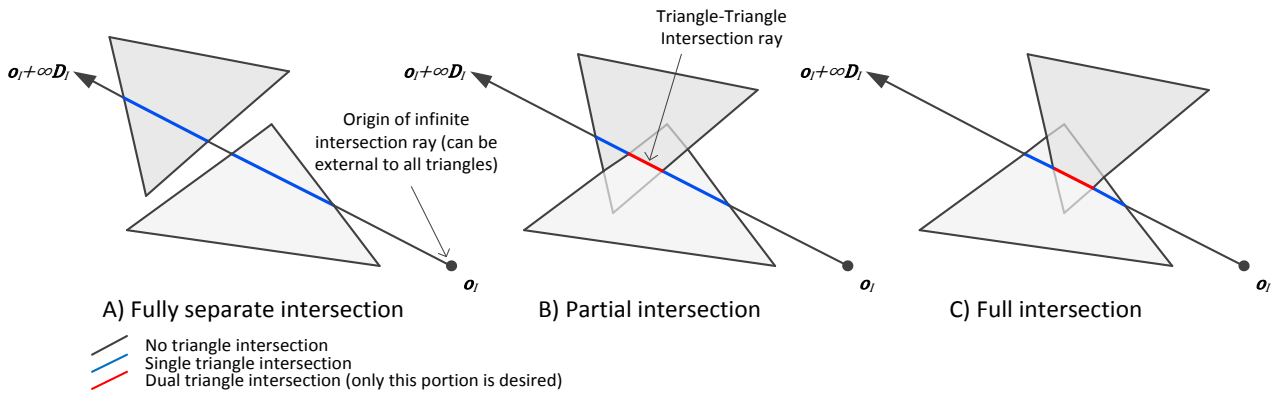


Figure 9-44 – Triangle-Triangle intersection types

Figure 9-45 defines two intersecting triangles, *i* and *j*. The portion of **I** that intersects each triangle is defined by  $d_{I,min}$  and  $d_{I,max}$  for triangles *i* and *j*.

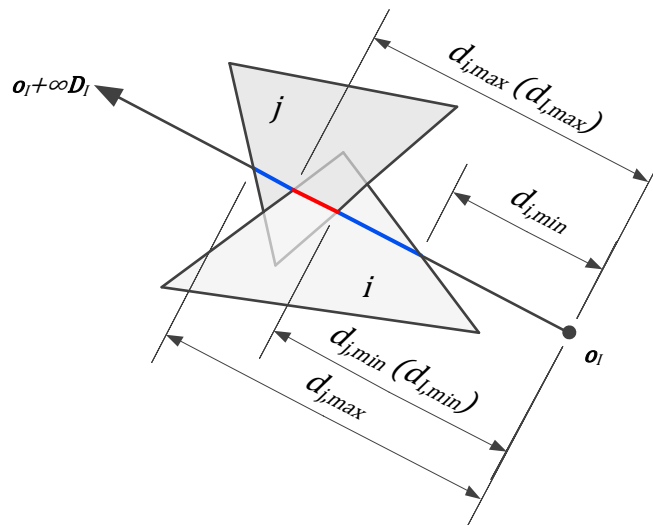


Figure 9-45 – Triangle-Triangle intersection defined as a series of intersection ray *d* values

For each triangle,  $d_{min}$  and  $d_{max}$  can be determined by creating finite **Rays** for each edge on each triangle and testing them against **I** using **Ray%RayIntersect** described earlier in section 9.8.7. For all **Ray** pairs that intersect, a record is kept of the minimum and maximum values of  $d$  along **I**, returned by **Ray%RayIntersect**,  $d_{min}$  and  $d_{max}$  respectively. In all cases, at least one edge **Ray** won't intersect **I**. In some cases, two or three edge **Rays** won't intersect **I**, in those cases there is no triangle-triangle intersection.

Once  $d_{min}$  and  $d_{max}$  value pairs have been found for each triangle, the maximum of minimum values is selected as the final minimum,  $d_{I,min}$ , and likewise, the minimum of maximum values is selected as the final maximum,  $d_{I,max}$ , as shown in equations 9-70 and 9-71 respectively.

$$d_{I,min} = \begin{cases} d_{i,min} & d_{i,min} > d_{j,min} \\ d_{j,min} & otherwise \end{cases} \quad (9-70)$$

$$d_{I,max} = \begin{cases} d_{i,max} & d_{i,max} < d_{j,max} \\ d_{j,max} & otherwise \end{cases} \quad (9-71)$$

Finally, the finite intersection **Ray**, **I'** is created according to equations 9-72, 9-73 and 9-74.

$$\hat{D}_{I'} = \hat{D}_I \quad (9-72)$$

$$\mathbf{o}_{I'} = \mathbf{o}_I + d_{I,min} \times \mathbf{D}_I \quad (9-73)$$

$$d_{I'} = d_{I,max} - d_{I,min} \quad (9-74)$$

The finite intersection **Ray** has a field called `boundFlags(2)`. Offsets one and two in this array refer to the first and second triangles passed to this function respectively. Each offset in `boundFlags` can store any combination of the flags `RF_Default`, `RF_Start`, `RF_End`. `RF_Default` is the default state. For each triangle, `RF_Start` and `RF_End` can be used (and combined) to signal that the **Ray** starts or ends on the respective triangle.

For example, the following code signals that **testRay** has an origin that is on an edge of triangle 1, and an end point on an edge of triangle 2.

```
testRay%boundFlags(1) = RF_Start    !Discrete NURBS triangle
testRay%boundFlags(2) = RF_End      !Mesh facet
```

This information is used in **SplitMesh** to determine which ends of which split **Rays** to use to subdivide hull facets. **SplitMesh** always passes a NURBS triangle in the first argument and a mesh triangle in the second.

#### 9.8.10 Ray-NURBS Intersection (**NURBSSurface%RayIntersect**)

**NURBSSurface%RayIntersect** is an extension of **Ray%TriIntersect**, with some additional features to improve the accuracy of the intersection coordinate.

Figure 9-46 A) shows an intersection between a **Ray** and curved NURBS surface. Determining the intersection coordinate works by first subdividing the NURBS surface into smaller triangles as shown in B). Marc recommends subdivision values based on some unknown internal algorithm. The subdivision values typically increase as the curvature of a NURBS surface increases. These recommended subdivision values

are used to create triangles that approximate the NURBS surface as shown in figure 9-46, B). By default, all the triangle points required for this construction are cached in the `NURBSSurface%surfaceGrid` property.

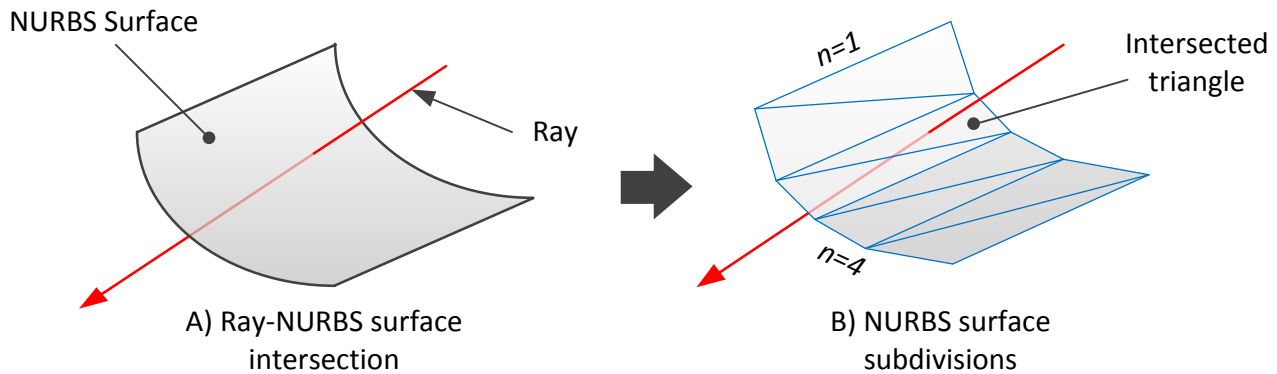


Figure 9-46 – Ray-NURBS surface intersection

Each of these triangles is passed in turn to `Ray%TriIntersect` discussed previously in section 9.8.8. Since `Rays` can penetrate NURBS surfaces multiple times (due to the curved nature of NURBS surfaces), only the smallest  $d'$  value along the intersection test `Ray` is returned.

As with `Ray%TriIntersect`, `NURBSSurface%RayIntersect` also supports infinite `Ray` and finite `Ray` modes. When operating in finite `Ray` mode, this function must correct for discretisation error. Figure 9-47 shows an intersection between a `Ray` and NURBS surface viewed from the side.

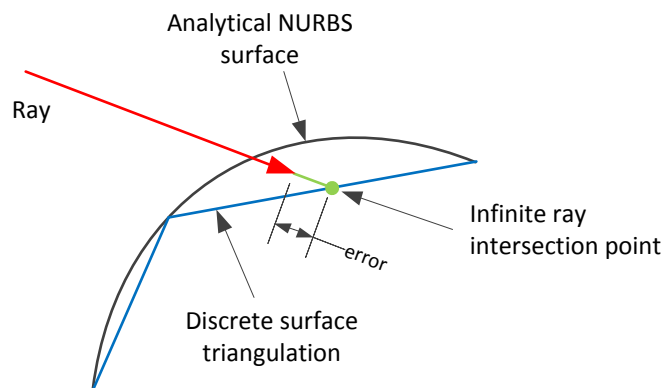


Figure 9-47 – Ray-NURBS intersection discretisation error

As the figure shows, the `Ray` penetrates the NURBS surface but ends somewhere in between the NURBS surface and discrete triangle surface. With no correction, this function would report no intersection, even though one has occurred.

If the error (the distance between the end of the finite `Ray` and the infinite `Ray` intersection point) is less than the maximum edge length of the discrete triangle, the algorithm will proceed to a refinement step.

To perform the refinement, the infinite `Ray` intersection point is mapped to the parametric NURBS surface domain using the `NURBSSurface%GetUV` function described in section 9.8.4. Points are generated around the parametric coordinate, such that four slightly overlapping triangles can be created between them as shown in figure 9-48 A), where the point in red is the parametric infinite `Ray` intersection point.



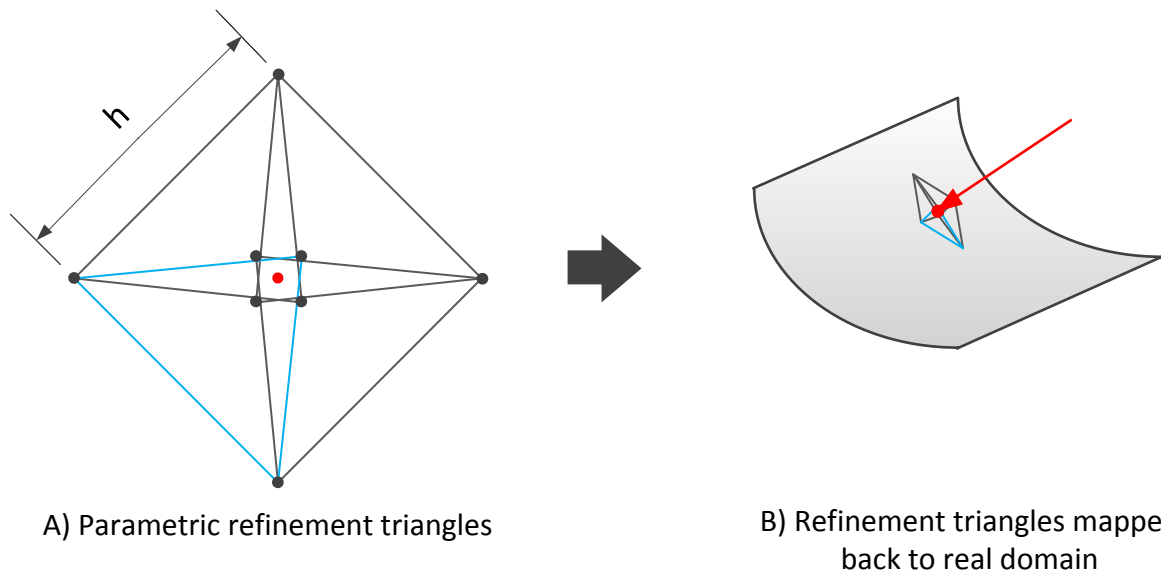


Figure 9-48 – Refinement triangles (one of which highlighted in blue)

These triangles are then mapped back to the real domain as shown in figure 9-48 B), using the NURBS equation described in section 9.8.3. The initial **Ray** is then tested against each of these triangles until a suitable intersection point is found.

If the distance between the new intersection and this intersection is greater than SWEEP\_NURBS, this process is repeated, up to three times, with gradually decreasing triangle sizes.

A demonstration of the refinement algorithm was developed in Python and plotted using Matplotlib as shown in figure 9-49.

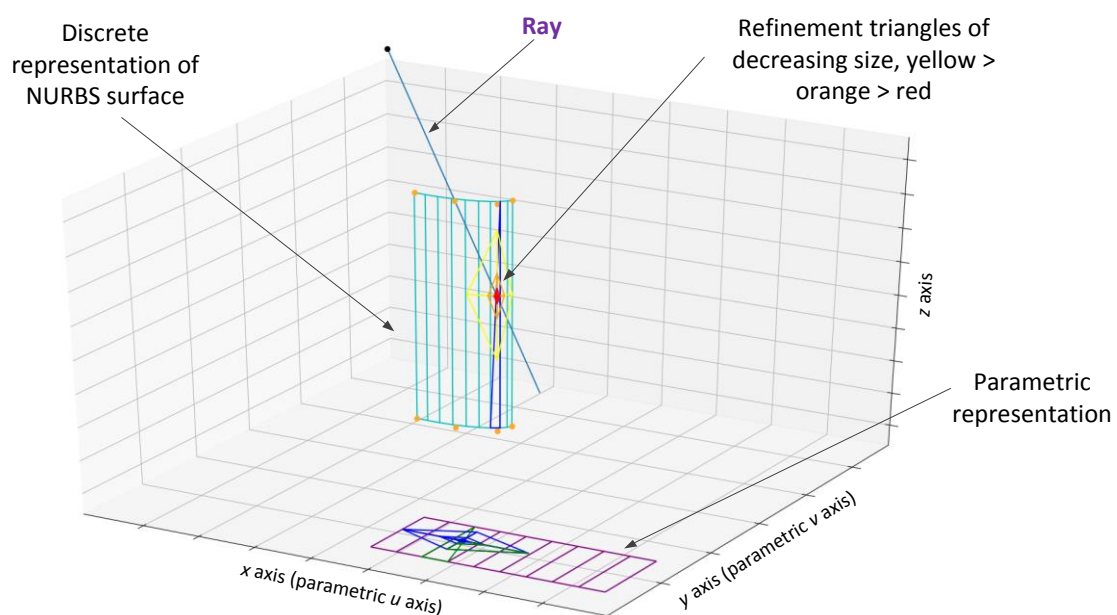


Figure 9-49 – Relationship between parametric NURBS coordinates and real coordinates, as plotted in the same coordinate system

### 9.8.11 Triangle-NURBS Intersection (`NURBSurface%TriIntersect`)

`NURBSurface%TriIntersect` is a simple extension of `TriTriIntersect` that returns a set of **Rays** representing the intersection between a triangle and a NURBS surface. Unlike the `NURBSurface%RayIntersect` method, `NURBSurface%TriIntersect` does not refine the discretisation of the NURBS surface any greater than the base subdivision values recommended by Marc. This choice is made to reduce the number of intersection **Rays** created and to reduce the computational burden of intersection testing.

Figure 9-50 A) shows the intersection of a triangle and NURBS surface. This NURBS surface is discretised according to the recommended subdivision values provided by Marc, as shown in B). Every discrete triangle generated is tested against the test triangle using `TriTriIntersect` described previously in section 9.8.9. If an intersection is found, the intersection **ray** will be added to a list. Once all combinations have been tested, the complete list of intersection **Rays**, shown in red in C) and D), are returned to the calling function.

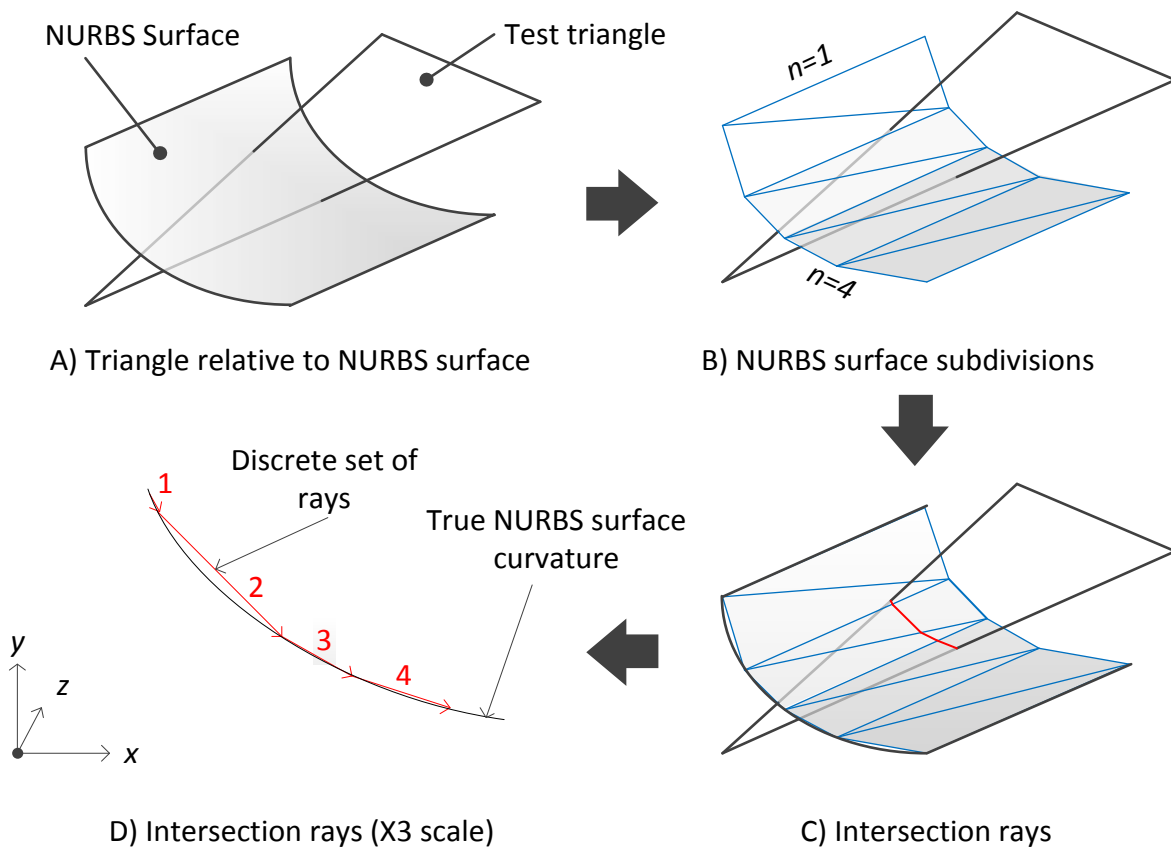


Figure 9-50 – Progression of Triangle-NURBS intersection testing

## 9.9 High-Level Functions and Subroutines

This section describes the higher level subroutines developed for this work. The subroutines discussed in this section are complex and it does not benefit the reader to examine the minutia of how they work exactly, therefore this section focuses on the broad principle of how each function works.

### 9.9.1 Two-Dimensional Meshing ([Giftwrap](#))

[Giftwrap](#), so named after the triangular meshing algorithm of the same name is responsible for generating two dimensional meshes during the hull construction phase after the hull has been split. Two dimensional meshes are required to mesh the region of space occupied by both the cutter and workpiece during an intersection. They are also required to replace intersected external workpiece facets with a facet set that conforms to the cutter geometry.

‘Gift wrapping’ is one of several methods used in literature to create two-dimensional meshes. To understand why this method is used, it is helpful to consider its main alternative, which is the ‘point insertion method’. The gift wrapping method as described by Shewchuk, 2002, and point insertion methods are the most widely discussed methods in literature. Both methods have many variations and optimisations, but broadly speaking, the methods work as follows.

**The point insertion method**, is simple to describe, simple to implement and a reasonably fast executing method to construct a Delaunay triangulation from a set of points. The method starts by constructing a super triangle that would enclose all points.

Points are added to the mesh sequentially. Whenever a point is added, all triangles whose circumcircle encloses the new point are deleted, exposing a perimeter of edges. New triangles are formed between all exposed edges and the newly added point. This process repeats until the points list has been exhausted. The procedure ends by returning all active triangles that do not share a point with the super triangle. Figure 9-51 below shows the procedure diagram for this method.

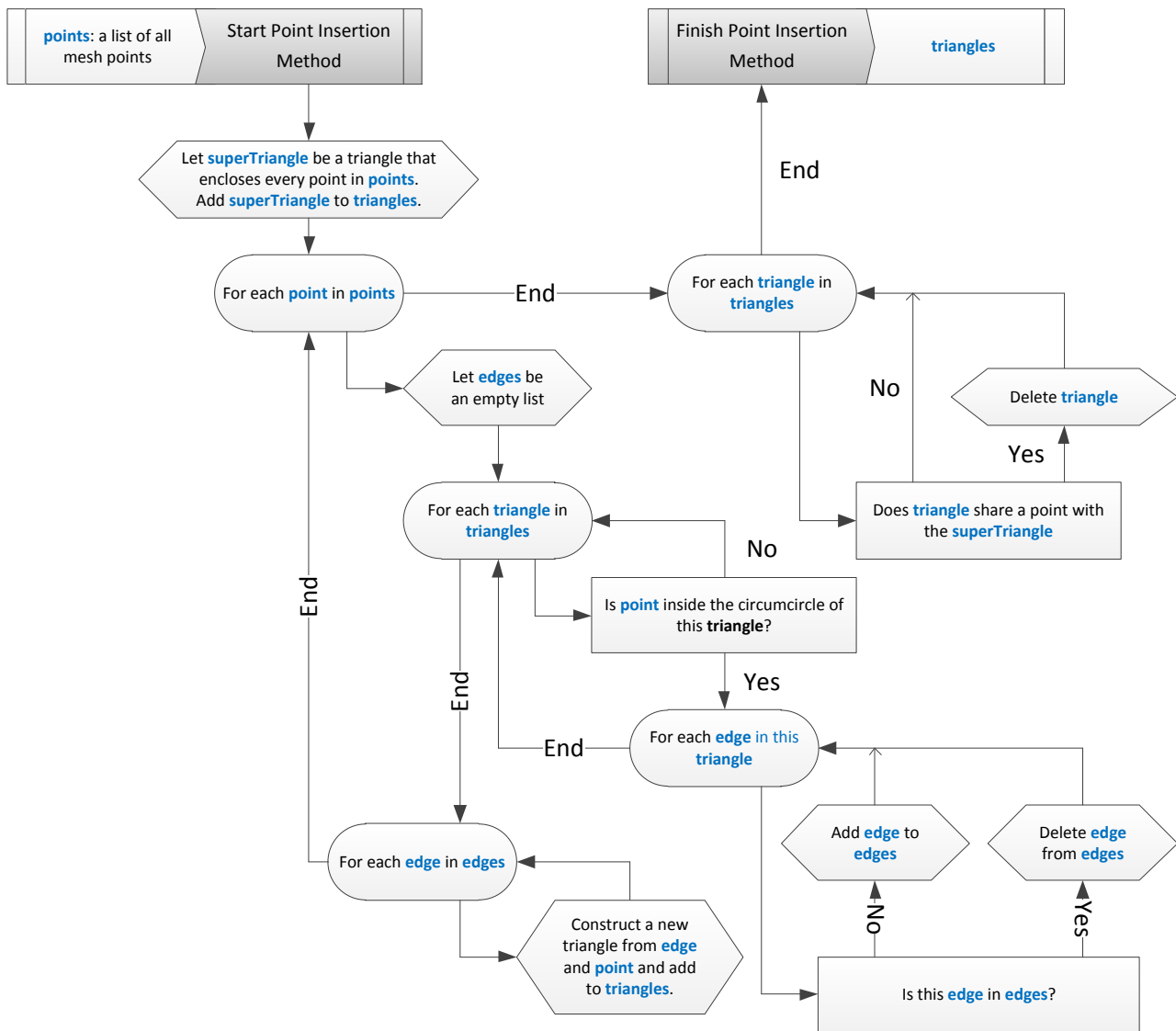


Figure 9-51 – Point insertion method procedure diagram

The major drawback to the point insertion method is its inability to respect segment boundaries that may enclose the mesh. This method will always generate a mesh that encloses the convex hull. Concave hulls will lose any voids and inclusions into the convex perimeter. Although rare, it is possible the meshes required for this work will be concave.

The inability to respect segment boundaries also creates a problem when attempting to generate a mesh that will interface with some other adjacent mesh, as the segments created by the point insertion method are not guaranteed to match those of the adjacent mesh, despite both meshes sharing the same points.

**Gift Wrapping** begins with a list of points to be meshed and a list of edges that contains at least one edge. Each edge joins two points, but no edge can cross any other edge. Generally speaking however, this is an extreme case. In most cases the edge list would contain at least one closed loop representing the perimeter and some internal edges representing geometric features that must be respected.

Each edge has two sides defined according to the common side definition given in section 9.2.5. Sides can be either free or not free. Edges in the external loop have only one free side (facing the inside of the perimeter), whereas internal edges have both sides free.

The end goal of the algorithm is to close every edge by constructing a triangle above its free side using the available points, keeping in mind that whenever a new triangle is added, it may introduce new edges, the sides of which facing away from the triangle are automatically free. Figure 9-52 shows how the gift wrapping algorithm works.

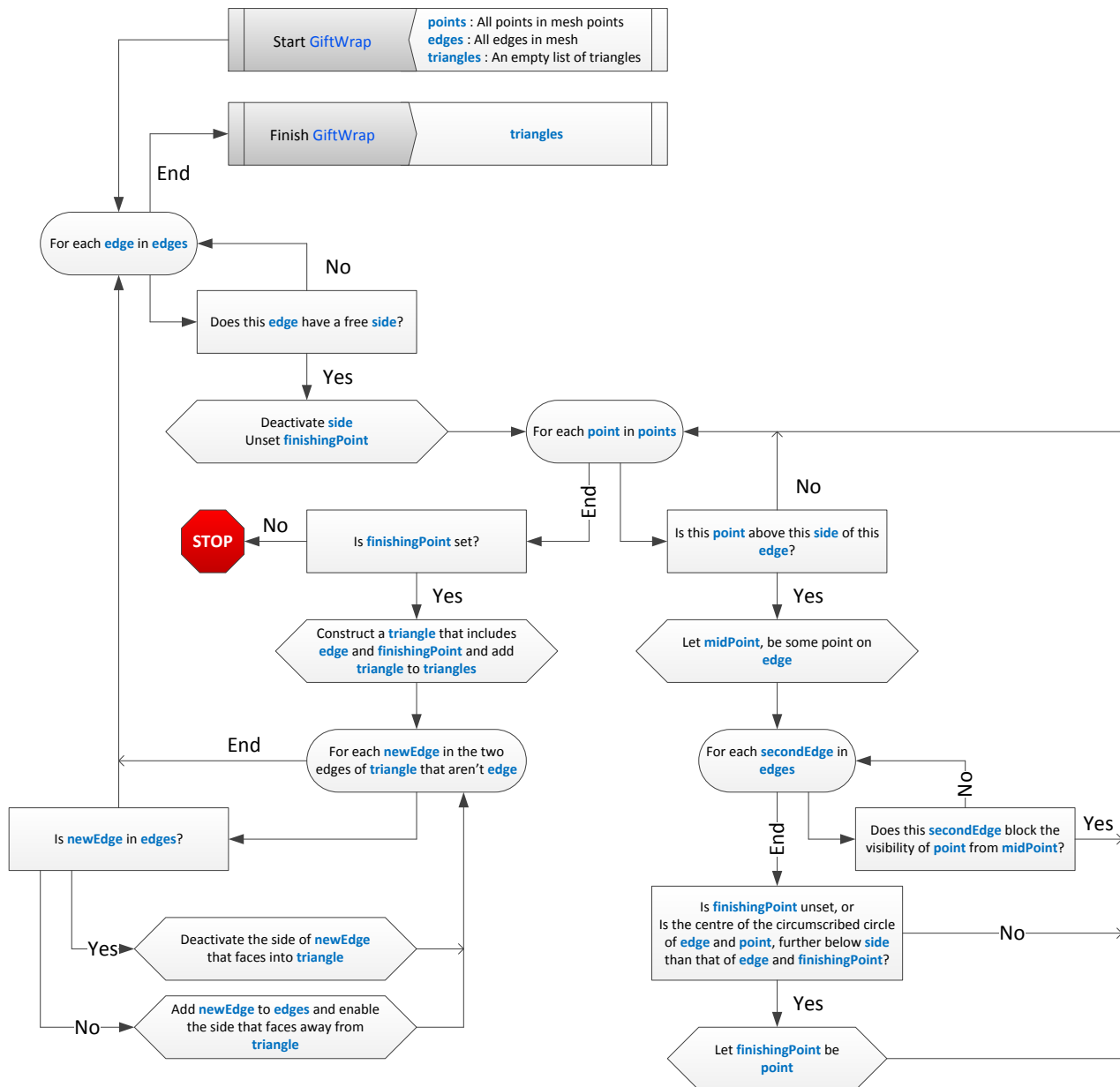


Figure 9-52 – 2D gift-wrapping procedure diagram

Unlike the point insertion method, the gift wrapping method will respect segment boundaries and can be used to generate a concave hull containing voids and inclusions. However, as the diagram shows, the gift wrapping algorithm has a high level of recursion. This is one of the major drawbacks of the algorithm, the price of which must be weighed against the relative simplicity of the algorithm for its ability to make a conforming mesh.

There are of course further caveats to the gift wrapping method, principally, that it does not make for an efficient algorithm. This is because gift wrapping must perform side tests, to ensure potential finishing

points are on the correct side of the edge. Furthermore, an even more expensive visibility test is necessary, to ensure that no pre-existing edges in the mesh block the visibility of the potential finishing point.

### 9.9.2 Three-Dimensional Meshing (TetMesh)

Both the point insertion and gift wrapping methods discussed in the previous section scale neatly to R3, with edges becoming facets and triangles becoming tetrahedrals.

Generating a tetrahedral mesh is never guaranteed. Many factors can cause a tetrahedral mesh to fail, especially when one must be created with a certain number of fixed external facets.

Despite these challenges, TetMesh is ruggedized against potential stumbling blocks and may pursue some drastic options to force a mesh to generate. Generally speaking, TetMesh will prefer to preserve ideal qualities of a mesh (Delaunay property and aspect ratio), but will gradually lower its standards if an ideal mesh cannot be generated. The following descriptions of strategies available to TetMesh are given below alongside their relative frequency of occurrence (observed during development), expressed as a percentage.

**First priority (70%):** A fully Delaunay mesh where all facets and tetrahedrals are Delaunay. No illegal segments, needles or caps are present in the mesh.

**Second priority (20%):** The nature of fixed facets and the order in which tetrahedrals were generated means that not every tetrahedral can be Delaunay. As many Delaunay elements will be created as possible, but some non-Delaunay elements will be considered where there is no choice. No illegal segments, needles or caps are present in the mesh.

**Third priority (disaster recovery, 8%):** A facet exists where no finishing point (that would not create a needle or cap) can be used to complete a tetrahedral on its free side. Disaster recovery traces a Ray through any existing tetrahedrals to an ideal occluded point. All tetrahedrals in the path of the Ray are deleted. Gift wrapping then resumes normally. In disaster recovery, illegal segments that do not mate with the surrounding mesh will be considered (providing their counter segments do not yet exist). This strategy is normally successful, but the result will highly likely contain a mix of Delaunay and non-Delaunay tetrahedrals. There may be some illegal (but not significant) external segments, but no needles or caps.

**Forth priority (panic mode ~1%):** All strategies are allowed including pre-existing tetrahedral destruction, use of illegal segments and freedom to consider a point that would generate a needle or cap. This is a last resort to attempt to force a mesh, but the mesh will definitely contain one or more needles or caps.

**Mesh failure (<1%)** is the next step, which causes the simulation to terminate.

**Unacceptable solutions:** Although failure occurs before unacceptable solutions are considered, there are some very poor options for forcing the mesh to generate beyond failure. These are listed simply for completeness, but no code has been developed to implement these. These options are given as follows:

- relax the side constraints of facets – normally segments are tested for externality to the hull. Relaxing this requirement may allow the mesh to proceed, but external tetrahedrals will form, significantly changing the geometry and volume;
- allow the creation of illegal segments against the surrounding mesh. This method would create a significant amount of flexibility, however, it will generate voids and intersecting volumes at the interface with the surrounding mesh; and

- delete incomplete facets and move on. Mesh failure normally occurs after the vast majority of tetrahedrals have been created, this approach would allow the simulation to continue, but with voids in the mesh.

The gift wrapping algorithm developed for this work is by far the most complex function in the program - not necessarily due to the gift wrapping procedure itself, but more due to all the exceptions and complications that can arise and must be addressed. Figure 9-53 shows a flow diagram of the essential steps of [TetMesh](#).

Due to the complexity of this function, this is not the true layout in code, but this does represent the same effect. As the reader will notice, particularly around *point fitness testing*, this diagram suggests an extremely inefficient design. In reality the function of *point fitness testing* is supported by caches and has satellite code dispersed throughout [TetMesh](#) to reduce the effort required to find suitable finishing points for facets.





## 9.10 Chapter Summary

The aim of this chapter was to document a bespoke Fortran program built to perform mesh splitting, remeshing and force calculation of a machining operation, including implementation of all the associated house keeping functionality required to interface with Marc, and update the simulation.

This chapter presented the technical implementation of that program, including the mathematical basis underpinning the more advanced functions of the program.

Above all, the development of this program has focused on speed. Many simplifications and optimisations have been made to increase simulation speed by reducing the computational work load as much as reasonably possible whilst maintaining the ability to calculate and produce realistic cutting force results. The result is highly parametric, making it easy for the end user to efficiently modify geometry, speeds, feed rates, and other features.

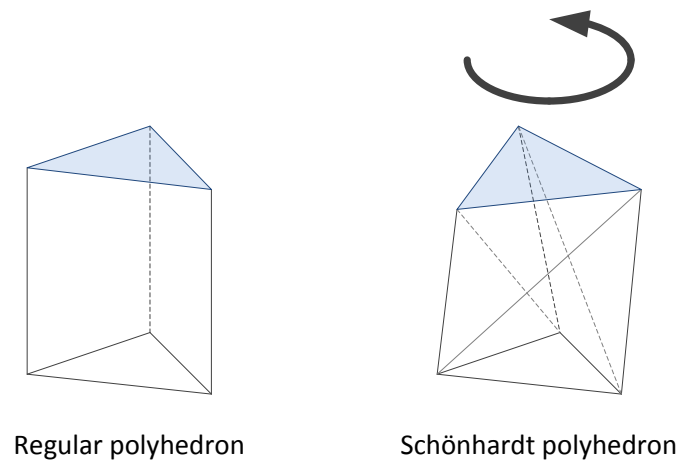
Careful attention has been paid to keep the model expandable to different scenarios including scenarios that involve a flexible spindle, or those where the valve seat is fixed in the cylinder head. Since the spindle is being driven by a user subroutine, it could even be possible to program a real cutting cycle into the model, where the spindle feeds and retracts into several valve seats pressed into the same cylinder head.

Although there are still advancements possible with this model, its implementation is regarded as a success. The next chapter will deal with comparing the model to experimental data, to show if it is capable of producing reasonable results.

In its current state, the tetrahedral mesher developed for this work is not robust enough to have total confidence in any given simulation running to completion. For this reason, the algorithm is only applied to a 30 sub-segment in the next chapter. Building a meshing algorithm to accommodate all possible hulls is extremely difficult. Even Marc's internal advancing front 3D meshing algorithm is not stable in all cases and regularly fails, especially during simulations that include large deformations.

The main issue affecting reliability of the mesher developed in this work is its inability to detect and deal with Schönhardt's polyhedrons. Schönhardt, 1928, showed that there are polyhedra for which no tetrahedralisation exists. A simple illustration of how to create a Schönhardt's polyhedron is given in figure 9-54. Starting with a regular polyhedron (left), take the top facet (shaded blue) and twist it slightly creating a kink in each of the three quadrilateral faces. There is now no way to fully tetrahedralise the space occupied by the polyhedron using only its own vertices.

These kinds of polyhedrons rarely emerge naturally, but when they do, it will cause the meshing algorithm to fail. The meshing algorithm's default response to a failed geometry containing a Schönhardt's polyhedral is to carve out a portion of the interior and begin filling it again from a different facet. This approach only works for cases where the initial untetrahedralisable region was caused by an element quality or other issue and not a Schönhardt's polyhedral (although the mesher currently has no way of detecting Schönhardt's polyhedrals).



*Figure 9-54 – Schönhardt's polyhedron*

A more reliable approach to resolving Schönhardt's polyhedrals is to insert a Steiner point, either on an edge or within the interior of the space. Referring to figure 9-54, a single Steiner point inserted at the geometric centre of this shape would decompose the region and allow the existing meshing code to tetrahedralise the space.

Selecting where and how many Steiner points to insert is still hotly debated in literature. There is no deterministic method that works for all cases. Figure 9-55 shows a simple Steiner point insertion algorithm developed for this project but not active in the mesher.

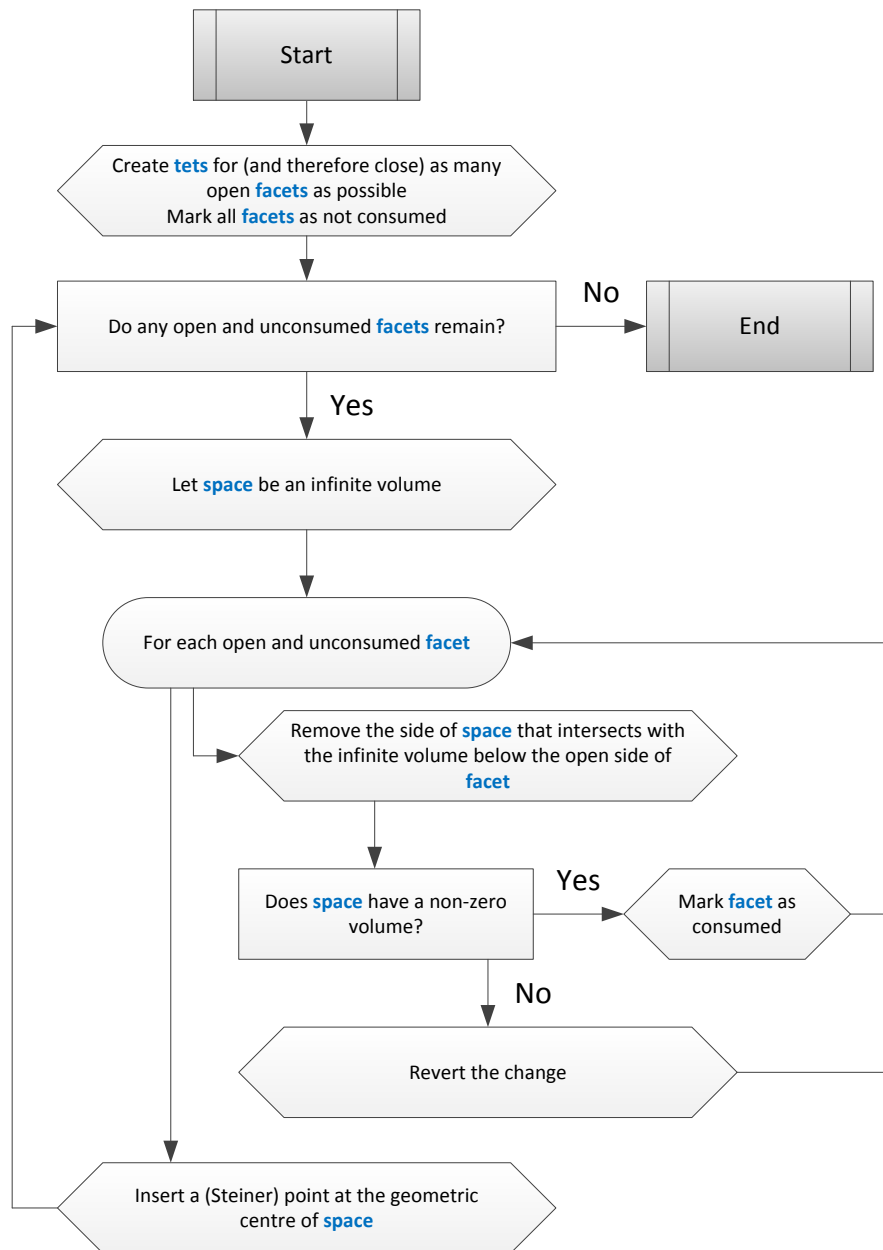


Figure 9-55 – Steiner point insertion procedure diagram

This method works in many scenarios, but will occasionally produce very poor aspect ratio elements and elements with glancing segments. Three specific pieces of further work would contribute substantially to increasing the reliability of the mesher:

- develop a more robust method to insert Steiner points superior to that described in figure 9-55;
- develop a method for Steiner point insertion on element edges; and
- implement a method to refine poor quality elements as they emerge to prevent glancing segment issues.

# Chapter Ten – Results and Discussion

This work set out to prove that multi-angle valve seat machining operations can be simulated in three dimensions in a number of hours rather than days, by using a range of simplifications. The model developed for this work is highly parametric and requires a minimum amount of experimental effort when setting up for a new cutting system.

This chapter presents a validation of the model against data gathered from three configurations used in the experiment presented in Chapter Seven. A review of performance indicators such as computation time, mesh quality and mesh sensitivity is also offered. Finally, this chapter applies the model to a hypothetical multi-angle problem and its proposed solution.

## 10.1 Model Performance & Validation

To validate the model, a series of test simulations were performed and compared to experimental data gathered in Chapter Seven. The experiment in Chapter Seven was based on a single cutter held rigidly with a rotating workpiece. The simulations generated to mimic that experiment have a fixed workpiece and single rotating cutter. As discussed in previous chapters, the relative motion between the tool and workpiece is the same.

Validation aims to ensure that the mesh splitting algorithm developed in section 9.7 and the 3D meshing algorithm developed in section 9.9 is fast and can process cutter-workpiece intersections and generate new geometry for each simulation increment, sufficient to model a sub-segment of cutting. This section also tests the ability of the code written to recover cutting forces based on the algorithm developed in section 9.7.12.

Validation was based on cutting forces measured in Chapter Seven according to the parameter selection given in table 10-1 for three different configurations. The simulations were allowed to run until a final feed depth of 0.7 mm (the final maximum feed depth for Sigma exhaust valve seats).

Parameter	Configuration 1	Configuration 2	Configuration 3
Target spindle speed	1666.7 RPM	1666.7 RPM	1666.7 RPM
Feed rate	0.08 mm / rev	0.03 mm /rev	0.08 mm / rev
Lubrication	Dry	Dry	MQL

*Table 10-1 – Configurations tested*

A 30° sub-segment of the workpiece was modelled for the reasons discussed in section 9.10. This provides an opportunity to demonstrate the intermittent cutting and rapid mode capabilities built into the Fortran program.

The simulation was run using a single core on a computer with an Intel Core i7-3630QM processor, 16 GB of random access memory (RAM) and a 500GB solid state drive (SSD).

The target element edge length was set to 0.6 mm (however the vast majority of edges were less than this due to curvature controls). The initial workpiece mesh contained 4048 nodes and 3234 elements and the workpiece bulk mesh contained 391 nodes and 220 elements.

Preparing the Marc model for simulation follows the procedure given in figure 8-3. First, the configuration file for each simulation was defined according to section 8.4.9 using the parameters given in table 10-1. For

each configuration, Marc was used to invoke the Python model builder script described in section 8.6, which sets up the Marc simulation including geometry, boundary conditions, contact bodies and materials as defined in sections 8.4.3 through 8.4.6 respectively. The script also sets up the contact table, remeshing, load case and job instructions as defined in sections 8.6.5 through 8.6.8 respectively. This process takes approximately 20 seconds and once complete, no further changes are required in Marc.

The next step is to simply run the simulation in Marc through its 'Run Job' user interface. Very soon after launch, Marc calls the user subroutine UMAKNET in the Fortran code. The first time this is called, the Fortran program defers to Marc's internal mesher to generate an initial tetrahedral mesh from the seed hexahedral mesh. For all subsequent calls to UMAKNET, the Fortran code uses its own mesher to generate meshes according to the procedure laid out in section 9.7. The relationship between Marc's call hierarchy and the Fortran code is detailed in section 9.5.

During simulation, the Fortran code logs load and position data for the spindle, workpiece and cutters to individual files. These files are used to generate the plots shown throughout this chapter.

### 10.1.1 Speed

Figure 10-1 shows a chart of cutting operation time vs. computation time in seconds for configuration one. As the chart shows, the simulation runs in approximately linear time. The steps visible within the data distinguish the periods of time where the cutter was in rapid mode and where the cutter was in contact with the workpiece. The times shown here include the time required by Marc to assemble and solve the stiffness matrix as well as process the post file outputs.

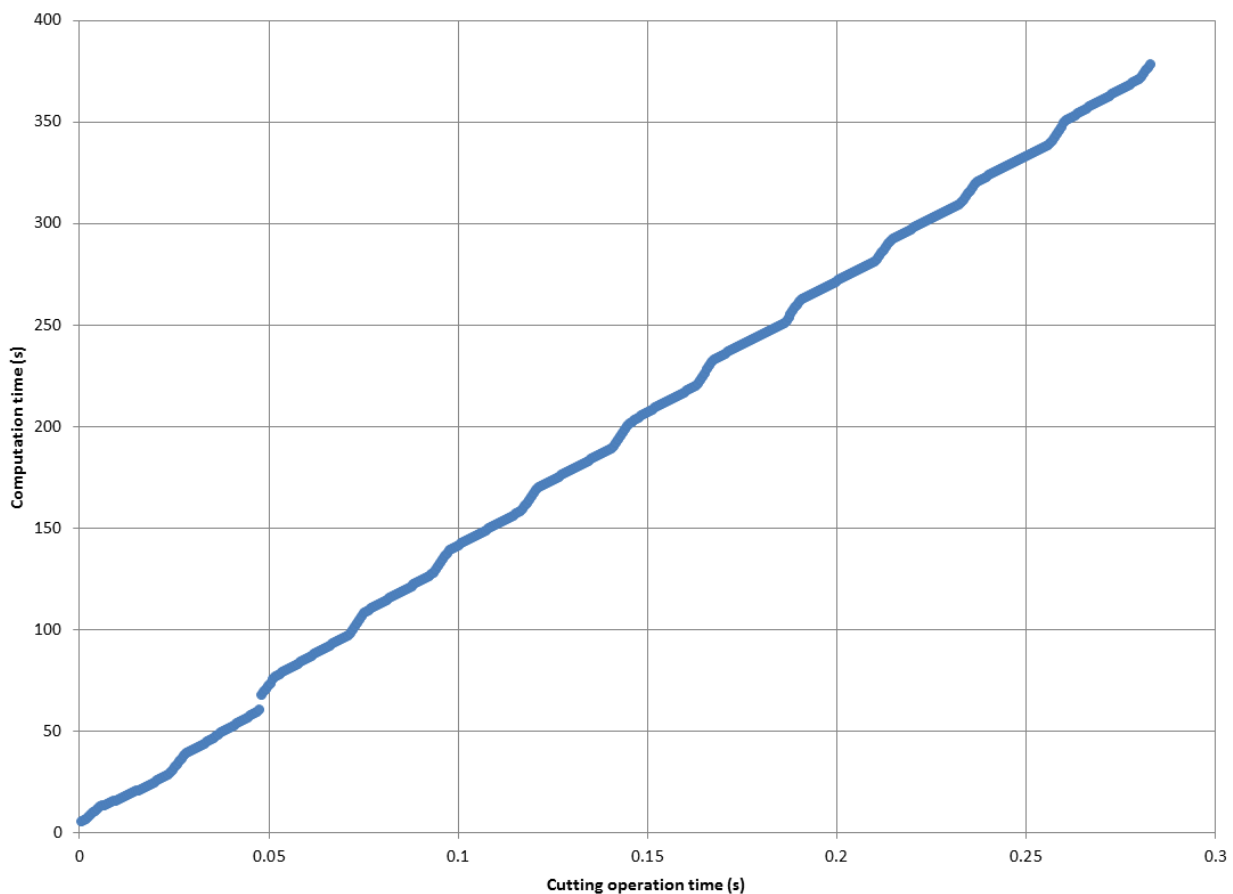


Figure 10-1 – Chart showing computation time (s) vs. cutting operation time (s)

As the figure shows, the 11 passes were completed in 378 seconds on the hardware stated above. An equivalent simulation with adaptive remeshing that models chip flow plasticity and damage would take considerably much more time.

### 10.1.2 Validation of Loads

For configurations one through three, the charts in figures 10-2 through 10-4 respectively, show comparisons between the experimentally measured loads, simulation loads and theoretical loads according to the cutting force prediction model developed in Chapter Seven (see table 7-4). In each chart, the experimentally measured loads are shown in red, the simulation loads in blue and the theoretical loads in green. The simulation loads were intermittent as only 30° of the seat was modelled. The first cutting load shown on the chart represented the initial contact load as the cutter fed into the workpiece.

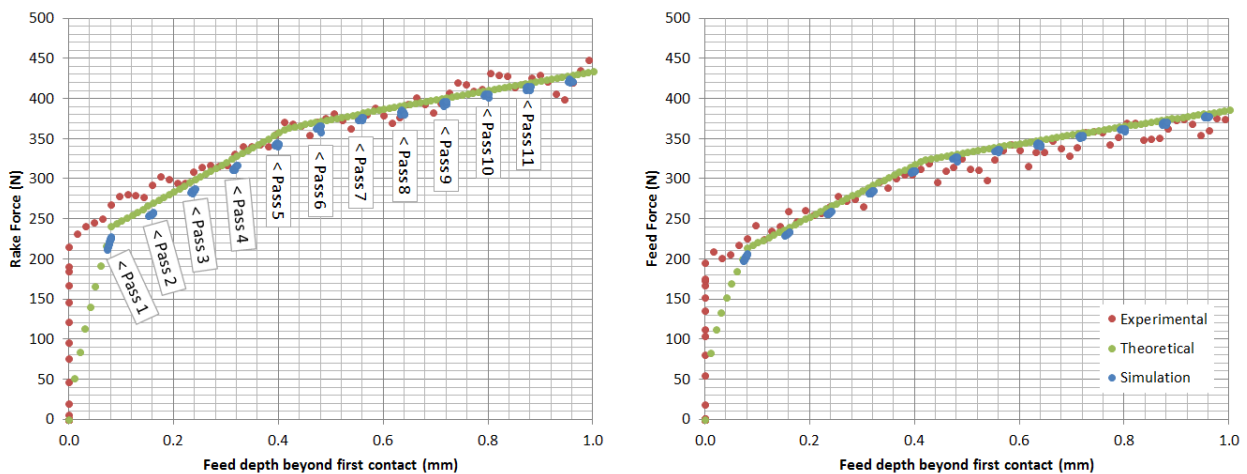


Figure 10-2 – Dry,  $f=0.08 \text{ mm rev}^{-1}$ , rake and feed, experimental, theoretical and simulation loads

For configuration one, shown in figure 10-2, the theoretical model and simulation results show close agreement for both rake and feed loads. From a feed depth of 0.08 mm, these data also show close agreement with the experimental data. Before this point however, the experimental data immediately ramps to full load in contradiction with the theoretical model. One possible explanation for this is that the initial friction rubbing between the cutter and workpiece generates some thermal expansion in the workpiece that temporarily accelerates the rate of penetration of the tool into the workpiece. Zhou, Andersson and Ståhl, 2004, identify thermal expansion of both the tool and workpiece as a considerable source of error in precision machining. This is not taken into account in the theoretical model, due to its complexity and the lack of a good justification for refining the initial contact representation given that this portion represents a very short period during machining.

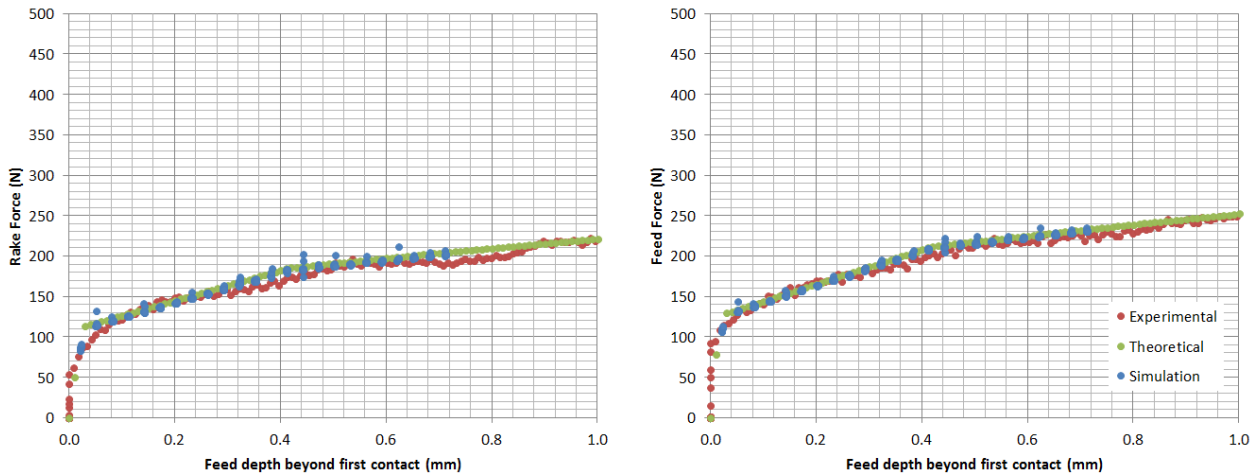


Figure 10-3 – Dry,  $f=0.03 \text{ mm rev}^{-1}$ , rake and feed, experimental, theoretical and simulation loads

For configuration two, shown in figure 10-3, the simulation produces rake and feed forces significantly lower than those shown in configuration one due to its shallower depth of cut. These are as expected when comparing the simulation loads with the theoretical model and experimental data.

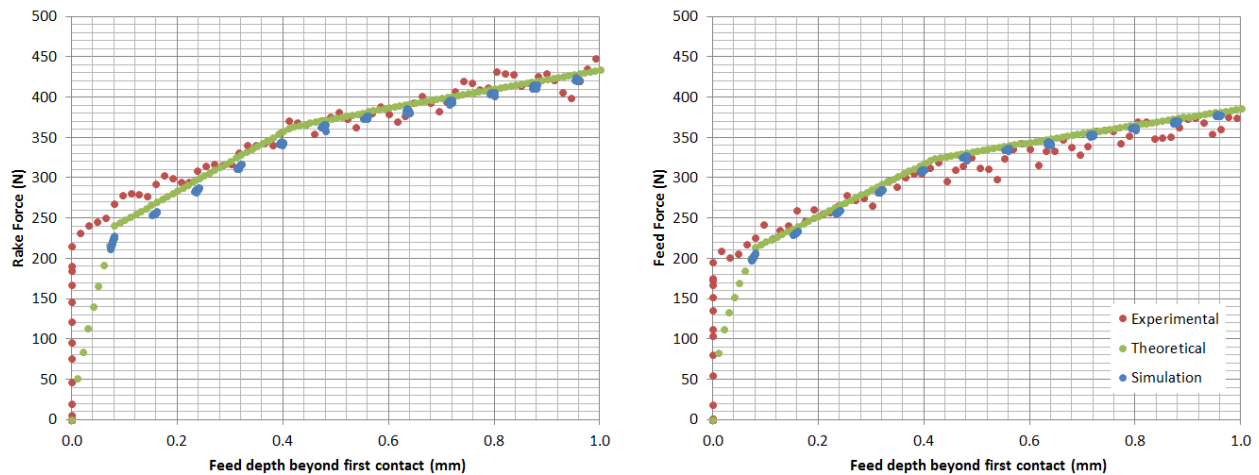


Figure 10-4 – MQL,  $f=0.08 \text{ mm rev}^{-1}$ , rake and feed, experimental, theoretical and simulation loads

Configuration three, shown in figure 10-4 also shows good agreement between experimental, theoretical and simulation loads. When comparing configurations one and three, it can be seen that the presence of MQL lubricant has little effect on the cutting load as expected based on the findings made in Chapter Seven.

### 10.1.3 Validation of Geometry

When considering the quality of the geometry between passes, the most important output is the shape, according to the following properties:

- the cut material shape should closely follow the path the cutter has taken through the material;
- the surviving surface should be flat, free from steps and without the saw-tooth pattern discussed in Chapter Nine; and
- the top surface should match the height of the bottom of the cutter.

These features are critical because subsequent cutter passes are affected by earlier passes, particularly when calculating the depth of cut. Secondary to these requirements, the following properties are desirable:

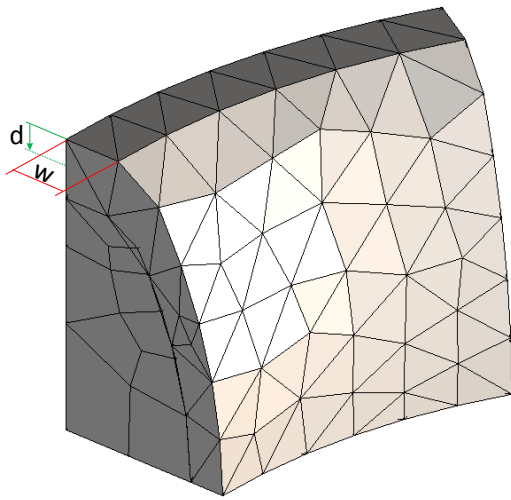
- decreasing, stable (or at least not significantly increasing) element and node counts;
- absence of needles and caps;
- absence of sharp changes in mesh density;
- absence of gaps, voids or cavities; and
- absence of noticeable nodal perturbations (nodes may sometimes be perturbed to help generate a closed split line or during simplification to avoid creating a cluster. If this mechanism is not correctly controlled, it can lead to nodes being raised above the cut surface creating lumps on the cut surface).

These properties primarily address requirements of Marc's finite element calculations, where smooth uniform meshes with ideal aspect ratio elements are desirable. They also decrease the probability that the tetrahedral mesher developed in Chapter Nine would fail to generate a mesh.

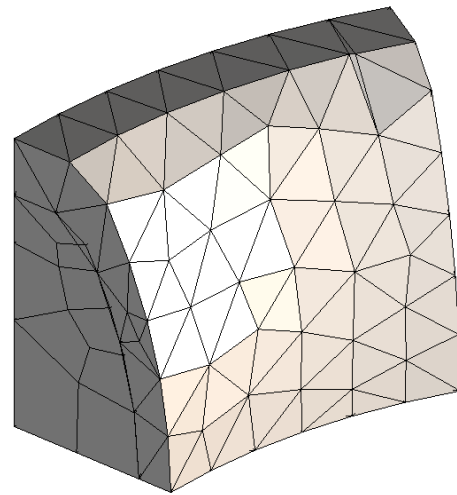
Figure 10-5 shows the original mesh from configuration one followed by the first five passes, likewise figure 10-6 shows passes 6 through 11. Passes 5 through 7 showed a lower quality representation of the width of cut. This stems from the mesh generated by Marc's tetrahedral mesher from the original input mesh and not from errors made in the Fortran program. If a shorter target edge length were used, Marc's mesher could generate a more accurate initial tetrahedral mesh, which would reduce the deviation seen on the width of cut in these passes.

Each pass in figures 10-5 and 10-6 represents the result of at least eight new mesh generations as the cutter traverses the exposed workpiece surface. The original mesh in figure 10-5 defines the width of cut,  $w$ , and depth of cut beyond first contact,  $d$ . Each subsequent mesh gives values for these parameters. Figure 10-2, labels the pass number corresponding to each cluster of simulation rake force points (in blue).

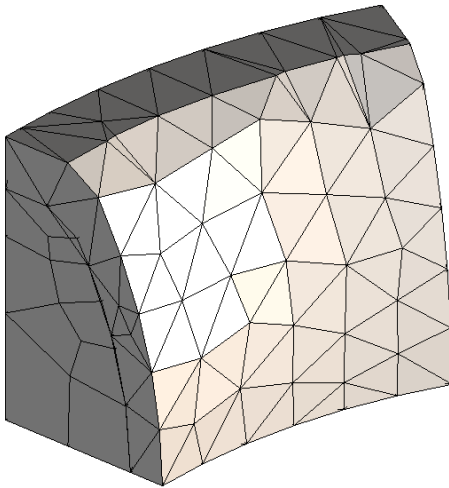




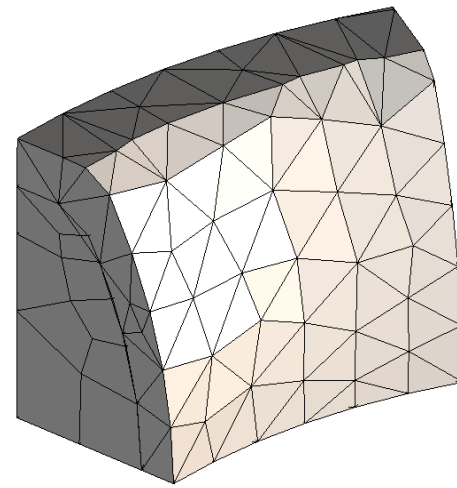
Pass 0 (Original Mesh)  
 $d = 0.00$ ,  $w = N/A$



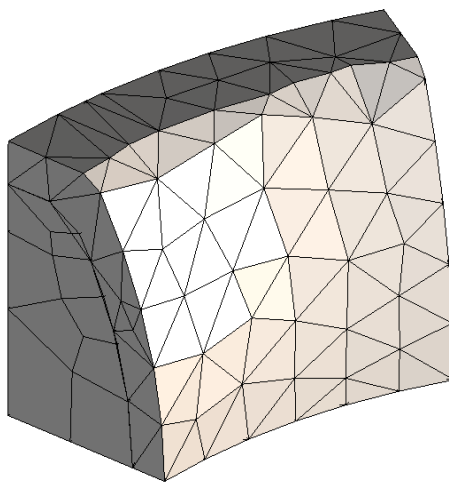
Pass 1  
 $d = 0.09$  mm,  $w = 1.13$  mm



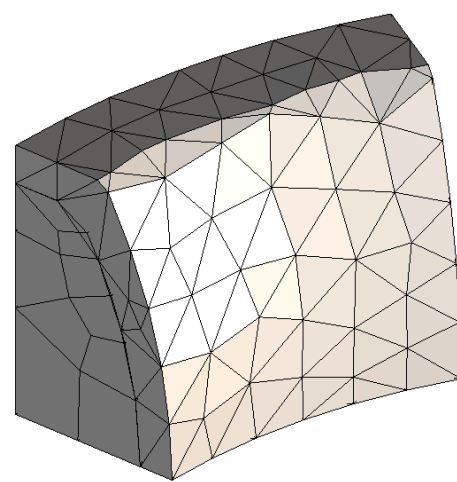
Pass 2  
 $d = 0.16$  mm,  $w = 1.27$  mm



Pass 3  
 $d = 0.24$  mm,  $w = 1.41$  mm

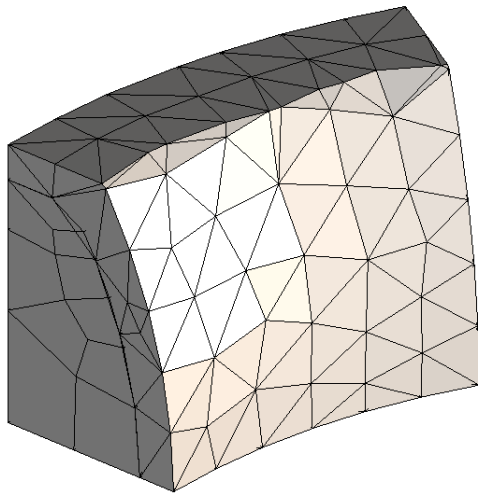


Pass 4  
 $d = 0.32$  mm,  $w = 1.55$  mm

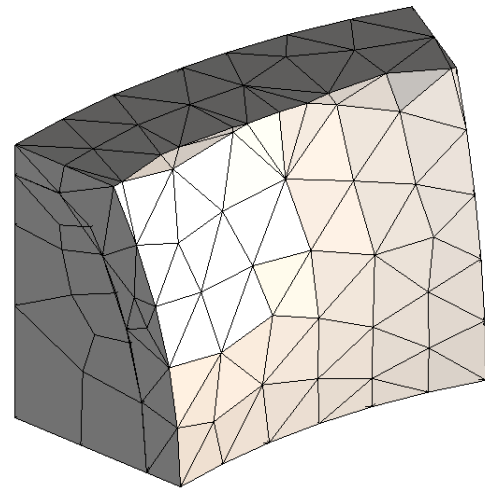


Pass 5  
 $d = 0.40$  mm,  $w = 1.69$  mm

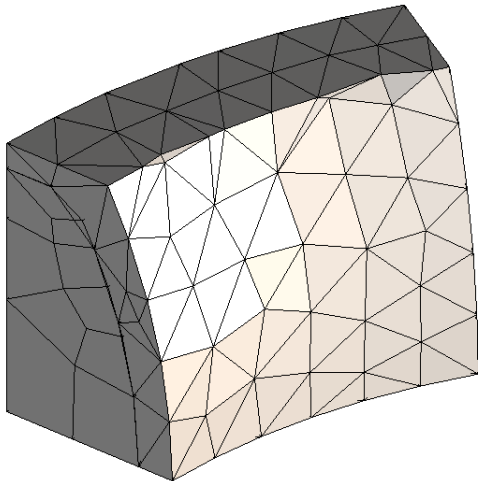
Figure 10-5 – Original mesh and passes 1 through 5



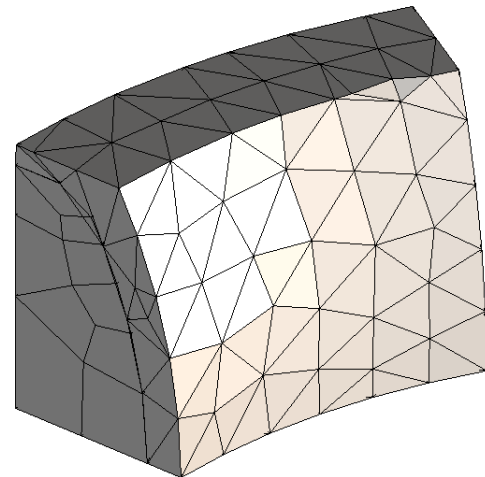
Pass 6  
 $d = 0.48 \text{ mm}$ ,  $w = 1.76 \text{ mm}$



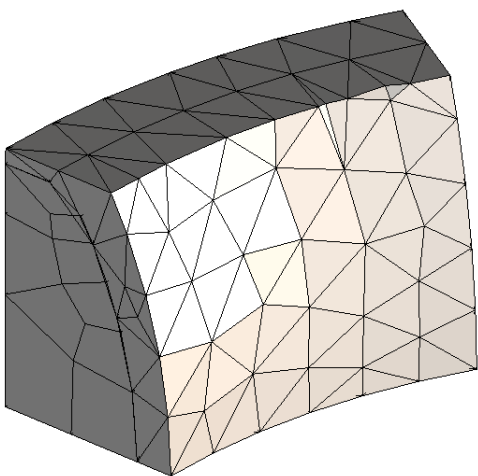
Pass 7  
 $d = 0.56$ ,  $w = 1.81 \text{ mm}$



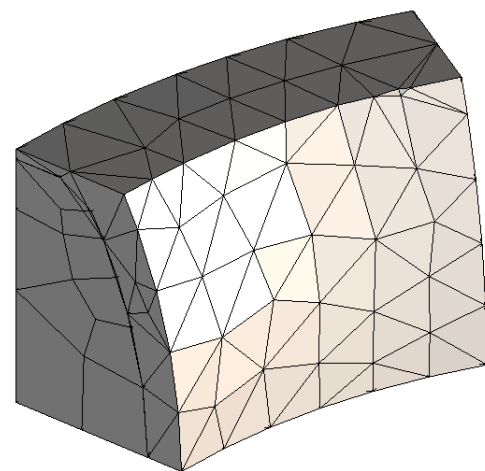
Pass 8  
 $d = 0.64$ ,  $w = 1.85 \text{ mm}$



Pass 9  
 $d = 0.72 \text{ mm}$ ,  $w = 1.90 \text{ mm}$



Pass 10  
 $d = 0.80$ ,  $w = 1.94 \text{ mm}$



Pass 11  
 $d = 0.88 \text{ mm}$ ,  $w = 1.99 \text{ mm}$

*Figure 10-6 – Passes 6 through 11*

As figures 10-5 and 10-6 show, the variation between each new mesh is subtle despite the entire mesh being regenerated eight times per pass. This is a strength of the meshing algorithm developed for this work in that mesher is able to generate incremental meshes while minimising the number of points inserted or moved and maximising the number of facets recycled from the previous increment. This helps reduce interpolation error, since the vast majority of nodal locations are the same, interpolation is barely required at all.

Other algorithms such as TetGen and Marc's own internal mesher will commonly re-discretise the entire volume in the pursuit of elements that have superior aspect ratios and angles. This is not surprising since they are both designed to generate a high quality mesh for a given piecewise linear complex (PLC), they are not designed to generate a mesh from a PLC and then continually update that mesh as elements are cut by a tool (Si, 2015; MSC Software, 2016a).

While they can both respect hard edges of the mesh they readily discard soft edges (edges between facets that share a shallow angle to one another) and will place nodes in locations best suited for element quality without considering interpolation. This would cause Marc to interpolate more frequently over longer distances, introducing error at every step.

The strategy employed by both Marc and TetGen can also lead to rounding off error (observed earlier in section 4.3), particularly in regions where the cutter has disrupted a hard edge. This can introduce significant error both in tracking the volume of material removed and calculating the cutting forces when the cutter revisits that region of the mesh. The mesher developed for this work is completely immune to rounding off error as shown in the figures.

Poor aspect ratio elements can be a source of error in non-linear simulations, it is therefore advisable to reduce the number of poor aspect ratio elements wherever possible. Despite the undesirability of poor aspect ratio elements, it was not critical in this work that the mesh was completely free from them, since there is no plastic deformation and no large elastic deformation of the mesh.

In the case of the simulation developed for this work, a balance needed to be struck between element count and the number of poor aspect ratio elements, since removing poor aspect ratio elements without affecting the external shape of the contact body would necessitate the insertion of many more elements. For this reason, the edge simplification procedure developed in Chapter Nine focused more on reducing element counts and respecting existing geometry features over generating ideal aspect ratio elements.

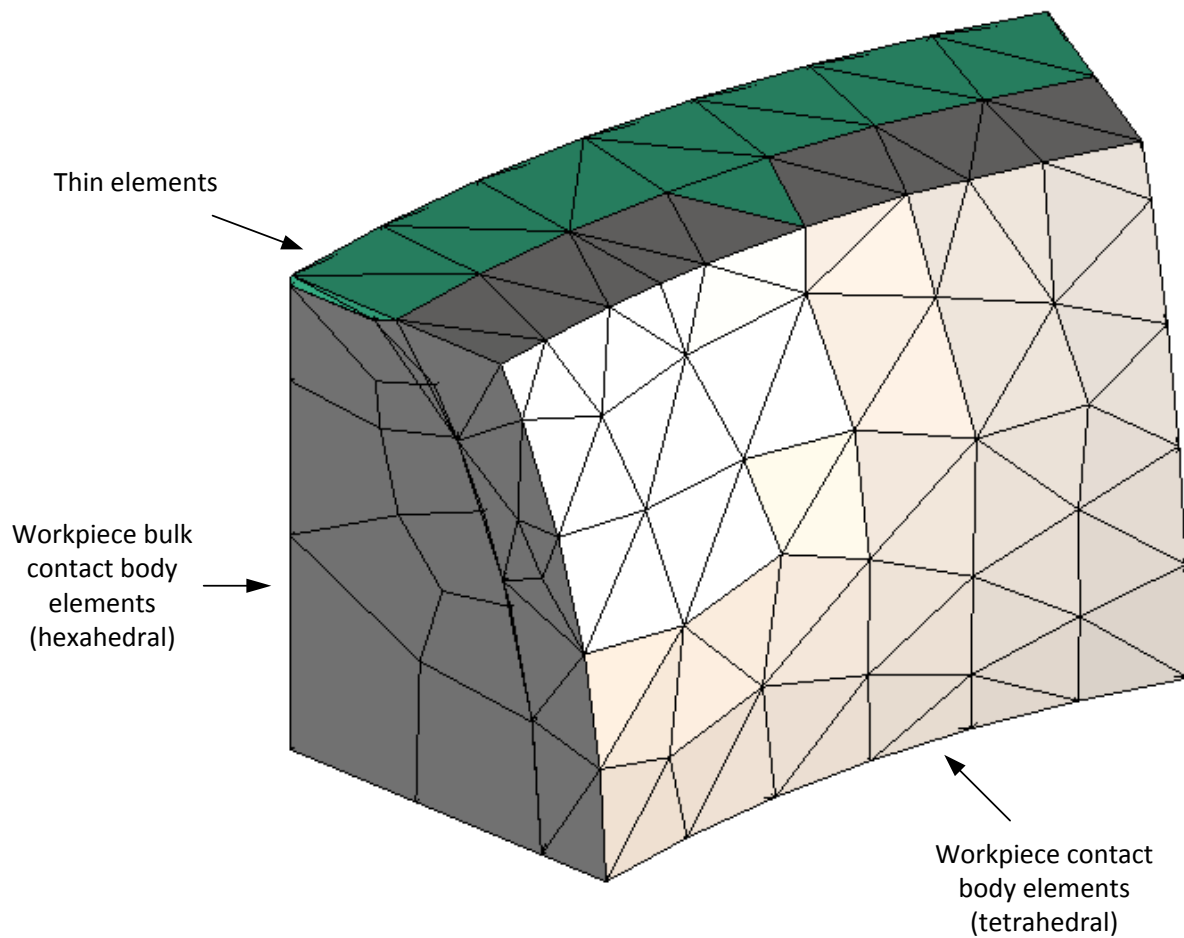
Table 10-2 shows a summary of poor aspect ratio element counts discovered in the output mesh according to Marc's element quality check tool. These summary data were taken at the end of each cutting pass. It is important to note therefore, that poor aspect ratio elements may have existed in intermediate mid-cut increments, these increments are not shown in this table.

Pass	Reference increment	Count of poor aspect ratio elements	Percentage of workpiece elements that are poor aspect ratio
Original	1	0	0.00%
1	11	2	0.87%
2	48	0	0.00%
3	87	1	0.41%
4	127	1	0.39%
5	164	2	0.74%
6	203	9	3.32%
7	243	13	4.78%
8	280	10	3.72%
9	319	2	0.77%
10	359	0	0.00%
11	396	7	2.79%
12	435	46	18.40%

*Table 10-2 – Summary of poor aspect ratio element counts*

As the table shows, with the exception of the last pass, the number of poor aspect ratio elements that appeared in the mesh was generally low. This demonstrates that a good balance had been reached between preserving the shape of the workpiece mesh and the quality of the mesh.

The last pass listed in the table was made beyond the 1 mm total depth of cut specified. Figure 10-7 shows the poor aspect ratio elements on the final pass highlighted. Those elements generated because they contained the freshly exposed cut surface and the glue contact between the workpiece and workpiece bulk contact bodies and are therefore naturally thinner than elements created on earlier passes.

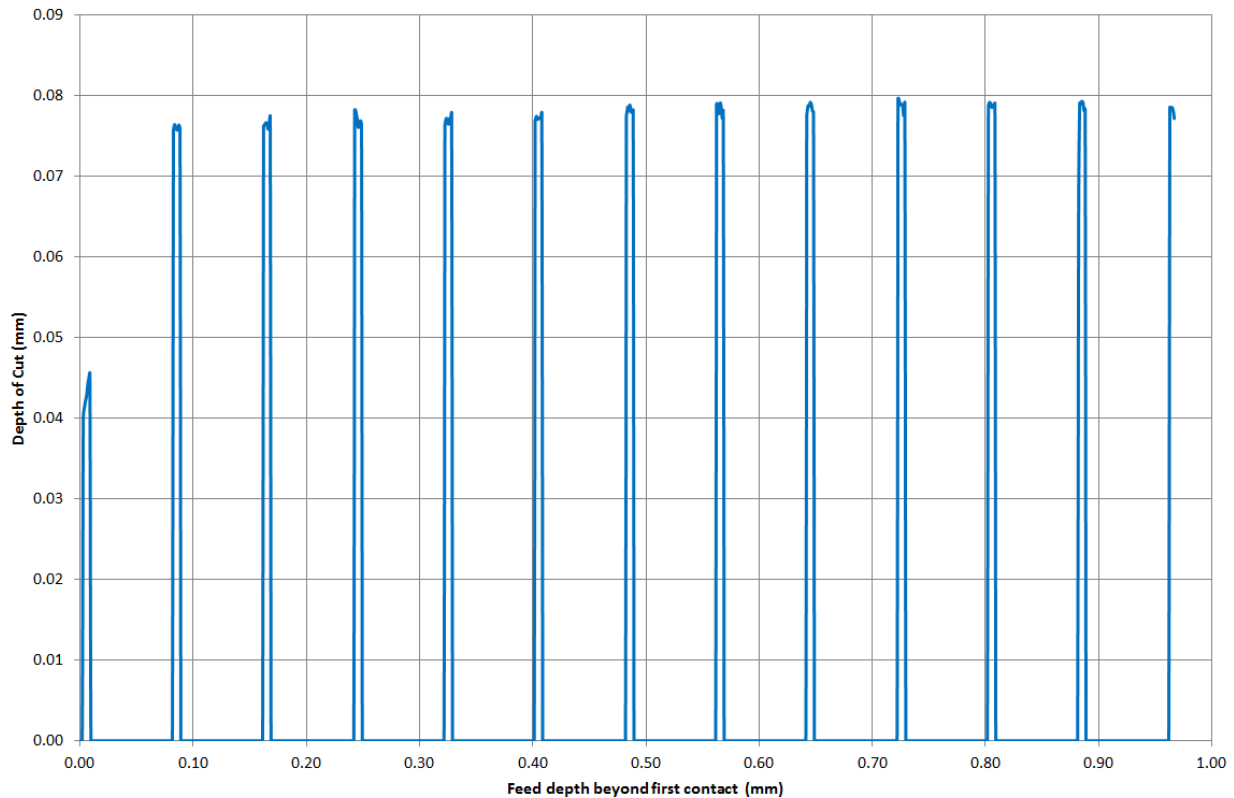


*Figure 10-7 – Poor aspect ratio elements on final pass*

This increase in poor aspect ratio element count where the workpiece contact body mesh thins, highlights the importance of including sufficient extra material below the final depth of cut to maintain compatibility with the remeshing algorithm developed in Chapter Nine.

In terms of quality, the cut face shape (the flatness and level of the exposed cut surface), was perfect across all passes.

Figure 10-8 shows the simulation depth of cut according to the cutter face projection algorithm developed in Chapter Nine. The data shows close agreement to the spindle feed rate of 0.08 mm per revolution.



*Figure 10-8 – Depth of cut vs. feed depth beyond first contact*

Perfect agreement was not reached for several reasons. Firstly, the projection algorithm works by projecting all split lines to the local feed-radial plane of the cutter. This method does not take into account the curvature of the cutter path and so the proximity of split lines to the feed-radial plane affects the radial error. For this reason, the peaks of the data during periods of contact are not flat, as they would normally be expected to be.

Secondly, the average of the peaks was clearly below 0.08 mm. This is because the feed-radial plane is aligned with the control node rather than the front face of the cutter. Split lines are projected along the incremental change in position vector, which is at a gradient to the global coordinate system. Therefore the relative position of the local feed-radial plane affects the height (along the feed axis) of the projected split lines. The front face of the cutter is not guaranteed to be coplanar with the feed-radial plane so there is no obvious position, other than the position of the control node, for the local feed-radial plane to intersect.

These two factors could be improved by reducing the cutter step size between increments and by moving the control node to the front plane of the cutter.

Finally, the split lines are further discretised by the algorithm described in Chapter Nine, which results in some loss of quality where the split lines form a curve or slope. This introduces some noticeable noise between increments. This error could be reduced by increasing the cutter face discretisation resolution, which is set in the simulation configuration file.

#### 10.1.4 Progression of Mesh Complexity

A problem that can plague non-linear simulations with many increments is a drift towards increasing element and node counts. In some cases this can be a dominant limiting factor that restricts how much progress certain simulations can achieve.

The type of simulation developed in this work is particularly vulnerable to this problem, as the frequent and significant changes in geometry, especially around the step change at the cutter / workpiece intersection, require increased numbers of elements to represent the increase in complexity.

Chapter Eight discussed a hexahedral subdivision based technique that would have eliminated the need for the tetrahedral mesher developed in Chapter Nine. Whilst the temptation to use a subdivision based adaptive mesh technique was great due to the reduced effort required in development, that chapter showed how the hexahedral element subdivision technique discussed is vulnerable to runaway neighbour subdivisions.

The number of elements and nodes is by far the most dominant factor affecting simulation speed, both within Marc and the Fortran program developed for this work.

Subdivision methods generally leave behind many more elements than are deactivated by the cutting pass that led to their creation. This leads to a constant upward trend in element and node count, with the only possible options available limited to controls that help reduce the rate of increase.

By using the bespoke tetrahedral meshing algorithm developed in Chapter Nine, many more possible options were available to control element and node counts, such as local only meshing and edge coarsening. For the model proposed in this work to be effective, it was essential that progression of the simulation was not dependant on increasing element and node counts. The simulation must be capable of reducing element and node counts wherever the opportunity arises, without compromising quality.

Figure 10-9 shows the workpiece element and node count change over time for configuration one. As the figure shows, there was no runaway increase in element and node count. Furthermore, as the simulation progressed, the element count gradually decreased.

The initial increase represents an initial convergence towards an element count that satisfies the target edge length and element count required to accurately model the shape of the geometry over time. Throughout the simulation, the workpiece volume was gradually decreasing, the dip in element count towards the end of the simulation shows that the tetrahedral remesher developed for this work is capable of successfully capitalising on opportunities to reduce element counts.

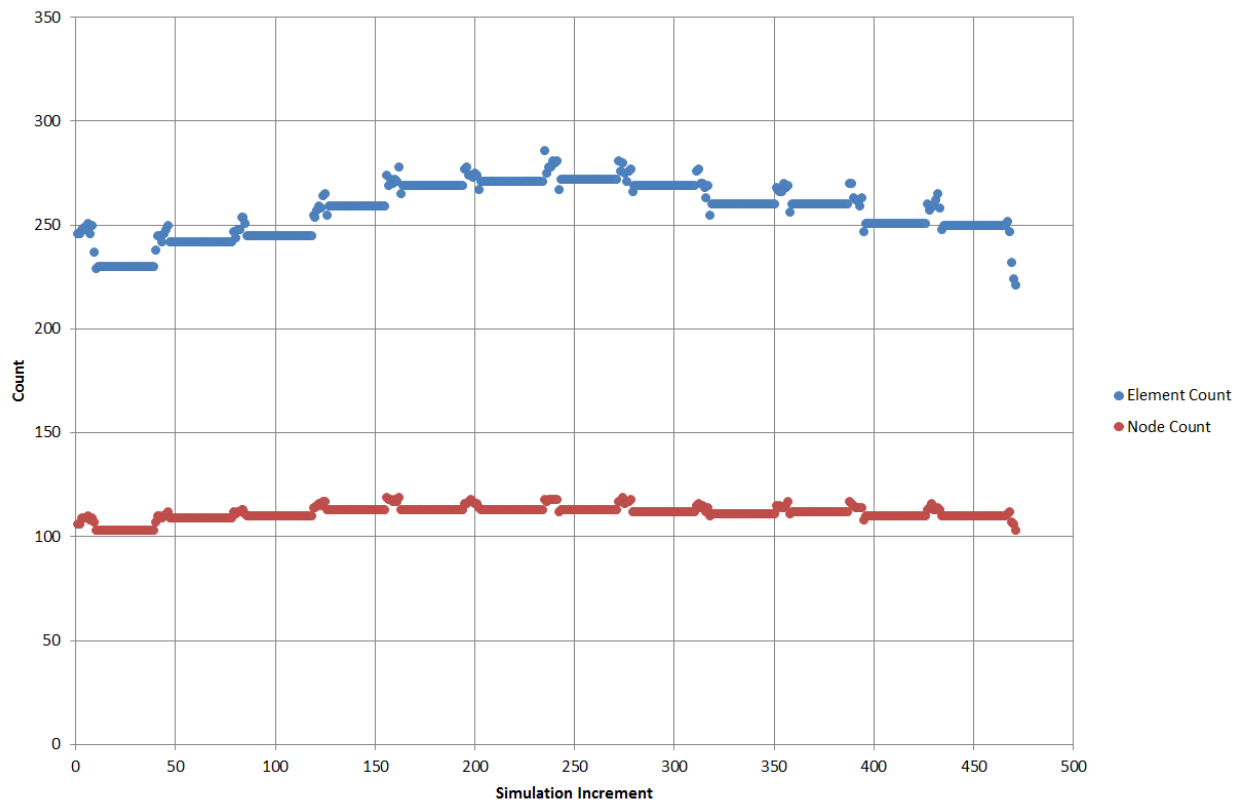


Figure 10-9 – Element and node counts vs. simulation increment

Figure 10-10 shows the relationship between rake force and feed depth beyond first contact for five different target element edge lengths. These data are based on the first ten passes of configuration one. As the data show, the rake force does not vary significantly regardless of target element edge length. Providing the input geometry can be represented to a sufficient degree of detail (based on user preference for curvature control) the force recovery algorithm and mesher developed for this work is not sensitive to element edge length. This means that simulations can be run with dramatically fewer elements than traditional simulations that model chip flow, plasticity and other phenomena that require high element densities.

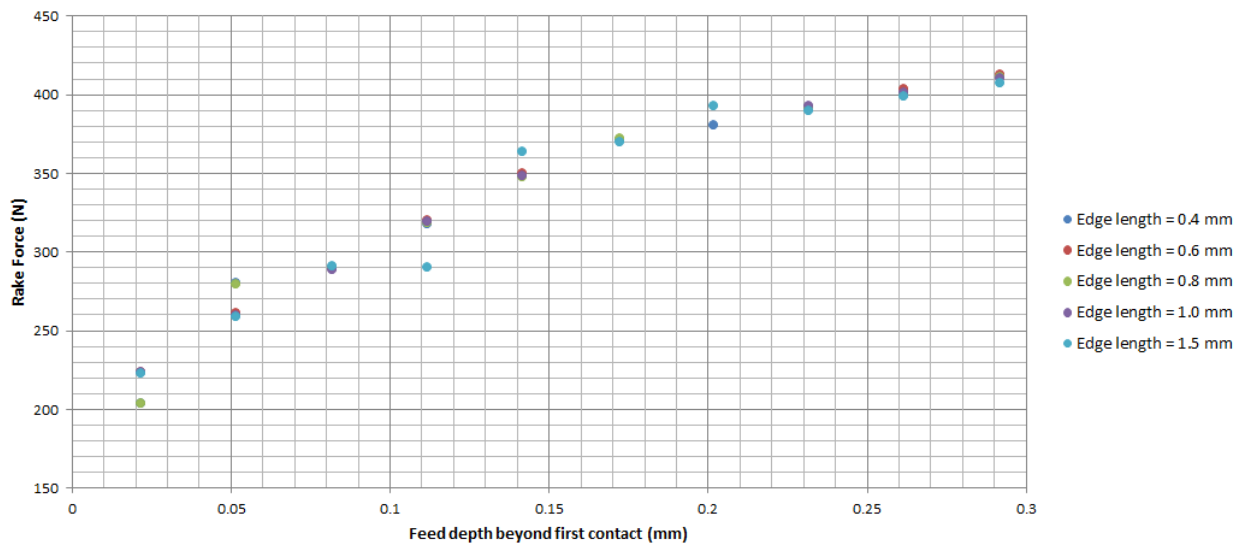


Figure 10-10 – Element density sensitivity



### 10.1.5 Validation of Volume Calculation

Unlike the volume of hexahedral elements (which can have ambiguous volumes), tetrahedral elements always have certain volumes.

Element volume calculations were used to output the volume of material removed per increment, as well as in the calculation of the workpiece cutting pressure. Although tracking the volume is a trivial accounting problem, it is important to verify that both the tetrahedron calculation used is implemented correctly and that the correct elements were included by the Fortran program in the overall sum.

Figure 10-11 shows the total workpiece contact body volume for every increment in blue (not including the workpiece bulk contact body volume which never changes). The figure also shows a series of manual check volume calculations performed using Marc's element volume tool from the user interface. These check volumes were taken from the point immediately after the cutter finishes a pass. As these data show, the volume calculation implemented in the Fortran program shows perfect agreement with the manual calculations performed in Marc.

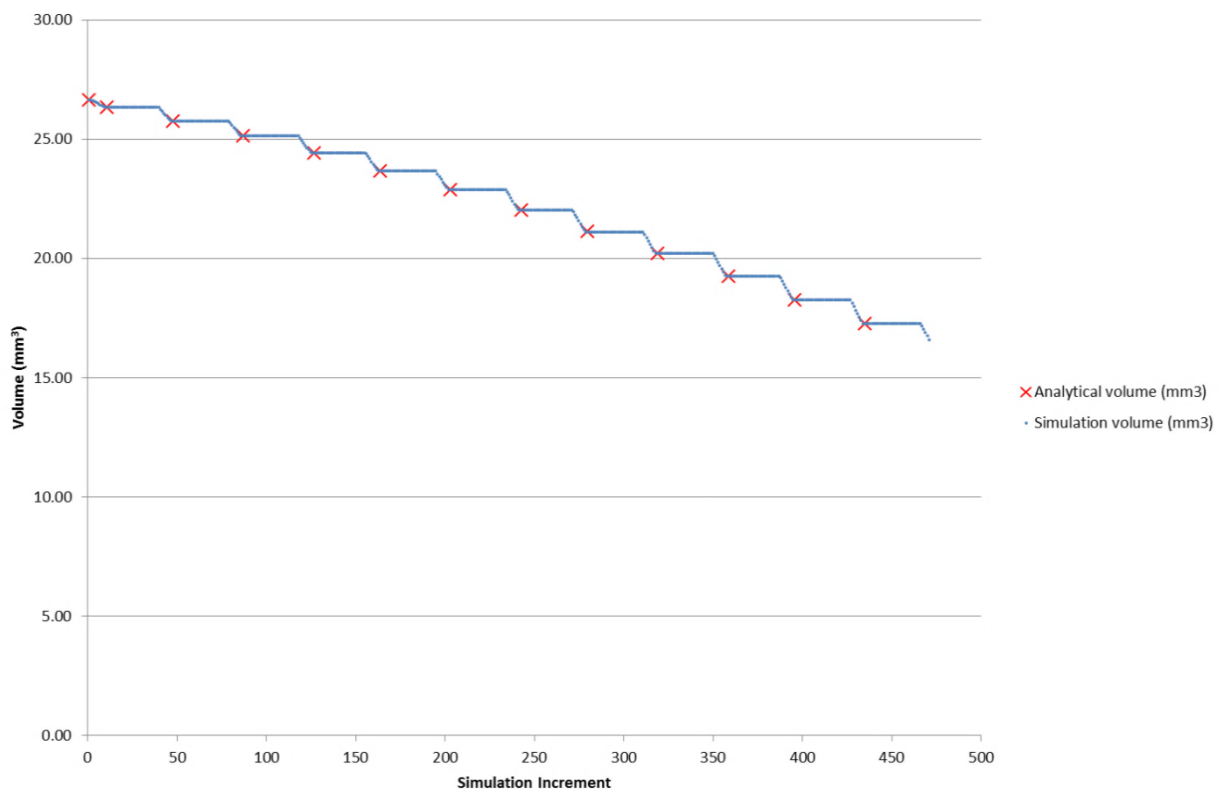


Figure 10-11 – Analytical volume vs. simulation volume

The close agreement between analytical and simulation calculated volume verifies that the volume calculation algorithm presented in section 9.8.1 is functioning correctly. Furthermore these data prove that the workpiece continues to lose volume throughout the simulation (as expected during cutting) and that the rate of volume loss accelerates with simulation increment as the width of cut and front facing area increases. The constant volume points between contact passes indicate that no mesher activity is taking place and no rounding off error is accumulating.

### 10.1.6 Cutter Face Duty Cycle Map

With a typical valve seat cutting operation, each cutting insert has a different duty cycle. Furthermore this duty cycle varies along the length of the cutting edge. This causes variation in wear level between the cutters in the cutting system.

The simulation developed for this work can output incremental cutting energy which can be viewed as a heat map across an illustration of the cutter. By enabling this feature in the configuration file, the simulation will incrementally dump the cutter face intersection profile to a comma separated values (CSV) file for each cutter. The data are saved in terms of each cutters local coordinate system. The force and rotational displacement associated with each point in the profile is also output to the file.

These data can be loaded by a bespoke Python utility that reconstructs the intersection profile and generates a heat map which is plotted over the cutter shape using Matplotlib. The script automatically selects an appropriate cutter geometry illustration to overlay on the data to give the user a better sense of where the profile is relative to the boundaries of the cutter.

Figure 10-12 shows an example of this plot for configuration one (defined in table 10-1) limited to a depth of 0.7 mm as in the real cutting operation. As the figure shows, the relative area of intersection was very small compared to the size of the cutting insert. Figure 10-13 shows a high zoom view of the intersection region.

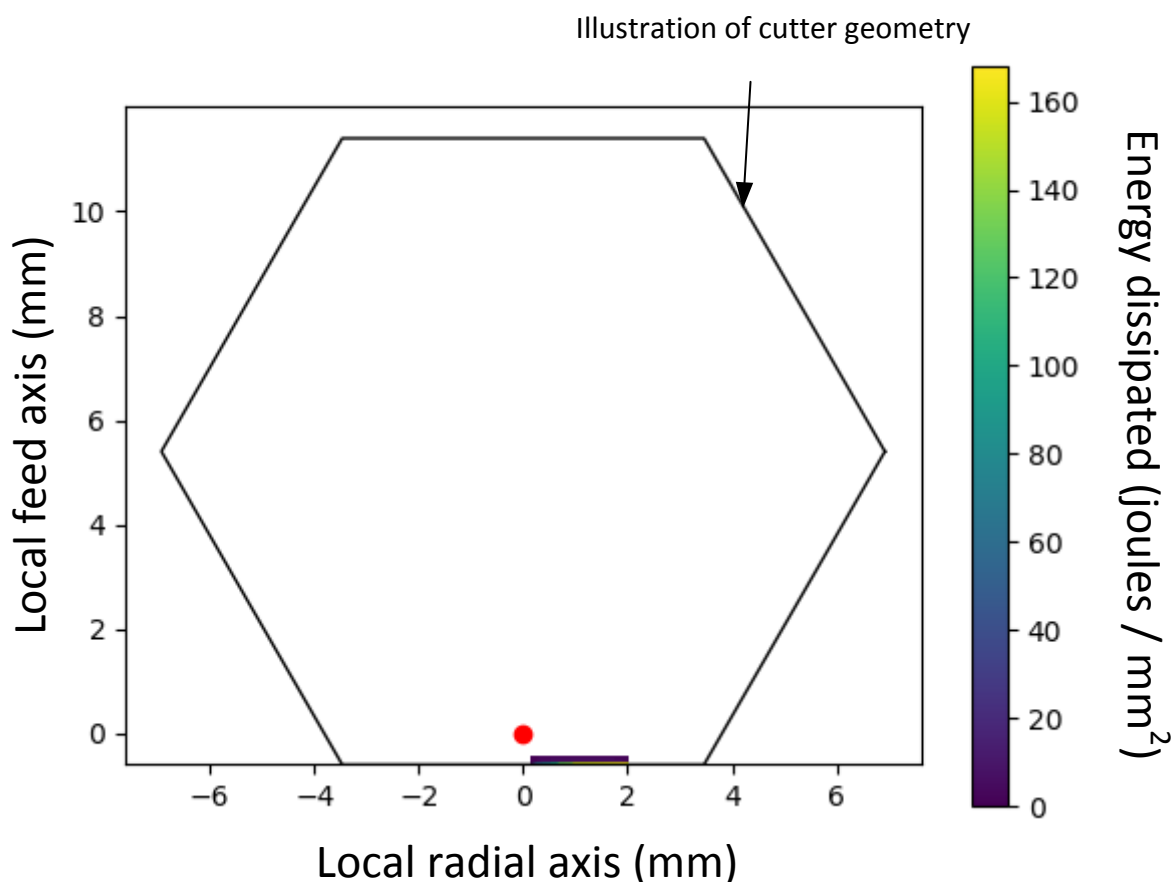


Figure 10-12 – Cutter duty cycle heat map

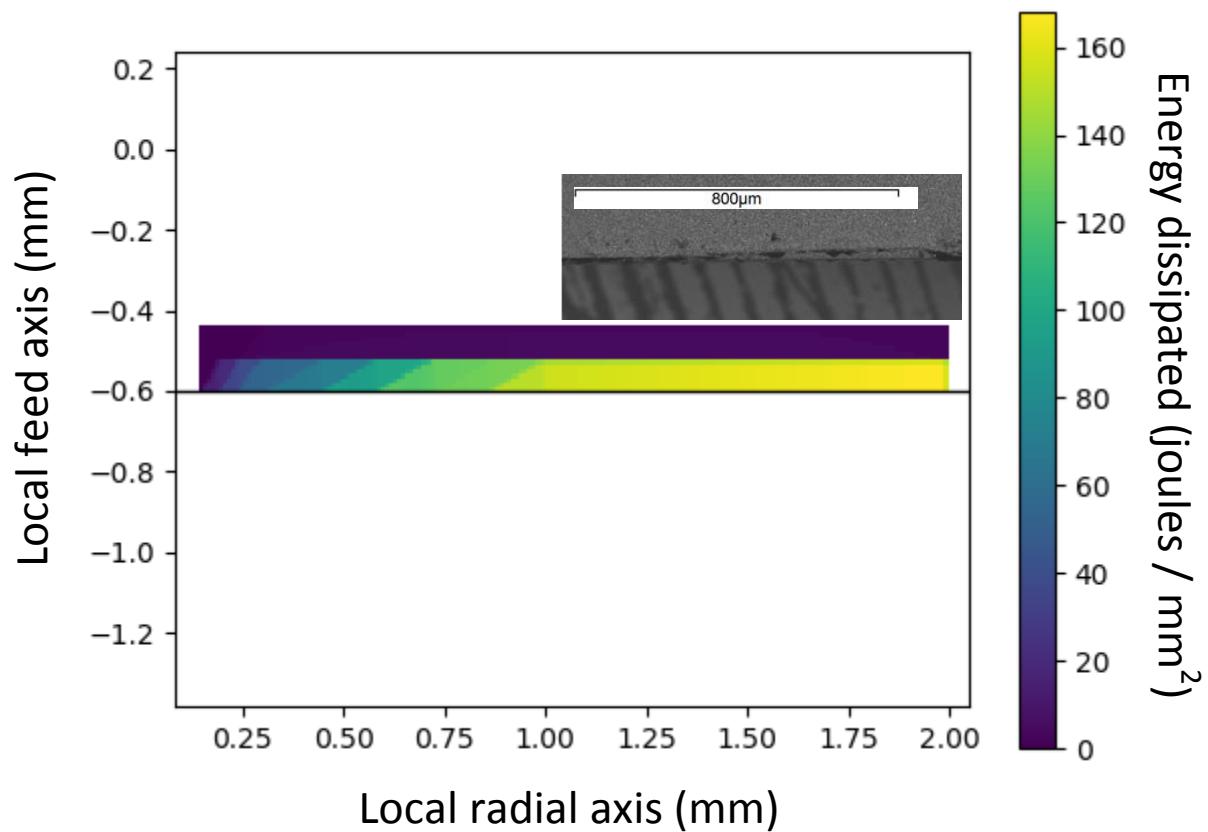


Figure 10-13 – Cutter duty cycle heat map (high zoom)

Also included in figure 10-13 is an overlay of the tapered wear profile observed on insert 16, shown in figure 5-19. This overlay has been scaled to match the plot axes.

Figure 10-13 clearly shows an increasing energy dissipation density from left to right. Between radial axis 1.00 mm and 2.00 mm, this increase is more gradual. This section represents the flat top of the valve seat and increases with radial distance due to the increase in distance the cutter must travel (as illustrated in figure 5-9). The left hand side of the plot shows a sharper change in energy dissipation as expected due to the tapered throat of the seat.

It is not possible to link these energy dissipation density data directly to temperature as the energy dissipation vectors and rates from the tool tip to the environment are unknown. It is however very likely that the areas of highest energy dissipation density correspond to areas of highest temperature.

The area of highest energy dissipation is on the far right hand side, this correlates well with a taper wear profile across the cutting edge first observed in Chapter Five. The taper wear profile is likely to result from high temperatures at the heavily worn end which act to accelerate chemical wear processes in pcBN (Arsecularatne, Zhang and Montross, 2006).

Chapter Five cited some possible failure modes relating to variations in cutting load across the cutting edge. The production of this duty cycle map could assist end users in reducing the duty cycle gradients along each edge or balancing the duty cycle between multiple cutters in order to reduce wear.

## 10.2 Multiple Cutters

Section 10.1 gave a validation of the model and showed that it was able to simulate the experimental configurations presented in Chapter Seven. In this section, the model is used to evaluate the cutting load imbalance for two additional configurations, in order to determine and compare the radial spindle loading characteristics for a hypothetical multi-angle problem and its proposed solution.

### 10.2.1 Configuration

The simulations presented are configured identically to those given in section 10.1 and were run to the same 0.7 mm depth of cut, but with the feed, speed, lubrication and cutter configurations given in table 10-3.

Configuration four is based on a two-cutter configuration presented by Lacerda and Siqueira, 2012 (shown in figure 1-7, b) and configuration five is based on a proposed solution configuration using three cutters.

Additionally, a control simulation was run as a further validation of the model with two perfectly opposing cutters. The expectation with two opposing cutters is that the combined radial imbalance of the spindle when both cutters are engaged is zero.

Parameter	Configuration 4		Configuration 5			Configuration 6 (control)	
Target spindle speed	1666.7 RPM		1666.7 RPM			1666.7 RPM	
Feed rate	0.08 mm / rev		0.08 mm /rev			0.08 mm /rev	
Lubrication	Dry		Dry			Dry	
Number of cutters	2		3			2	
Cutter reference	A	B	C	D	E	F	G
Cutter inclination (cone angle)	0°	45°	0°	45°	45°	10°	10°
Cutter angle (-ve about spindle axis)	10°	190°	90°	230°	10°	10°	190°

Table 10-3 – Multiple cutter configurations

Figure 10-14 gives the simulation coordinate system. Radial, feed and rake loads are output by the simulation both relative to the cutter local coordinate system and relative to the simulation coordinate system. The simulation also outputs the cumulative spindle load for all simulation increments.

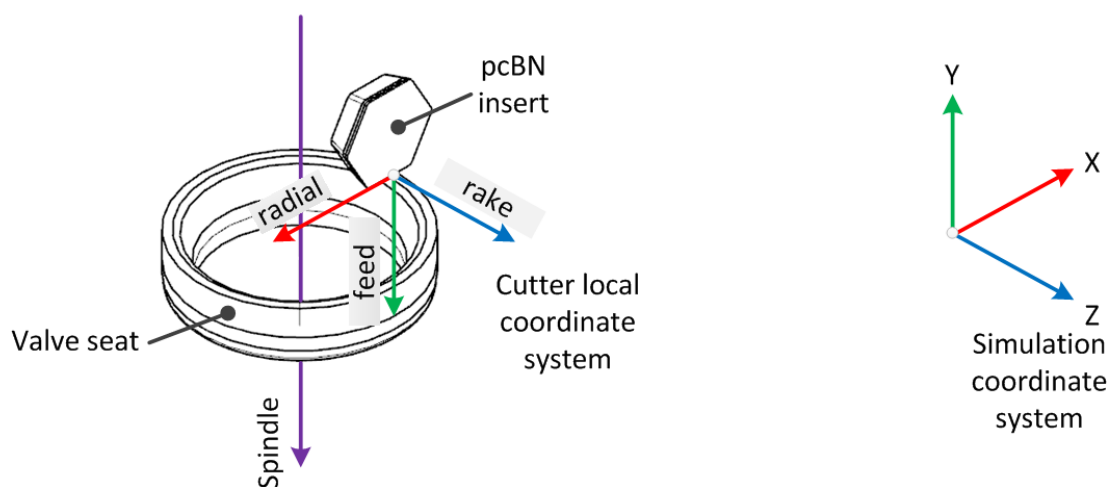


Figure 10-14 – Local cutter, and global simulation coordinate systems

Cutters are placed according to their configuration as specified in table 10-3, following the order of operations as defined in section 8.6.4. For example, the first cutter, A) has an inclination of 0° rotated about the cutter's local rake axis and a spindle angle offset rotated negative 10° about the spindle axis.

Figure 10-15 shows the simulation domain at the start of simulation for all configurations. As the figure shows, only a sub-segment of the valve seat was used to determine loads for each cutter.

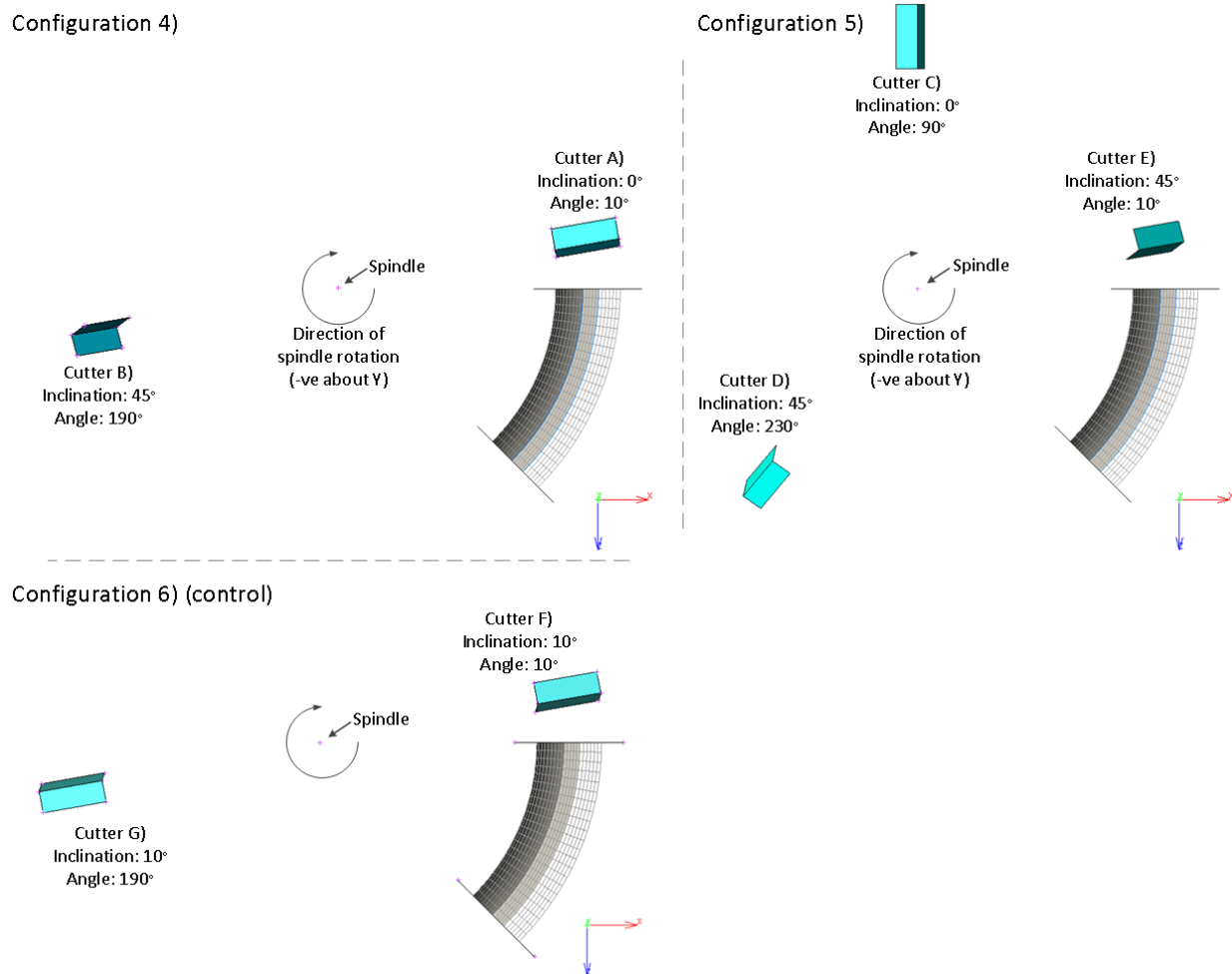


Figure 10-15 – Cutter configurations as viewed MSC Marc

During periods where a cutter was moving at cutting depth, but not in contact with the sub-segment, loads were linearly interpolated between clusters of data generated during periods of contact. For example, figure 10-16 shows the rake force for cutter B) in black and the linearly interpolated loads between clusters for every increment in blue.

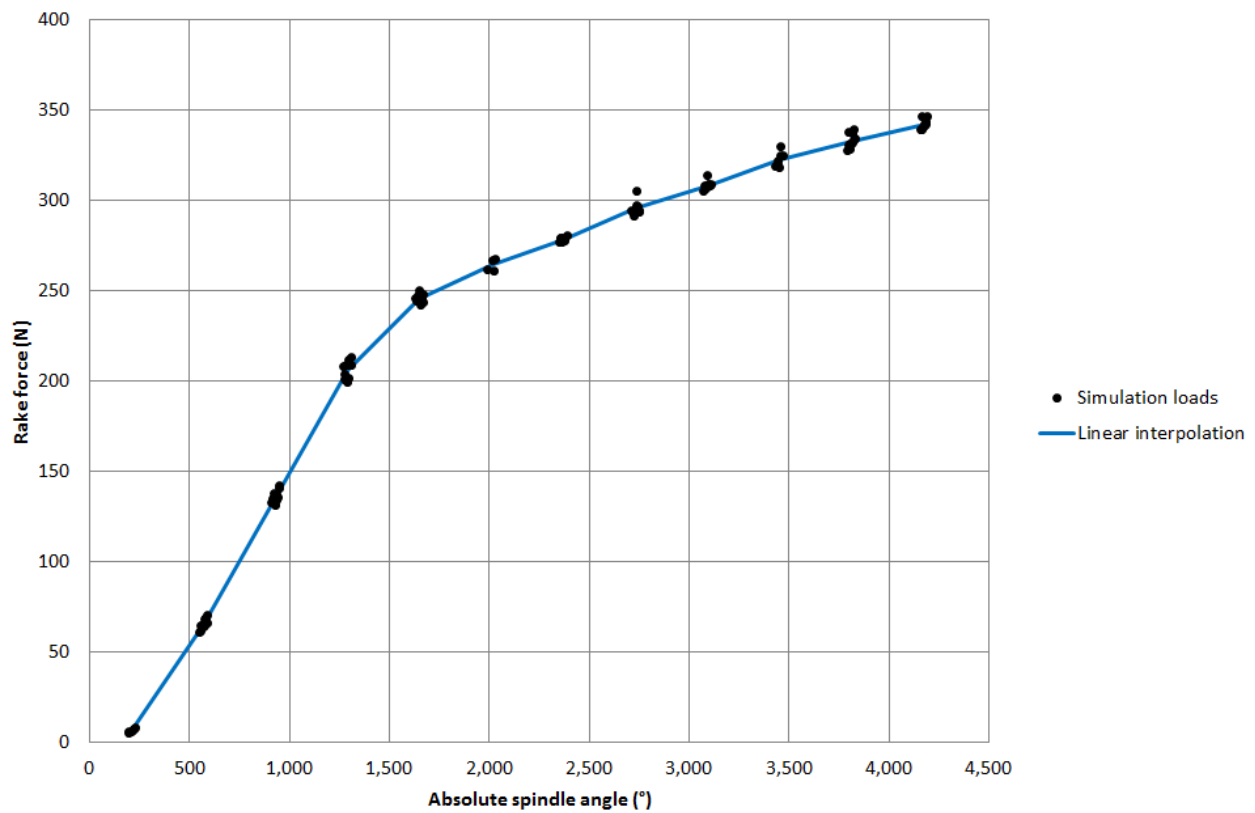


Figure 10-16 – Load interpolation for intermittent cutting

### 10.2.2 Validation of Balanced Configuration

Figure 10-17 a) shows the X-axis spindle loading for cutters F) and G) in control configuration six. The combined spindle loading in both the X and Z axes for cutters F) and G) is given in b). Markers show the engagement times of cutters as labelled. As the data show, in a), the load applied to the spindle by each cutter is equal, but 180° out of phase. The X-Z spindle loading given in b) which represents the combined load applied by both cutters, shows that after both cutters engage, the total radial load on the spindle drops to near zero.

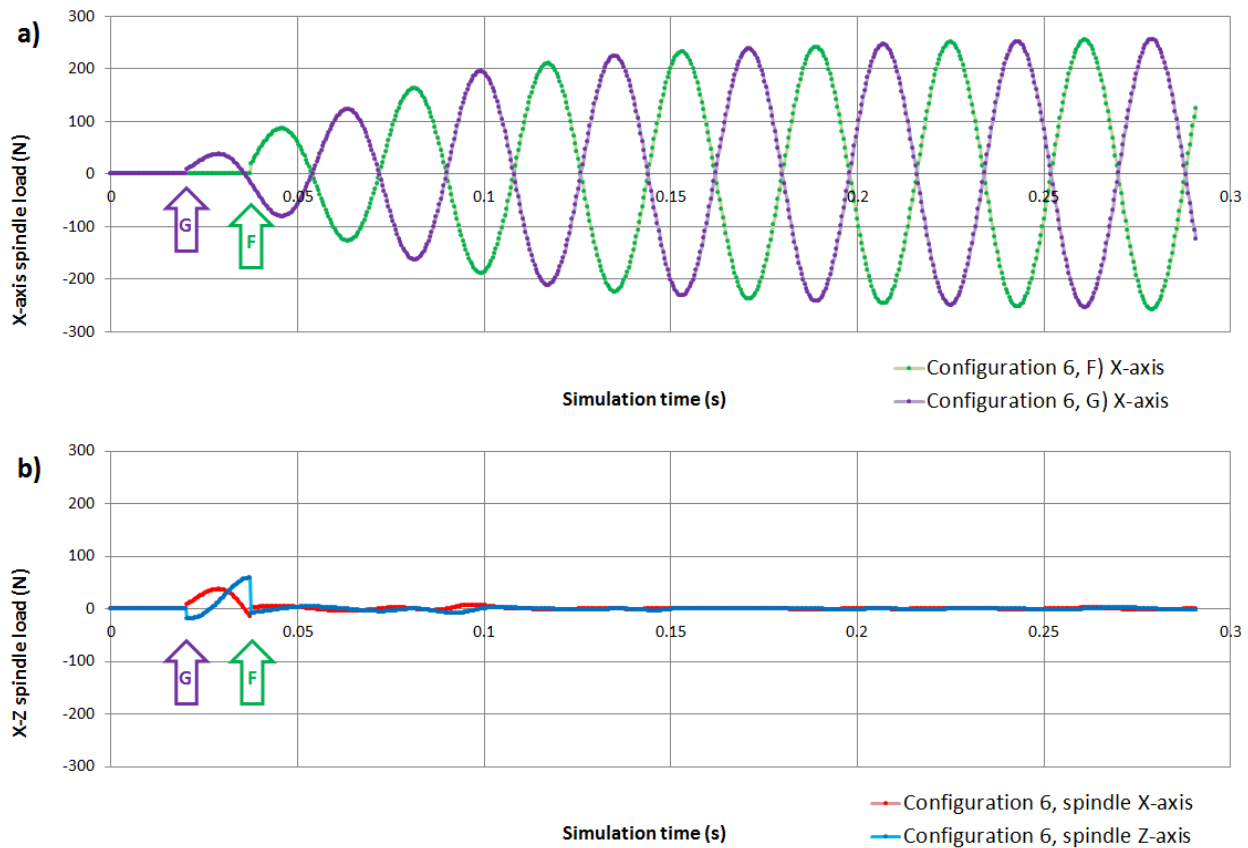


Figure 10-17 – a) Individual and b) combined spindle loading for configuration 6)

Before settling at approximately 0.1 seconds, there is some error in the spindle load. This error comes from discretisation errors in the mesh where the geometry is curved. These errors slightly affect the width of cut and thus the calculated load. The data also misrepresent the engagement profile (the region of data after engagement of G) and before engagement of F)). This is due to the way in which load data is interpolated for periods where the cutter is not in contact with the sub-segment mesh. Interpolation needs at least one historic non-zero cutter load to function. Overall, these data show the simulation is correctly evaluating a scenario in which radial loading is expected to be near zero.

### 10.2.3 Radial Imbalance Reduction Study

Configuration five represents a proposed solution to the large imbalance expected in configuration four. In this solution a third cutter is added to counter the cutter likely to generate the highest radial load (the 45° cutter). In the solution proposed by Lacerda and Siqueira, 2012, (shown in figure 1-7, d), two additional inserts were added. Whilst their solution has the potential to reduce radial loading to negligible levels (providing near identical engagement times), it does not consider difficulties in developing a tool to hold four inserts and provide adequate coolant and space for chip ejection. Configuration five cannot reduce radial imbalance to zero, but if it can reduce it by any amount, it may help to extend the life of cutting inserts by reducing the amplitude of vibration.

Figure 10-18 shows configuration four spindle loading in the X-Z plane for all increments, where each point represents a unique increment and points are joined sequentially according to increment number. The first increment, at  $t=0$  seconds, is at coordinate 0,0 (no load) since all simulations start with no cutter-workpiece contact. Each datapoint thereafter is separated from the preceding point by  $4E-4$  seconds. As the figure shows, cutter A) rapidly spirals out to a radial imbalance of approximately 240 N and consistently holds this magnitude throughout the simulation. This is expected as the 45° cutter, B), has the effect of regulating A)'s width of cut throughout the simulation, despite what would otherwise be a gradually increasing width of cut for A), assuming B) was not present.

Cutter B) has a much more dramatic loading profile that converges gradually towards 400 N at the full depth of cut. Again, this is expected as the cutter's 45° inclination directs load away from the spindle thrust direction (it's most rigid axis) and applies it as a bending load against the spindle.

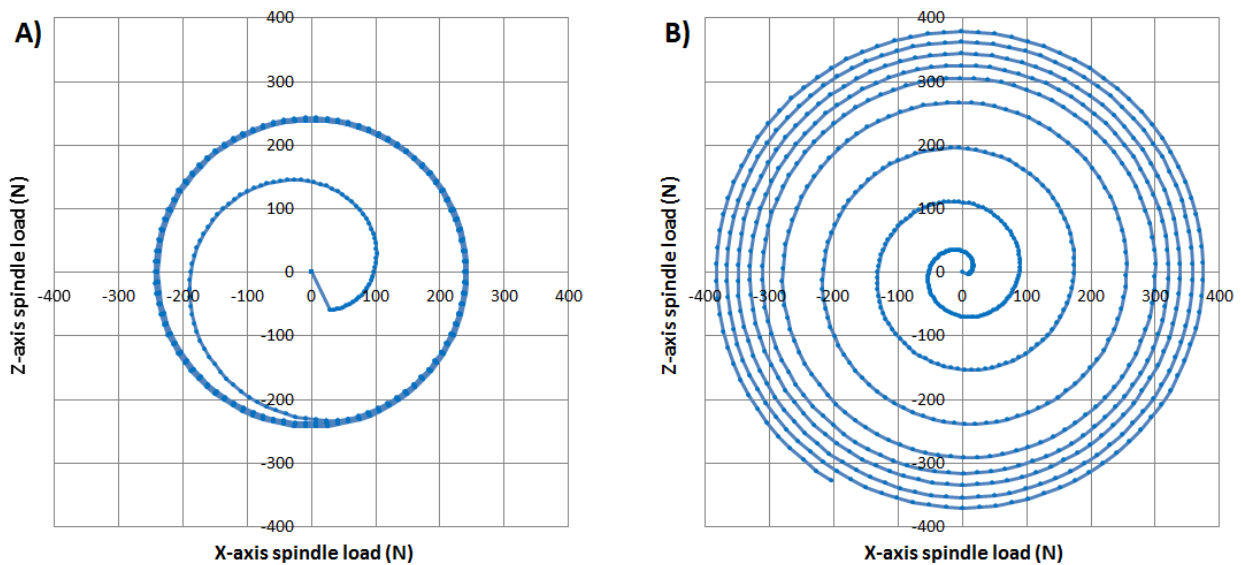


Figure 10-18 – Individual spindle loads for cutters A) and B)



Figure 10-19, shows like data from configuration five for cutters C), D) and E). Cutter C), representing the single 0° inclination cutter shows a similar loading profile to that of A) in figure 10-18. As expected, sharing the 45° inclination cutting load between two cutters, D), and E) has the effect of reducing the peak load on each when compared to B) in figure 10-18.

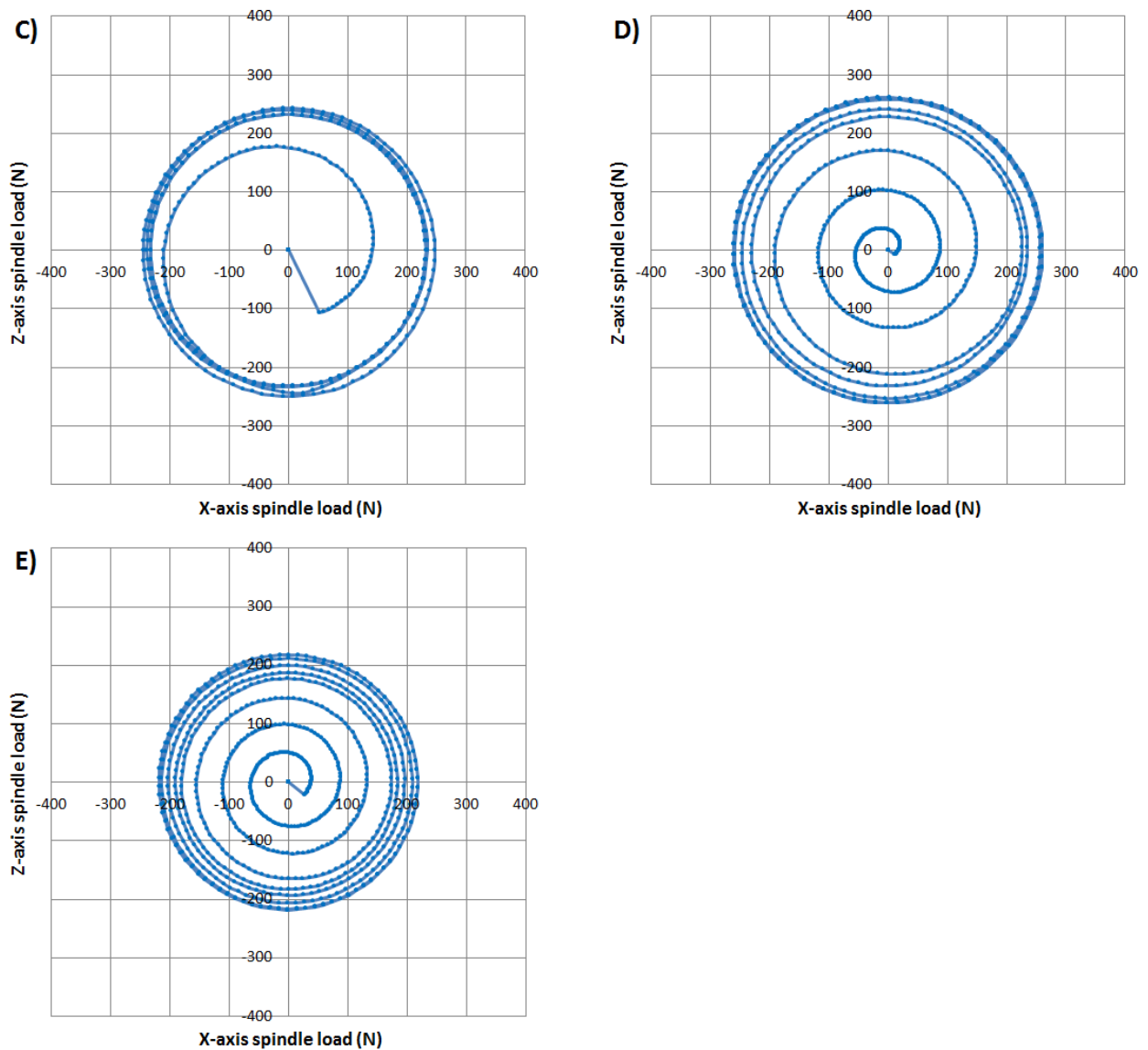
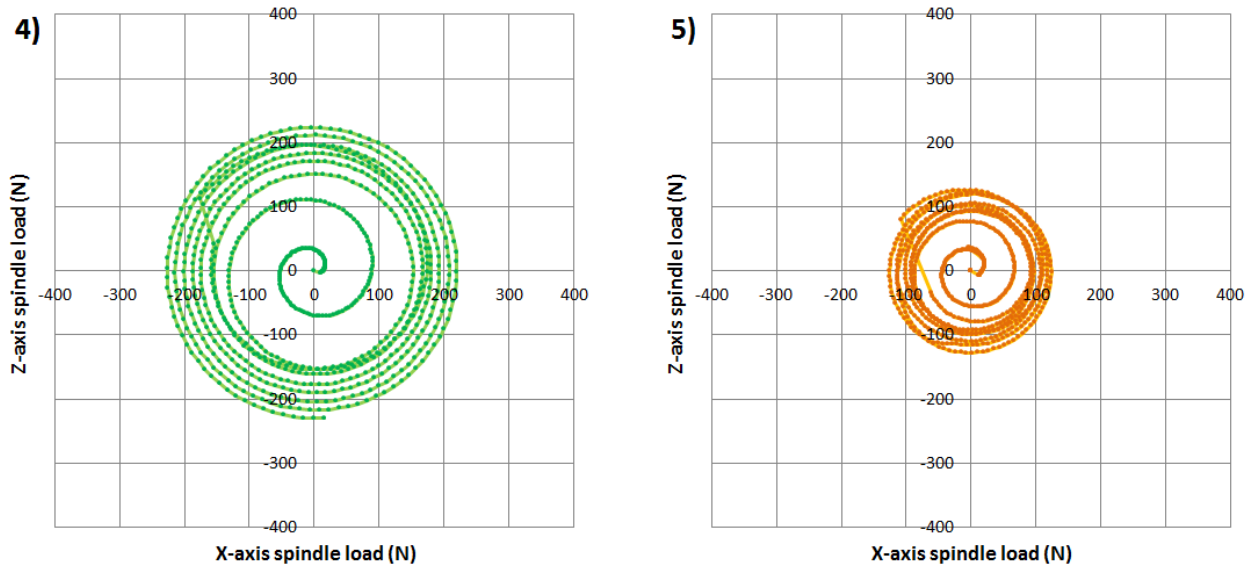


Figure 10-19 – Individual spindle loads for cutters C), D) and E)

Figure 10-20 shows the combined spindle loading for configurations four and five. As the data show, both configurations exhibit a similar loading profile, however, configuration five converges to a maximum radial imbalance near 120 N, whereas configuration four continues growing until the final increment of the simulation where it reaches 220 N.

The 45% decrease in maximum radial load in configuration five vs. that of configuration four shows that the cutting insert configuration and layout has a considerable effect on radial imbalance.



*Figure 10-20 – Total spindle loading for configurations 4) and 5)*

The combined loading data features several abrupt corrections as cutters sequentially come into contact with the workpiece. Although not considered in this work, the timing and magnitude of these artefacts may also help tooling designers select for configurations that minimise the magnitude of sudden changes in radial imbalance, in order to reduce the probability of nucleating chatter.

Figure 10-21 shows spindle loading comparisons for configurations four and five in the X and Z planes respectively. The time of engagement is indicated for all five cutters. As the data show, each time a new cutter engages, it affects both the phase and magnitude of radial loading. The combined radial loading of configuration five is consistently lower than that of configuration four in both the X and Z planes.

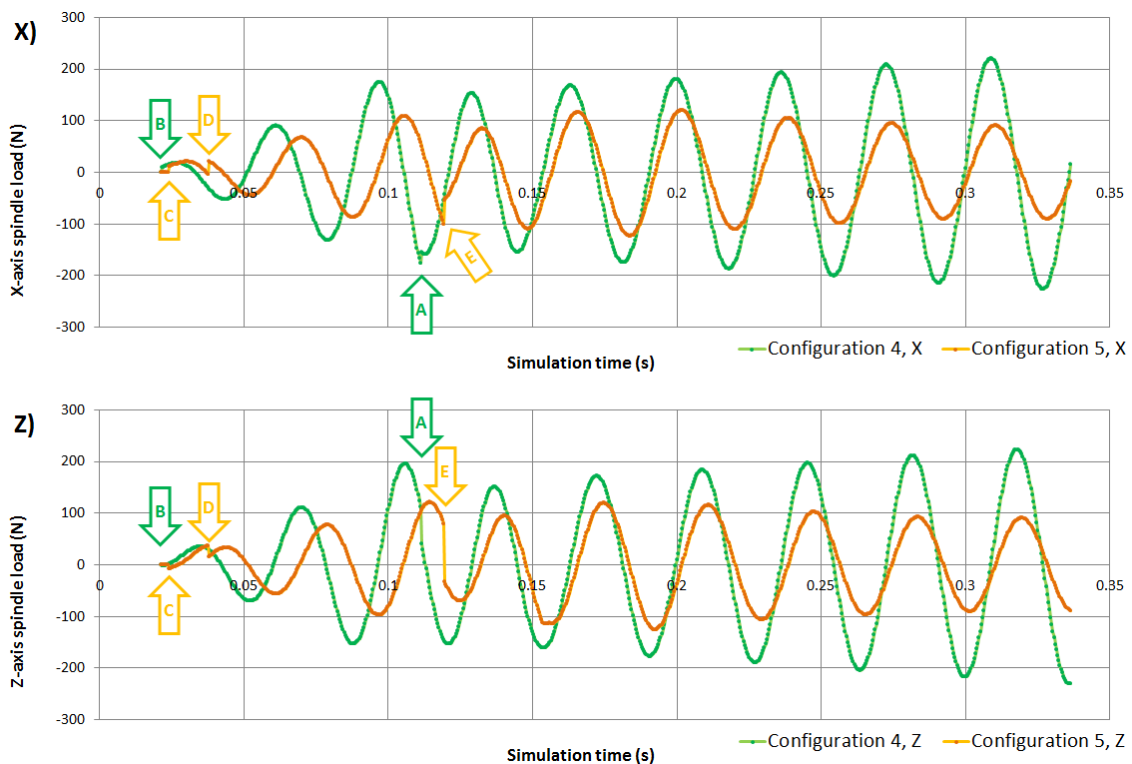


Figure 10-21 – X & Z plane spindle loading comparisons for configurations 4) and 5)

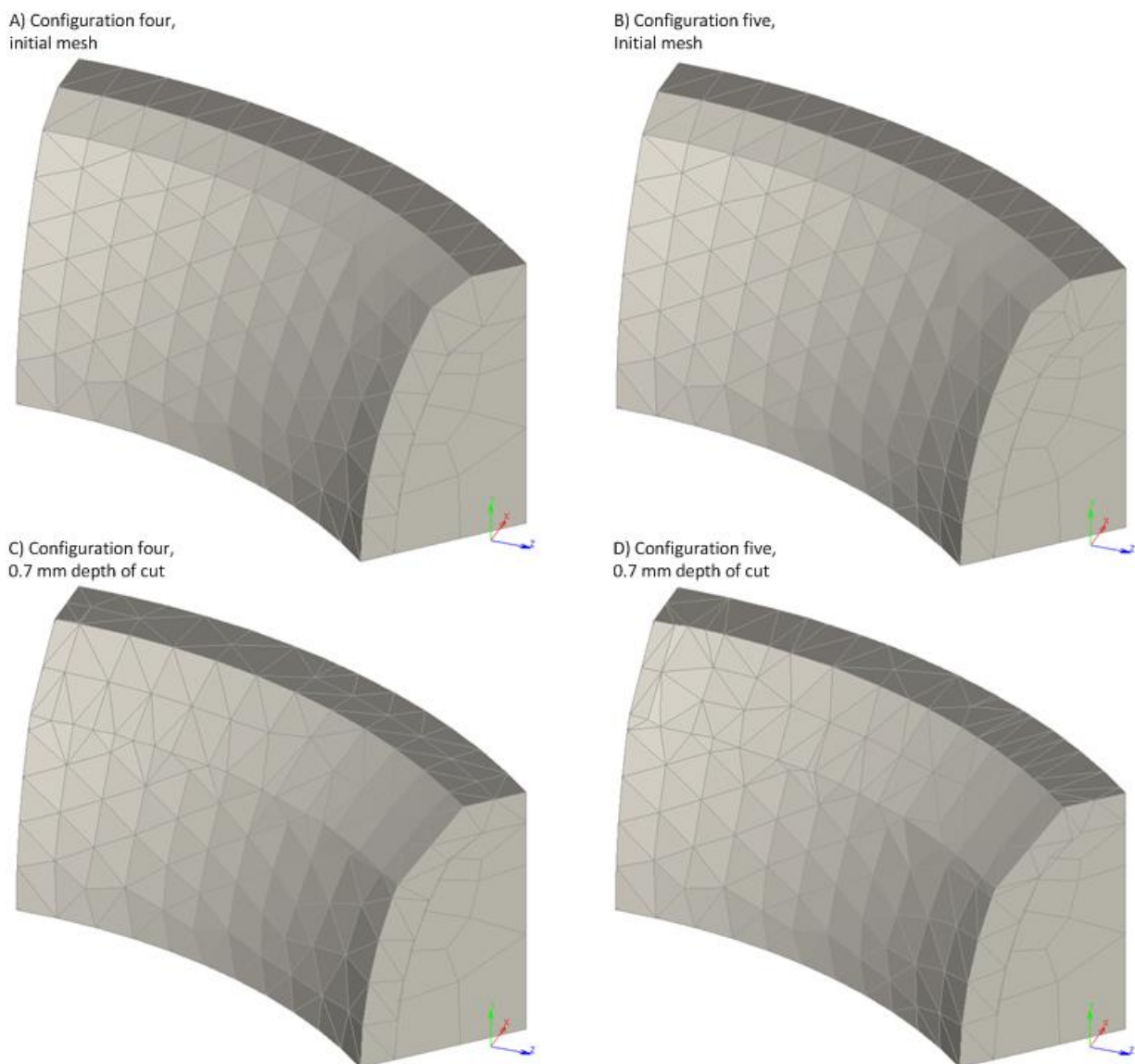
In an ideal configuration, there would be no radial loading. In both configurations four and five, the spindle loading is cyclical with the load vector shifting dramatically around the spindle axis as it rotates. An equivalent, but opposite reaction load is applied to the valve seat itself. The magnitude of these loads must be borne by the spindle, cylinder head and fixture structures. Insufficient stiffness in any of these structures can lead to excessive relative displacement between the tool and workpiece and out-of-round error as shown in Chapter Six. In extreme cases, the radial imbalance can also cause resonance within surrounding structures which can lead to excessive vibration and tool damage.

Lacerda and Siqueira, 2012 show that excessive radial imbalance can accelerate wear of the cutting edge and destroy the workpiece surface. Likewise Rocha *et al.*, 2004 identify the vulnerability of pcBN to chipping due to its low fracture toughness. Evidence of tool chipping was shown throughout Chapter Five. Tool configuration comparisons such as the one presented in this section can help designers configure tools to minimize reciprocal loading in the cutting plane, thus reducing vibration and the probability of tool damage.

#### 10.2.4 Simulation Stability

Both simulations ran beyond the required depth of cut of 0.7 mm and were terminated at a final depth of cut of 0.98 mm. Computation time for configurations four and five was 674.8 seconds and 981.6 seconds respectively.

Figure 10-22 shows A) the condition of the mesh prior to simulation and C) after a final depth of cut of 0.7 mm for configuration four. Likewise, B) shows the initial and D) final state for configuration five. A) and B) show both simulations start with a uniform mesh that approximates curvature well. C) shows that for configuration four, quality and curvature representation has been maintained well through to the final iteration. The same is largely true for configuration five, as shown in D) although there are fewer Delaunay elements as can be seen at the extremes of the 0° and 45° surfaces. Both meshes remain free from needles, caps and wedges.



*Figure 10-22 – Mesh condition at start and end of simulation for both configurations*

### 10.3 Remeshing Algorithm

The previous chapter dealt with the technical implementation of a tetrahedral meshing algorithm based on the ‘gift-wrapping’ technique. As highlighted in that chapter, the algorithm developed cannot handle Schönhardt’s polyhedra and is not guaranteed to generate a mesh in other circumstances, despite the many contingency strategies implemented in the algorithm developed for this work. It is not possible to predict with certainty whether or not a given set of input parameters will guarantee successful mesh generation for every increment in the simulation. Meshing failures are therefore exceptionally troublesome since they can result in lost time and are not possible to resolve from careful parameter selection alone.

The meshing algorithm developed for this project is generally very robust, however it will occasionally fail. Failure is almost always indicated by the exception *“No valid point to complete tetrahedral.”* This error message is raised when a facet remains open, but there were no points in front of it that could be used to complete a tetrahedron on the facet. This error message only ever occurs after all recovery strategies have been trialled (as described in Chapter Nine).

There is normally a weak relationship between certain types of geometry and the probability of failure. For example, small facets at sharp angles to neighbour facets in a mesh of otherwise large facets have a higher probability of failure as the resulting tetrahedron will contain both small and large facets, serving as a bridge between dense and coarse elements. This characteristic of the meshing algorithm is particularly relevant as the mesh will almost always have a collection of elements somewhere that creates a step between depths of cut at the workpiece-cutter interface.

Another difficult geometric feature is surface, almost cap-like, elements on the inner edge of the valve seat ring. Although creation of these types of elements is normally avoided, in some cases they are a viable solution necessary to form the curvature of the inner surface of the valve seat ring, where necessary. These types of elements can cause further caps and wedges to form on their sides that face into the workpiece volume.

With these limitations in mind, the following solutions are offered:

- Almost all state of the art meshing algorithms will seed the interior of the volume with nodes. This is an essential step to maintain smooth transitions between element densities and to disrupt the formation of poor aspect ratio elements. In this work, interior seeding was considered using a crude pseudo-random node insertion method that targets abrupt changes in density, however it was found that this method had little affect on stability, but did increase complexity. Interior seeding was ultimately scrapped, however a more targeted method could possibly improve reliability. Interior seeding is not critically important in this work, since the method developed only remeshes a subsection of the parent mesh. As only one or two layers of elements are selected to be replaced, the mesh is generally well protected against the formation of sharp density transitions and poor aspect ratio elements. This is because the sub mesh hull contains many facets from the parent mesh (at the interface between the parent mesh and sub mesh), and has also undergone simplification and optimisation of its own unique facets; and
- Ideally the meshing algorithm should not fail catastrophically. Some obvious resolution strategies may include:
  - Offer the user a chance to change some parameters to recompute a mesh, for example, change the target edge length, discard external caps (if these are the only errors), exporting the mesh for manual repair before reimporting. These features would increase the

- probability of salvaging the bulk of simulation effort, however they would require human intervention to monitor and control.
- Introduce a method that relaxes the natural characteristic of the splitting algorithm to require exact geometry. For example, where a sharp change in surface angle occurs, perhaps a slope can be tolerated if it would avoid catastrophic failure.
  - Where a mesh cannot be formed, the simulation could advance the cutters slightly and try again. To the end user, this would be an insignificant deviation from the specified spindle speed and feed rates, but to the remeshing algorithm, this would present a fundamentally different arrangement of external hull facets that would be as different to the failed facets as they are to the successful sets of facets that proceeded.

## 10.4 Chapter Summary

This chapter presented the model in its final state at the end of this work. As demonstrated, the model is capable of running and outputting a range of results including cutter loads and cutter face duty cycle data.

These cutter load results agree well with experimental data gathered and presented in Chapter Seven. Furthermore, the calculated geometry closely follows the expected depth of cut and the calculated volume of elements removed matches manual measurements taken directly from Marc.

This chapter also showed that the tools developed can be used to compare cutter layouts in order to select a configuration that significantly reduces radial loading. In a pair of trial simulations based on a real problem identified by Lacerda and Siqueira, 2012 and a solution proposed in this work, it was shown that radial imbalance could be reduced by more than 45 % by adding an additional cutting insert.

Various internal aspects of the Fortran program were reviewed in this chapter, including the performance of the tetrahedral mesher developed for this work in terms of stability and quality. Whilst the tetrahedral mesher developed for this work was found to be by no means perfect, it is stable enough under the conditions tested to run to the end of a machining operation when using a sub-segment valve seat model. Furthermore, the quality of elements produced is generally very good, especially since no internal seed nodes are added under any circumstances. This performance is largely due to careful selection of sub mesh elements to remesh and the extensive preconditioning steps taken to simplify the sub mesh as described in Chapter Nine.

# Chapter Eleven – Conclusions

---

## 11.1 Summary of Results

This work identified that multi-angle valve seat cutting tools, although balanced dynamically during manufacture, are not balanced during cutting. This is clear from the imbalance of cutter angles around the tool and varying engagement times. Real world uses of such tools rarely lead to issues until manufacturers attempt to use them with pcBN cutting inserts. pcBN cutting inserts are vulnerable to chipping when exposed to excessive vibration due to their low brittle fracture toughness. Close inspection of damaged pcBN cutting inserts show many defects characteristic of brittle fracture.

A review of literature suggested that pcBN tools can be used, but only in circumstances where vibration is kept low. A simulation model was designed that would be able to model radial loading of the spindle during multi-angle valve seat machining. The intent of the model was to be able to simulate various different cutter configurations in order to identify those with lower radial imbalance during machining.

This work has shown that it is possible to simulate multi-angle valve seat machining with a minimum of computation effort in order to determine the loads applied to the spindle during multi-angle valve seat machining. To achieve this, a cutting force model was developed that relates feed rate and width of cut to cutting force. This model was applied to a finite element model using a vast body of code to parametrically generate simulation files and calculate changes in geometry and cutting forces for each increment.

Although only a sub-segment of a valve seat could be simulated due to on-going issues with the 3D tetrahedral mesher developed for this work, it was shown that the model could simulate cutting and produce results which agree with experimentally gathered data. Using this model, it was shown that different cutter configurations have a substantial effect on spindle loading and that by reorganising and inserting additional inserts, imbalanced spindle loading can be reduced.

## 11.2 Future Research

Future research objectives include both use-based objectives and development objectives. The primary future research objective is to use the model developed to simulate various cutting tool configurations and present an optimised configuration that features significantly reduced radial cutting force imbalance.

Section 9.10 offered a comprehensive overview of a limitation in the code that reduces its reliability. The mesher developed currently has no way of resolving Schönhardt's polyhedra and will crash whenever one is encountered. Section 9.10 laid out one possible strategy for overcoming the issue but potentially other strategies could be developed to improve stability of the mesher. These should be explored to fully realise the potential of the work developed.

With sufficient manufacturer interest, it would be possible to design, simulate and manufacture several alternative configurations and test them on production line representative machines for their effect on stability and tool life. Although the cost of this work would be significant due to the specialised tooling that must be manufactured and the CNC machine time required, the potential benefits of a positive verification could justify the expense to tooling manufacturers and the industries that utilise them. It is envisaged that, with the assistance of multi-angle valve seat machining tool manufacturers, this model can be used to redesign tools to reduce imbalance and extend tool life.

Although this model can predict cutting forces and changes in geometry and will theoretically increase cutting load and remove more material in response to some feed facing cutter perturbation (originating

due to vibration), no claim is made as to the model's ability to predict regenerative chatter. Whilst the foundational mechanism is simulated within the model (increased non-linear loading in response to depth of cut) no experiments were performed to measure spindle flexibility, or chatter, to confirm that the cutter load responds in exactly the same way it does to depth of cut. It is highly unlikely that cutting loads respond the same way under chatter conditions as they do under normal cutting conditions. Despite this, it may be possible to incorporate a model that is capable of predicting chatter and regenerative chatter. However, many further experiments would need to be done in this area. Unlike the cutting force experiment discussed in Chapter Seven, it is unlikely a similar experiment aimed at characterising chatter could be simplified to a point where it would be economically justifiable. This area of research was ignored due to the following reasons:

- The main area of interest for this work was random chipping of polycrystalline cubic boron nitride (pcBN) cutting inserts during multi-angle valve seat machining, observed by Ford to occur under stable cutting conditions where no significant vibration was reported (assuming that vibration would have been audible) and where no evidence of regenerative chatter was found on the seat. Lacerda and Siqueira, 2012, found evidence of chatter in their experiments, however their cutting and workpiece materials were significantly less hard than the ones used by Ford. Furthermore data is not available to draw a comparison between the stiffness of their cutting system vs. that of Ford's. Despite the differences in expression, this work asserts that dynamic imbalance is responsible for both failure modes, thus there is no reason to require regenerative chatter in Fords case to agree with the theory presented by Lacerda and Siqueira.
- Furthermore, simulating the nature of regenerative chatter for some measurable effect (period, amplitude, energy, etc.) is of little interest to tooling manufacturers, since these properties manifest beyond the point of failure. It is infinitely more important to manufacturers to study the physics leading up to the point where regenerative chatter occurs. In cases such as the one presented by Lacerda and Siqueira, this model may help to determine a safe zone, measured in maximum radial imbalance in Newtons, below which regenerative chatter is extremely unlikely to occur. The probability of chatter nucleation shares no relationship with cutter load, however it may share a relationship with radial imbalance.

Chapter Five found evidence of variable wear along the cutter face. The literature review for this work highlighted the importance of cutting edge radius on cutting performance, particularly with regards to cutting loads, the quality of surface finish and vulnerability of the system to chatter. The model developed in this work can accurately track the relative duty cycle of the cutting edge, revealing potential opportunities for optimisation or to guide tool change schedules. It may be possible to add another dimension to the cutting force prediction model developed in Chapter Seven, that takes into account the wear state of the cutting edge, in addition to width and depth of cut. Currently, the model does not adjust cutting force based on wear level, but by extending the governing equations in this way, the simulation could model combinations of cutters with different wear states. However, capturing this data would require considerably more experimental data, since varying spindle speeds and feed rates, must also be repeated for varying known states of cutter wear.



### 11.3 Conclusions

This work set out to help overcome the issue of random pcBN cutting tool failure during multi-angle valve seat machining by developing tools to help reduce radial imbalance. The motivation came from a case study concerning the Ford Sigma and Ford Fox engines. In that case study, manufacturer specifications for pcBN promised to offer superior tool life and greater potential to hold tighter tolerances for longer and thus more economical yields than traditional tungsten carbide alternatives. Ford commonly used pcBN on other operations such as cylinder boring, but had little success with valve seat machining.

This section reviews the objectives set out in section 1.3 as follows:

1. *investigate Ford's cylinder head and fixture geometry and determine whether or not it undergoes resonance at typical valve seat cutting feed rates and speeds;*
2. *design and execute an experiment aimed at capturing specific feed and rake forces for the valve seat cutting operation, using a range of feed rates and spindle speeds for both dry and minimum quantity lubricant (MQL) conditions;*
3. *justify, design and develop a substantial body of code capable of calculating cutting forces for a sub-segment of the valve seat cutting operation at typical feed rates and speeds; and*
4. *test the simulation code by using it to calculate the sub-segment cutting load of a single cutter up to and beyond the typical cutting depth.*

#### 11.3.1 Objective One

A visit to Ford's engine plant in Craiova, Romania was arranged in order to take a first-hand look at the valve seat machining process. For this trip, an accelerometer system was designed specifically to look for low and medium frequency resonance in the valve seat cutting operation at typical feed rates and spindle speeds. Although the cylinder head fixture system had some stability issues (such as inadequate stiffness and excessive flex at high thrust loads) evidence was collected that shows the head does not undergo resonance during valve seat machining.

The accelerometer system was used to test other cutting operations such as valve guide reaming in which resonance was found. Not only did this show that the data capture system was working as expected, it also showed that other parts of the valve seat and guide machining process had stability issues.

#### 11.3.2 Objective Two

Chapter Seven laid out the design of an experiment which used a lathe to simulate the valve seat machining operation using Ford supplied pcBN cutting tools and valve seats. Cutting speeds, feed rates and lubrication settings were selected to encompass the full range of conditions that Ford can use for the operation.

In Chapter Seven, it was shown that cutting velocity has negligible effect on cutting load contrary to initial expectations. The only factors that played a significant role in determining cutting load were width of cut and feed rate.

These data were successfully used to construct a cutting force model for dry and MQL conditions that can predict cutting load for a given width and depth of cut.

#### 11.3.3 Objective Three

**Justify:** Chapter Two introduced the numerous and broad fundamental concepts of cutting processes and discussed the complexities and factors that influence cutting operations such as cutting materials, cutter geometry, cutting fluids, speeds, feed rates, heat and wear.

Chapter Three introduced finite element analysis as a commonly used approach to modelling cutting phenomena. This section presented the fundamentals of finite element modelling, including aspects such as linear vs. non-linear, meshing and mesh refinement, verification and a review of available finite element software options.

Chapter Four discussed some of the ways in which the phenomena discussed in Chapter Two could be modelled by the tools introduced in Chapter Three. Many of the methods presented were commonly used in isolation and presented in literature in the form of simulations covering very limited durations.

The chapter showed that combining the models in order to produce a single 3D numerical model of valve seat machining that can run for multiple passes and multiple iterations was unrealistic. This work suggested that by using simplifications such as a cutting force model and specialised techniques for remeshing, the setup time and computation time required to simulate valve seat cutting could be drastically reduced. Section 8.1 presented a justification for a building a substantial body of code for performing the functions discussed, such as intersection detection, mesh splitting, remeshing and resolving forces.

**Design:** Chapter Eight, section 8.4 presented a comprehensive design for the model, including how a specially developed Python script would build the geometry in MSC Marc according to parametric inputs. Section 8.3 explained how this parametric model would fit together with the simulation engine developed in Chapter Nine.

**Develop:** Chapter Eight, section 8.6 walked through how the parametric model generator script builds the finite element model geometry and sets up the job, load case and mesh configuration.

Chapter Nine, section 9.7 presented a detailed step-by-step explanation of how the Fortran code developed for this work treats each increment, including isolating a sub-section of the mesh, calculating where to split the mesh, generating 3D tetrahedral elements to represent the newly machined surface and resolving cutting forces. Section 9.8 covers in detail how low level operations such as ray-ray intersection and NURBS surface modelling are performed.

#### **11.3.4 Objective Four**

Chapter Ten applied the model to a 30° valve seat sub-segment, cut according to three different configurations given in table 10-1. The testing showed that the simulation produces results which closely approximate both the experimental data and theoretical cutting force model.

Further analysis showed that the algorithm developed was able to keep element and node counts low and reliably generated new tetrahedral meshes for every increment in which the cutter was engaged with the workpiece.

The model was applied to a multi-angle problem raised by Lacerda and Siqueira, 2012, and a proposed solution was developed to reduce radial loading. The simulations performed showed that by introducing a third cutter, maximum radial load could be reduced by 23%.

## **11.4 Novel Contributions**

This work proves that it is possible to simulate the radial imbalance of cutters in a multi-angle valve seat cutting system, by developing a parametric numerical model and bespoke software.

Furthermore, this work shows that it is possible to carry out such a simulation in drastically reduced time compared to traditional finite element methods.

It also offers a novel experimental design to measure the cutting forces acting on a single cutter of a multi-angle valve seat cutting tool. The experiment developed works by recreating an equivalent cutting system using production valve seat blanks and pcBN cutting inserts. The data yielded are used to establish a map that relates lubrication regime, depth of cut and width of cut to cutting force, suitable for use in a numerical model.

It is envisaged that the numerical model developed for this work will enable tooling designers to cost-effectively simulate multi-angle valve seat machining tool designs in future. This will enable designers to compare various cutter layouts and select for those which produce the lowest radial imbalance, leading to more radially balanced configurations with lower vibrational amplitudes and therefore reduced pcBN tool damage. These optimised tool configurations will help automotive manufacturers take full advantage of cutting materials such as pcBN.

# References

---

AMETEK 2009 'LANDCAL - A Range of Temperature Calibration Sources'.

Analog Devices Inc. 2009 'ADXL326 Small, Low Power, 3-Axis  $\pm 16g$  Accelerometer'. Available at: <http://www.analog.com/en/products/mems/mems-accelerometers/adxl326.html>.

Arsecularatne, J. A., Zhang, L. C. and Montross, C. 2006 'Wear and tool life of tungsten carbide, PCBN and PCD cutting tools', *International Journal of Machine Tools and Manufacture*, 465, pp. 482–491. doi: 10.1016/j.ijmachtools.2005.07.015.

Astakhov, V. P. V. 2006 *Tribology of Metal Cutting, Volume 52 (Tribology and Interface Engineering)*. doi: 10.1097/00000542-200203000-00013.

ASTM 2000 'E646 - Standard Test Method for Tensile Strain-Hardening Exponents (n-Values) of Metallic Sheet Materials'.

ASTM 2004 'B783 - Standard Specification for Materials for Ferrous Powder Metallurgy (P/M) Structural Parts'.

Attanasio, A. *et al.* 2008 '3D finite element analysis of tool wear in machining', *CIRP Annals - Manufacturing Technology*, 571, pp. 61–64. doi: 10.1016/j.cirp.2008.03.123.

Barry, J. and Byrne, G. 2001 'Cutting tool wear in the machining of hardened steels. Part II: Cubic boron nitride cutting tool wear', *Wear*, 2472, pp. 152–160. doi: 10.1016/S0043-1648(00)00528-7.

Bell, D. D. *et al.* 1999 'Modeling of the environmental effect of cutting fluid©', *Tribology Transactions*, 421, pp. 168–173. doi: 10.1080/10402009908982204.

Bil, H., Kiliç, S. E. and Tekkaya, A. E. 2004 'A comparison of orthogonal cutting data from experiments with three different finite element models', *International Journal of Machine Tools and Manufacture*, 449, pp. 933–944. doi: 10.1016/j.ijmachtools.2004.01.016.

Bleistahl 2006 'NOVOFER AR20'.

Blumenthal, L. M. 1970 *Theory and applications of distance geometry*.

Boisse, P., Altan, T. and Luttervelt, K. van 2003 *Friction & flow stress in forming & cutting*.

Bölling, C., Kuhne, M. and Abele, E. 2017 'Modeling of process forces with consideration of tool wear for machining of sintered steel alloy for application to valve seat in a combustion engine', *Production Engineering*. Springer Berlin Heidelberg, 114–5, pp. 477–485. doi: 10.1007/s11740-017-0759-y.

Borvik, T. *et al.* 2001 'A computational model of viscoplasticity and ductile damage for impact and penetration', *European Journal of Mechanics, A/Solids*, 205, pp. 685–712. doi: 10.1016/S0997-7538(01)01157-3.

Brehl, D. E. and Dow, T. A. 2008 'Review of vibration-assisted machining', *Precision Engineering*, 323, pp. 153–172. doi: 10.1016/j.precisioneng.2007.08.003.

Brito, R. F., Carvalho, S. R. and Lima E Silva, S. M. M. 2015 'Experimental investigation of thermal aspects in a cutting tool using comsol and inverse problem', *Applied Thermal Engineering*. Elsevier Ltd, 86, pp. 60–68. doi: 10.1016/j.applthermaleng.2015.03.083.

Ceretti, E. *et al.* 2000 'Turning simulations using a three-dimensional FEM code', *Journal of Materials Processing Technology*, 981, pp. 99–103. doi: 10.1016/S0924-0136(99)00310-6.

- Chen, G. *et al.* 2013 'Measurement and finite element simulation of micro-cutting temperatures of tool tip and workpiece', *International Journal of Machine Tools and Manufacture*. Elsevier, 75, pp. 16–26. doi: 10.1016/j.ijmachtools.2013.08.005.
- Choi, J. *et al.* 1993 'Effect of B<sub>2</sub>O<sub>3</sub> and hBN Crystallinity on cBN Synthesis', *Journal of the American Ceramic Society*, 7610, pp. 2525–2528. doi: 10.1111/j.1151-2916.1993.tb03976.x.
- Costes, J. P. *et al.* 2007 'Tool-life and wear mechanisms of CBN tools in machining of Inconel 718', *International Journal of Machine Tools and Manufacture*, 477–8, pp. 1081–1087. doi: 10.1016/j.ijmachtools.2006.09.031.
- Le Coz, G. *et al.* 2012 'Measuring temperature of rotating cutting tools: Application to MQL drilling and dry milling of aerospace alloys', *Applied Thermal Engineering*. Elsevier Ltd, 361, pp. 434–441. doi: 10.1016/j.applthermaleng.2011.10.060.
- Crisfield, M. a 1996 *Nonlinear Finite Element Analysis of Solids and Structures, Non-Linear Finite Element Analysis of Solids and Structures, Second Edition*. doi: 10.1017/CBO9781107415324.004.
- Dan, W. J. *et al.* 2007 'An experimental investigation of large-strain tensile behavior of a metal sheet', *Materials and Design*, 287, pp. 2190–2196. doi: 10.1016/j.matdes.2006.07.005.
- Dassault Systèmes 2014 'Abaqus 6.14 - Abaqus User Subroutines Reference Guide'.
- Department for Business Innovation & Skills 2010 *Manufacturing in the UK: An economic analysis of the sector*.
- DeVries, W. R. 1992 *Analysis of Material Removal Processes*. doi: 10.1007/978-1-4612-4408-0.
- Dhondt, G. 2004 *The Finite Element Method for Three-Dimensional Thermomechanical Applications*. Chichester, UK: John Wiley & Sons, Ltd. doi: 10.1002/0470021217.
- Dobrzański, L. A. 1995 'Effects of chemical composition and processing conditions on the structure and properties of high-speed steels', *Journal of Materials Processing Tech.*, 481–4, pp. 727–737. doi: 10.1016/0924-0136(94)01715-D.
- Dogra, M. *et al.* 2012 'Tool life and surface integrity issues in continuous and interrupted finish hard turning with coated carbide and CBN tools', *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 2263, pp. 431–444. doi: 10.1177/0954405411418589.
- 'Dura-Bond Catalog' 2014.
- Eda, H., Kishi, K. and Hashimoto, H. 1981 'The Newly Developed CBN Cutting Tool', in *Proceedings of the Twenty-First International Machine Tool Design and Research Conference*. London: Macmillan Education UK, pp. 253–258. doi: 10.1007/978-1-349-05861-7\_33.
- Edwards, R. 1993 *Cutting Tools*.
- El-Gallab, M. and Sklad, M. 1998 'Machining of Al/SiC particulate metal-matrix composites Part I: Tool performance', *Journal of Materials Processing Technology*, 831–3, pp. 151–158. doi: 10.1016/S0924-0136(98)00054-5.
- FORD 2016a '[Ford confidential internal document / 1.0L I3 Fox GTDI Upgraded Head Valve Seat Machining Deformation Analysis / 2016MY]'.
- FORD 2016b '[Ford confidential internal document / Cylinder Head / WSS-M2A178-A3]'.
- FORD 2016c '[Ford confidential internal document / ZPL Fixture Model / Steel 16MnCr5 / 016ZE00AO10001]'.

- FORD [Praveen. T] 2016 '[Ford confidential internal document / 1.0L I3 Fox GTDI Upgrade for C344 MA4 Modal Analysis]'.
- Fu, X. and Zheng, S. 2014 'New approach in dynamics of regenerative chatter research of turning', *Communications in Nonlinear Science and Numerical Simulation*. Elsevier B.V., 1911, pp. 4013–4023. doi: 10.1016/j.cnsns.2014.04.003.
- Fujimoto, M. and Ichida, Y. 2008 'Micro fracture behavior of cutting edges in grinding using single crystal cBN grains', *Diamond and Related Materials*, 177–10, pp. 1759–1763. doi: 10.1016/j.diamond.2008.03.008.
- Giasin, K. *et al.* 2017 '3D Finite Element Modelling of Cutting Forces in Drilling Fibre Metal Laminates and Experimental Hole Quality Analysis', *Applied Composite Materials*. Applied Composite Materials, 241, pp. 113–137. doi: 10.1007/s10443-016-9517-0.
- Guerra Silva, R. *et al.* 2015 'Finite element modeling of chip separation in machining cellular metals', *Advances in Manufacturing*, 31, pp. 54–62. doi: 10.1007/s40436-015-0099-0.
- Gurson, A. L. 1977 'Continuum theory of ductile rupture by void nucleation and growth: Part 1 - yield criteria and flow rules for porous ductile media', *Journal of Engineering Materials and Technology, Transactions of the ASME*, 991, pp. 2–15. doi: 10.1115/1.3443401.
- Hajmohammadi, M. S., Movahhedy, M. R. and Moradi, H. 2014 'Investigation of thermal effects on machining chatter based on FEM simulation of chip formation', *CIRP Journal of Manufacturing Science and Technology*. CIRP, 71, pp. 1–10. doi: 10.1016/j.cirpj.2013.11.001.
- Hambli, R. 2001 'Comparison between Lemaitre and Gurson damage models in crack growth simulation during blanking process', *International Journal of Mechanical Sciences*, 4312, pp. 2769–2790. doi: 10.1016/S0020-7403(01)00070-4.
- Harris, T. K., Brookes, E. J. and Taylor, C. J. 2004 'The effect of temperature on the hardness of polycrystalline cubic boron nitride cutting tool materials', *International Journal of Refractory Metals and Hard Materials*, 222–3, pp. 105–110. doi: 10.1016/j.ijrmhm.2004.01.004.
- Hidnert, P. 1937 'Thermal expansion of cemented tungsten carbide', *Journal of Research of the National Bureau of Standards*, 181, p. 47. doi: 10.6028/jres.018.025.
- Hoyle, G. 1988 *High Speed Steels*.
- Huang, S. *et al.* 2008 'Detecting tool breakage using accelerometer in ball-nose end milling', *2008 10th International Conference on Control, Automation, Robotics and Vision, ICARCV 2008*. IEEE, December, pp. 927–933. doi: 10.1109/ICARCV.2008.4795642.
- Huang, Y. and Liang, S. Y. 2005 'Effect of cutting conditions on tool performance in CBN hard turning', *Journal of Manufacturing Processes*, 71, pp. 10–16. doi: 10.1016/S1526-6125(05)70077-3.
- Iglesias, A. *et al.* 2016 'Analytical expressions for chatter analysis in milling operations with one dominant mode', *Journal of Sound and Vibration*. Elsevier, 375, pp. 403–421. doi: 10.1016/j.jsv.2016.04.015.
- Irish, S. and Simmons, R. 2009 'Finite Element Modeling Continuous Improvement: Validity Checks', in *Finite Element Modeling Continuous Improvement*. Available at: <https://femci.gsfc.nasa.gov/index.html>.
- Johnson, G. R. and Cook, W. H. 1983 'A constitutive model and data for metals subjected to large strains, high strain rates and high temperatures.pdf', *International Symposium on Ballistics*, pp. 541–547.
- Kato, H., Shintani, K. and Sumiya, H. 2002 'Cutting performance of a binder-less sintered cubic boron nitride tool in the high-speed milling of gray cast iron', *Journal of Materials Processing Technology*, 1272, pp. 217–221. doi: 10.1016/S0924-0136(02)00145-0.

- Klocke, F. 2011 *Manufacturing Processes 1: Cutting, Rwthedition*. doi: 10.1007/978-3-642-11979-8.
- Knight, W. A. and Boothroyd, G. 2005 *Fundamentals of Metal Machining and Machine Tools, Third Edition*. doi: 1574446592.
- König, W. and Neises, A. 1993 'Wear mechanisms of ultrahard, non-metallic cutting materials', *Wear*, 162–16412–21, pp. 12–21. doi: 10.1016/0043-1648(93)90479-6.
- Krar, S. F. and Ratterman, E. 1990 *Superabrasives: Grinding and Machining With Cbn and Diamond*.
- Lacarbonara, W. 2013 *Nonlinear structural mechanics: Theory, dynamical phenomena and modeling, Nonlinear Structural Mechanics: Theory, Dynamical Phenomena and Modeling*. doi: 10.1007/978-1-4419-1276-3.
- Lacerda, H. B. and Siqueira, I. L. 2012 'Blade geometry effects on the boring of valve seats of internal combustion engines', *International Journal of Advanced Manufacturing Technology*, 631–4, pp. 269–280. doi: 10.1007/s00170-012-3905-x.
- Lakshminarayanan, P. A. et. al. 2001 'Solving Inlet Valve Seat Wear Problem in High BMEP Engines', *Symposium on International Automotive Technology 2001*, p. 8. doi: 10.4271/2001-26-0024.
- Lauro, C. H. et al. 2013 'Monitoring the temperature of the milling process using infrared camera', *Scientific Research and Essays*, 723, pp. 1112–1120. doi: 10.5897/SRE12.579.
- Law, A. M. 2001 'How to Build Valid and Credible Simulation Models', in *Proceedings of the Winter Simulation Conference, 2005*. IEEE, pp. 24–32. doi: 10.1109/WSC.2005.1574236.
- Leine, R. I. et al. 1998 'Stick-Slip Vibrations Induced by Alternate Friction Models', *Nonlinear Dynamics*, 161, pp. 41–54. doi: 10.1023/A:1008289604683.
- Lemaitre, J. 1985 'A Continuous Damage Mechanics Model for Ductile Fracture', *Journal of Engineering Materials and Technology*, 1071, pp. 83–89. doi: 10.1115/1.3225775.
- Li, L. et al. 2002 'High speed cutting of Inconel 718 with coated carbide and ceramic inserts', *Journal of Materials Processing Technology*, 1291–3, pp. 127–130. doi: 10.1016/S0924-0136(02)00590-3.
- Liew, W. Y. H., Ngoi, B. K. A. and Lu, Y. G. 2003 'Wear characteristics of PCBN tools in the ultra-precision machining of stainless steel at low speeds', *Wear*, 2543–4, pp. 265–277. doi: 10.1016/S0043-1648(03)00002-4.
- Lin, Z. C. and Chen, D. Y. 1995 'A study of cutting with a CBN tool', *Journal of Materials Processing Tech.*, 491–2, pp. 149–164. doi: 10.1016/0924-0136(94)01321-Q.
- Liu, Z. Q., Wan, Y. and Ai, X. 2004 'Recent Developments in Tool Materials for High Speed Machining', *Materials Science Forum*, 471–472, pp. 438–442. doi: 10.4028/www.scientific.net/MSF.471-472.438.
- Liujie, X. et al. 2007 'Optimisation of chemical composition of high speed steel with high vanadium content for abrasive wear using an artificial neural network', *Materials and Design*, 283, pp. 1031–1037. doi: 10.1016/j.matdes.2005.10.015.
- LoRusso, J. A. et al. 1984 'Electrohydraulic gas sampling valve', *Review of Scientific Instruments*, 555, pp. 786–792. doi: 10.1063/1.1137818.
- Ludwik, P. 1909 *Elemente der Technologischen Mechanik, Elemente der Technologischen Mechanik*. doi: 10.1007/978-3-662-40293-1.
- MacNeal and Richard 1993 *Finite Elements: Their design and performance*.

- Majeed, A., Iqbal, A. and Lv, J. 2018 'Enhancement of tool life in drilling of hardened AISI 4340 steel using 3D FEM modeling', *International Journal of Advanced Manufacturing Technology*. The International Journal of Advanced Manufacturing Technology, 955–8, pp. 1875–1889. doi: 10.1007/s00170-017-1235-8.
- Malakizadi, A. *et al.* 2017 'Influence of friction models on FE simulation results of orthogonal cutting process', *International Journal of Advanced Manufacturing Technology*, 889–12, pp. 3217–3232. doi: 10.1007/s00170-016-9023-4.
- Malakizadi, A., Sadik, I. and Nyborg, L. 2013 'Wear mechanism of CBN inserts during machining of bimetal aluminum-grey cast iron engine block', *Procedia CIRP*. Elsevier B.V., 8, pp. 188–193. doi: 10.1016/j.procir.2013.06.087.
- Maranhão, C. and Paulo Davim, J. 2010 'Finite element modelling of machining of AISI 316 steel: Numerical simulation and experimental validation', *Simulation Modelling Practice and Theory*. Elsevier B.V., 182, pp. 139–156. doi: 10.1016/j.simpat.2009.10.001.
- Marusich, T. D. and Ortiz, M. 1995 'Modelling and simulation of high-speed machining', *International Journal for Numerical Methods in Engineering*, 3821, pp. 3675–3694. doi: 10.1002/nme.1620382108.
- Meyers, V. J., Smith, I. M. and Griffiths, D. V. 1989 *Programming the Finite Element Method.*, *Mathematics of Computation*. doi: 10.2307/2008738.
- Micro-Epsilon 2012 'Operating Instructions thermolMAGER TIM'.
- Milton C. Shaw 2005 *Metal cutting principles: Second Edition*, New York: Oxford university press. doi: 10.1016/0025-5408(96)80018-3.
- Mitrofanov, A. V. *et al.* 2005 'Effect of lubrication and cutting parameters on ultrasonically assisted turning of Inconel 718', *Journal of Materials Processing Technology*, 162–163SPEC. ISS., pp. 649–654. doi: 10.1016/j.jmatprotec.2005.02.170.
- Moller, T. and Trumbore, B. 1998 'Fast, minimum storage ray-triangle intersection', *Doktorsavhandlingar vid Chalmers Tekniska Hogskola*, 1425, pp. 109–115. doi: 10.1145/1198555.1198746.
- Monteiro, S. N. *et al.* 2013 'Cubic boron nitride competing with diamond as a superhard engineering material - An overview', *Journal of Materials Research and Technology*. Korea Institute of Oriental Medicine, 21, pp. 68–74. doi: 10.1016/j.jmrt.2013.03.004.
- Moradi, H. *et al.* 2013 'Forced vibration analysis of the milling process with structural nonlinearity, internal resonance, tool wear and process damping effects', *International Journal of Non-Linear Mechanics*. Elsevier, 54, pp. 22–34. doi: 10.1016/j.ijnonlinmec.2013.02.005.
- Mourad, A., Mourad, B. and Abderrahim, B. 2017 'Measurement and numerical simulation of the cutting temperature in cutting tool during turning operation', *Journal of Engineering Science and Technology*, 125, pp. 1307–1317.
- MSC Software 2016a 'Marc User Documentation Volume A: Theory and User Information', A.
- MSC Software 2016b 'Marc User Documentation Volume D: User Subroutines and Special Routines', D.
- Noor, A. K. 1988 'Parallel Processing in Finite Element Structural Analysis', 241, pp. 225–241.
- Olgun, E., Compton, W. D. and Chandrasekar, S. 2003 'The effect of superimposed low-frequency modulation on lubrication in machining', pp. 727–735.
- Oxley, P. L. B. 1989 *Mechanics of Machining: An Analytical Approach to Assessing Machinability*, *Journal of Applied Mechanics*. doi: 10.1115/1.2888318.



- Özel, T. 2006 'The influence of friction models on finite element simulations of machining', *International Journal of Machine Tools and Manufacture*, 465, pp. 518–530. doi: 10.1016/j.ijmachtools.2005.07.001.
- Pierce, D. *et al.* 2019 'High temperature materials for heavy duty diesel engines: Historical and future trends', *Progress in Materials Science*. Elsevier Ltd, 103, pp. 109–179. doi: 10.1016/j.pmatsci.2018.10.004.
- Rao, S. S. 2010 *The Finite Element Method in Engineering*. 5th edn, *The Finite Element Method in Engineering: Fifth Edition*. 5th edn. doi: 10.1016/C2009-0-04807-7.
- Roberts, R. B., White, G. K. and Fawcett, E. 1983 'Thermal expansion of Cr and Cr-V alloys', *Physica B+C*, 1191–2, pp. 63–67. doi: 10.1016/0378-4363(83)90167-5.
- Rocha, C. A. *et al.* 2004 'Evaluation of the wear mechanisms and surface parameters when machining internal combustion engine valve seats using PCBN tools', *Journal of Materials Processing Technology*, 1453, pp. 397–406. doi: 10.1016/j.jmatprotec.2003.10.004.
- Saglam, H., Yaldiz, S. and Unsacar, F. 2007 'The effect of tool geometry and cutting speed on main cutting force and tool tip temperature', *Materials and Design*, 281, pp. 101–111. doi: 10.1016/j.matdes.2005.05.015.
- Sato, M., Ueda, T. and Tanaka, H. 2007 'An experimental technique for the measurement of temperature on CBN tool face in end milling', *International Journal of Machine Tools and Manufacture*, 4714, pp. 2071–2076. doi: 10.1016/j.ijmachtools.2007.05.006.
- Schönhardt, E. 1928 'Über Die Zerlegung Von Dreieckspolyedern in Tetraeder', *Mathematische Annalen*, 98, pp. 309–312.
- Scientific Forming Technologies Corporation 2014 'DEFORM 3D Machining Product Brochure'.
- Shams, A. and Mashayekhi, M. 2012 'Improvement of orthogonal cutting simulation with a nonlocal damage model', *International Journal of Mechanical Sciences*. Elsevier, 611, pp. 88–96. doi: 10.1016/j.ijmecsci.2012.05.008.
- Shewchuk, J. R. 2002 'Constrained Delaunay Tetrahedralizations and Provably Good Boundary Recovery', *11th International Meshing Roundtable*, pp. 193–204. doi: 10.1.1.11.9064.
- Si, H. 2015 'TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator', *ACM Transactions on Mathematical Software*, 412, pp. 1–36. doi: 10.1145/2629697.
- Sreejith, P. S. and Ngoi, B. K. A. 2000 'Dry machining: Machining of the future', *Journal of Materials Processing Technology*, 1011, pp. 287–291. doi: 10.1016/S0924-0136(00)00445-3.
- Srikanth, A. and Zabaras, N. 2001 'An updated Lagrangian finite element sensitivity analysis of large deformations using quadrilateral elements', *International Journal for Numerical Methods in Engineering*, 5210, pp. 1131–1163. doi: 10.1002/nme.245.
- Stachowiak, G. W. 2005 *Wear - Materials, Mechanisms and Practice*. Edited by G. W. Stachowiak. Chichester, England: John Wiley & Sons Ltd. doi: 10.1002/9780470017029.
- Stotter, A. 1965 'Exhaust Valve Temperature ■ A Theoretical and Experimental Investigation = J': doi: 10.4271/650019.
- Suh, C. S., Khurjekar, P. P. and Yang, B. 2002 'Characterisation and identification of dynamic instability in milling operation', *Mechanical Systems and Signal Processing*, 165, pp. 853–872. doi: 10.1006/mssp.2002.1497.
- Sutter, G. *et al.* 2003 'An experimental technique for the measurement of temperature fields for the orthogonal cutting in high speed machining', *International Journal of Machine Tools and Manufacture*, 437,

pp. 671–678. doi: 10.1016/S0890-6955(03)00037-3.

Sutter, G. 2005 'Chip geometries during high-speed machining for orthogonal cutting conditions', *International Journal of Machine Tools and Manufacture*, 456, pp. 719–726. doi: 10.1016/j.ijmachtools.2004.09.018.

Svoboda, A., Wedberg, D. and Lindgren, L. E. 2010 'Simulation of metal cutting using a physically based plasticity model', *Modelling and Simulation in Materials Science and Engineering*, 187. doi: 10.1088/0965-0393/18/7/075005.

Tang, L. *et al.* 2019 'Wear performance and mechanisms of PCBN tool in dry hard turning of AISI D2 hardened steel', *Tribology International*. Elsevier Ltd, 1321, pp. 228–236. doi: 10.1016/j.triboint.2018.12.026.

Tatar, K., Rantatalo, M. and Gren, P. 2007 'Laser vibrometry measurements of an optically smooth rotating spindle', *Mechanical Systems and Signal Processing*, 214, pp. 1739–1745. doi: 10.1016/j.ymsp.2006.08.006.

Taylor F.W 1907 *On the art of cutting metals*, *Transactions of ASME*. doi: 10.1109/SUTC.2008.30.

Thepsonthi, T. and Özel, T. 2015 '3-D finite element process simulation of micro-end milling Ti-6Al-4V titanium alloy: Experimental validations on chip flow and tool wear', *Journal of Materials Processing Technology*, 221, pp. 128–145. doi: 10.1016/j.jmatprotec.2015.02.019.

Toh, C. K. 2004 'Vibration analysis in high speed rough and finish milling hardened steel', *Journal of Sound and Vibration*, 2781–2, pp. 101–115. doi: 10.1016/j.jsv.2003.11.012.

Tounsi, N. *et al.* 2002 'From the basic mechanics of orthogonal metal cutting toward the identification of the constitutive equation', *International Journal of Machine Tools and Manufacture*, 4212, pp. 1373–1383. doi: 10.1016/S0890-6955(02)00046-9.

Trent, E. M. and Wright, P. K. 2000 *Metal Cutting*. 4th edn, *Butterworth-Heinemann Ltd.*, 1991,. 4th edn.

Turner, M. J. *et al.* 1956 'Stiffness and Deflection Analysis of Complex Structures', *Journal of the Aeronautical Sciences*, 239, pp. 805–823. doi: 10.2514/8.3664.

Tvergaard, V. and Needleman, A. 1995 'Effects of nonlocal damage in porous plastic solids', *International Journal of Solids and Structures*. Elsevier Science Ltd, 328–9, pp. 1063–1077. doi: 10.1016/0020-7683(94)00185-Y.

Uhlmann, E., Fuentes, J. A. O. and Keunecke, M. 2009 'Machining of high performance workpiece materials with CBN coated cutting tools', *Thin Solid Films*. Elsevier B.V., 5185, pp. 1451–1454. doi: 10.1016/j.tsf.2009.09.095.

Umbrello, D., M'Saoubi, R. and Outeiro, J. C. 2007 'The influence of Johnson-Cook material constants on finite element simulation of machining of AISI 316L steel', *International Journal of Machine Tools and Manufacture*, 473–4, pp. 462–470. doi: 10.1016/j.ijmachtools.2006.06.006.

Vaz, M. *et al.* 2007 'Modelling and simulation of machining processes', *Archives of Computational Methods in Engineering*, 142, pp. 173–204. doi: 10.1007/s11831-007-9005-7.

Ventura, C. E. H., Köhler, J. and Denkena, B. 2013 'Cutting edge preparation of PCBN inserts by means of grinding and its application in hard turning', *CIRP Journal of Manufacturing Science and Technology*, 64, pp. 246–253. doi: 10.1016/j.cirpj.2013.07.005.

Wang, Y. P., Wilkinson, G. B. and Drallmeier, J. A. 2004 'Parametric study on the fuel film breakup of a cold start PFI engine', *Experiments in Fluids*, 373, pp. 385–398. doi: 10.1007/s00348-004-0827-x.

- Wang, Y. S. *et al.* 1995 'Wear Mechanisms of Valve Seat and Insert in Heavy Duty Diesel Engine', *SAE Technical Paper*, 412. doi: 10.4271/952476.
- Weisstein, E. W. 2017 *MathWorld - A Wolfram Web Resource - 'Circumsphere'*. Available at: <http://mathworld.wolfram.com/Circumsphere.html>.
- Wentorf Jr., R. H. 1961 'Synthesis of the Cubic Form of Boron Nitride', *The Journal of Chemical Physics*, 343, p. 809. doi: 10.1063/1.1731679.
- Werschmoeller, D. and Li, X. 2010 'Measurement of tool internal temperatures in the tool-chip contact region by embedded thin film micro thermocouples', *ASME 2010 International Manufacturing Science and Engineering Conference, MSEC 2010*. Elsevier Ltd, 12, pp. 371–377. doi: 10.1115/MSEC2010-34176.
- White, G. K. 1965 'Thermal expansion of magnetic metals at low temperatures', *Proceedings of the Physical Society*, 861, pp. 159–169. doi: 10.1088/0370-1328/86/1/320.
- Wilcoxon 2018 'Low frequency machinery monitoring : measurement considerations'.
- Wong-Ángel, W. D. *et al.* 2014 'Effect of copper on the mechanical properties of alloys formed by powder metallurgy', *Materials and Design*, 58, pp. 12–18. doi: 10.1016/j.matdes.2014.02.002.
- Wriggers, P. 2005 *Adaptive methods for contact problems, CISM International Centre for Mechanical Sciences, Courses and Lectures*. doi: 10.1007/3-211-38060-4\_6.
- Xie, J. Q., Bayoumi, A. E. and Zbib, H. M. 1998 'FEA modeling and simulation of shear localized chip formation in metal cutting', *International Journal of Machine Tools and Manufacture*, 389, pp. 1067–1087. doi: 10.1016/S0890-6955(97)00063-1.
- Yagawa, G. and Shioya, R. 1993 'Parallel finite elements on a massively parallel computer with domain decomposition', *Computing Systems in Engineering*, 44–6, pp. 495–503. doi: 10.1016/0956-0521(93)90017-Q.
- Zhao, T. *et al.* 2017 'Effect of cutting edge radius on surface roughness and tool wear in hard turning of AISI 52100 steel', *International Journal of Advanced Manufacturing Technology*. The International Journal of Advanced Manufacturing Technology, 919–12, pp. 3611–3618. doi: 10.1007/s00170-017-0065-z.
- Zheng, J. *et al.* 2016 'An improved local remeshing algorithm for moving boundary problems', *Engineering Applications of Computational Fluid Mechanics*, 101, pp. 403–426. doi: 10.1080/19942060.2016.1174888.
- Zhou, J. M., Andersson, M. and Ståhl, J. E. 2004 'Identification of cutting errors in precision hard turning process', *Journal of Materials Processing Technology*, 153–1541–3, pp. 746–750. doi: 10.1016/j.jmatprotec.2004.04.331.
- Zhu, D., Zhang, X. and Ding, H. 2013 'Tool wear characteristics in machining of nickel-based superalloys', *International Journal of Machine Tools and Manufacture*. Elsevier, 64, pp. 60–77. doi: 10.1016/j.ijmachtools.2012.08.001.
- Zienkiewicz, O. C., Taylor, R. L. and Zhu, J. Z. 1967 'The Finite Element Method: Its Basis and Fundamentals', in *The Finite Element Method: its Basis and Fundamentals*. 7th edn. Elsevier. doi: 10.1016/B978-1-85617-633-0.00019-8.
- Zoya, Z. . and Krishnamurthy, R. 2000 'The performance of CBN tools in the machining of titanium alloys', *Journal of Materials Processing Technology*, 1001–3, pp. 80–86. doi: 10.1016/S0924-0136(99)00464-1.
- Zsolt János Viharos, Markos, S. and Szekeres, C. 2003 'ANN-based chip-form classification in turning', *XVII IMEKO World Congress*, pp. 1469–1473.

# Appendix A – MarcTools

---

This appendix explains the technical implementation of MarcTools. MarcTools relies heavily on object inheritance and polymorphism to drastically reduce the quantity of code required to generate models in MSC Marc with Python. The Python classes defined below, show an example of how object inheritance can be used to avoid re-implementing methods that are common between multiple subclasses.

```
class RoundGeometry(object):
    def Radius(diameter):
        return radius / 2.0

class Circle(RoundGeometry):
    def Area(diameter):
        return math.pi * self.Radius(diameter) ** 2.0

class Sphere(RoundGeometry):
    def Area(diameter):
        return 4.0 * math.pi * self.Radius(diameter) ** 2.0
```

In this example, **RoundGeometry** inherits the **object** class from Python and contains a method named **Radius**. **Object** is a base class defined by Python, in later Python builds, this inheritance is not necessary, but in Python 2.7 used by Marc, this step is necessary to define a ‘new-style’ class.

**Circle** inherits **RoundGeometry**, and adds one of its own methods – **Area**. Instances of **Circle** have both methods, **Radius** and **Area**, due to the inheritance of **Radius** from **RoundGeometry**. Likewise, **Sphere** can also make use of **Radius**, but must implement a different solution for **Area**.

The following reference gives a list of key classes defined in MarcTools to handle geometry creation in Marc. Of these objects, those that represent some entity in Marc do not create themselves in Marc until a method named **Make** is called. This is to allow the calling code to delay creating the entity in Marc until absolutely necessary, an adaptation which drastically speeds up geometry creation as there is often a significant delay between sending a command to Marc and observing the result. Furthermore, it allows **MarcObject** based objects to be used in construction contexts, without having to involve Marc and add processing overhead (e.g. the origin point of an arc does not need to exist in Marc). If the calling code queries the object’s ID in Marc, or attempts an operation that requires that the corresponding entity exists in Marc, then the objects will silently create their respective entities in Marc automatically.

**MarcObject**(object): Base class for all other classes that represent some entity in Marc.

- **MarcID**(): Method to return the ID of this object, or if this object has not yet been made in Marc, this method makes a corresponding entity in Marc first, then returns the ID of the entity just made.

**CoordinateBased**(**MarcObject**): Base class for entities defined by a coordinate (nodes and points).

- **\_\_init\_\_**([x=0.0],[y=0.0],[z=0.0]): Method to instantiate this class, and set the x, y and z properties
- x, y & z: properties to represent a point in space.
- **Position**(): Method to return an array of this **CoordinateBased** object's position.
- **Invalidate**(): Method to signal that future calls to **Position** must retrieve the latest position from Marc rather than from the local cache.
- **Distance**(other): Method to return the distance from this **CoordinateBased** object to another **CoordinateBased** object, other.
- **VectorToOther**(other): Method to return the vector from this **CoordinateBased** object to another **CoordinateBased** object, other.

**MarcPoint**(**CoordinateBased**): Represents a point entity in Marc.

- **\_\_init\_\_**([x=0.0],[y=0.0],[z=0.0]): Method to instantiate this class, and set the x, y and z properties.
- **Make**(): Method to create this point as an entity in Marc and return itself.

**MarcNode**(**CoordinateBased**): Represents a node entity in Marc.

- **\_\_init\_\_**([x=0.0],[y=0.0],[z=0.0]): Method to instantiate this class, and set the x, y and z properties.
- **Make**(): Method to create this node as an entity in Marc and return itself.

**MarcCurve**(**MarcObject**): Represents a curve entity (line, arc or circle) in Marc.

- **\_\_init\_\_**([\*points]): Method to instantiate this class with a list of zero or more **MarcPoint** objects.
- **Make**(): Method to make a curve between the two points used to instantiate this class and return itself.
- **MakeArc**(): If this class is instantiated with points, *a*, *b* & *c*. Then this method makes an arc entity in Marc between *b* and *c*, with centre *a* (*a* does not need to exist in Marc).
- **MakeCircle**(radius): If this class is instantiated with point *a*, this method makes a circle entity in Marc with a given radius, using *a* as the centre.
- **MidPoint**(): Returns the average coordinate of all points used to instantiate this class.
- **CurveDivisions**(L1,[L2],[Count]): Creates mesh seeds along this curve according to the following rules.
  - If just L1 is prescribed, spans will have a target length of L1
  - If L1 and L2 are prescribed, spans will have a target length of L1 at one end transitioning to L2 at the other end.
  - If L1, L2 and Count are prescribed, there will be Count spans, where the length of spans at one end is related to the length of spans at the other end by the relationship L1 / L2
- **Mesh**(): Creates line **MarcElement** objects corresponding to line element entities along this curve. If mesh seeds are prescribed, they will be used to set the element edge lengths.

**MarcCurveLoop(MarcObject)**: A collection of **MarcCurve** objects.

- **\_\_init\_\_**([\*curves]): Method to instantiate this class with a list of zero or more **MarcCurve** objects.
- **QuadMesh**(): Method to call **self.Mesh**("quadmesh").
- **TriMesh**(): Method to call **self.Mesh**("trimesh").
- **Mesh**([meshType="quadMesh"]): Method to generate a mesh of type, meshType, that fills this curve loop area. If any of the curves involved have mesh seeds, they will be used in the creation of this area mesh. This method returns a list of **MarcObject** objects corresponding to the **MarcNode** and **MarcElement** objects created during meshing.

**MarcSurface(MarcObject)**: Represents a surface entity in Marc.

- **\_\_init\_\_**([\*points]): Instantiates this class with a list of zero or more **MarcPoint** objects.
- **Make**(): Method to create a surface entity in Marc between the **MarcPoint** objects prescribed when instantiating this class and returns itself.
- **Flip**(): Surfaces are sided (required by Marc to process contact), this method flips the surface so that the surface can be reoriented to face the object it is intended to contact.

**MarcElement(MarcObject)**: Represents an element entity in Marc.

- **\_\_init\_\_**([\*nodes]): Instantiates this class with a list of zero or more **MarcNode** objects.
- **Make**(): Method to make element entities in Marc from **MarcNode** objects prescribed when instantiating this **MarcElement**, depending on the number of **MarcNode** objects prescribed,
  - if two, this method makes a line element entity, and
  - if three, this method makes a triangle element entity, and finally,
  - if four, this method makes a quadrilateral element entity.

Aside from these classes, MarcTools also offers two highly polymorphic functions, **Move** and **Expand**, that control the move and expand tools within the Marc user interface, shown side by side in figure 1.



Figure 1 – Marc Move tool (left) and Expand tool (right)

The primary function of move, as the name implies, is to move any type of entity from one location to another. However, this tool can also be used to rotate, skew, deflate and inflate geometry. Expand is similar to 'extrude' commonly found in computer aided design (CAD) packages. [Expand](#) can be used to extrude nodes into line elements, line elements into quadrilateral elements and quadrilateral elements into hexahedral elements. Likewise, with points to curves and curves to surfaces.

Both tools are indispensable when modelling in Marc with Python, therefore MarcTools offers functions to control these tools. To reduce programming effort, these functions manage the setting of default parameters, and are highly polymorphic. For example, all of the following are valid uses of [Move](#).

Let mesh be a tuple or list of [MarcPoint](#), [MarcCurve](#), [MarcSurface](#), [MarcNode](#) and [MarcElement](#) objects.

```
Move(dx=10,dy=-2,*mesh) #Moves all objects in 'mesh' 10 along x, and -2 along y

Move(sx=2,*mesh) #Doubles (scales) the x component of all entities in 'mesh'

Move(tz=45,oy=10,*mesh) #Rotates all entities in 'mesh', 45 degrees around an axis
parallel with the model z axis, and intersecting a point with origin x=0, y=10,
z=0.
```

[Move](#) has an alias named [Rotate](#) that simply redirects all of its inputs to [Move](#).

[Expand](#) is slightly more complex as this function must manage the destruction of entities (and their corresponding MarcTools objects) to be expanded and the creation of new MarcTools objects for new entities created as a result of expansion.

The creation of new entities is detected by checking the last ID of that particular type of entity before the operation and then again after the operation. Since Marc assigns IDs sequentially, it can be assumed that every ID added in between checks corresponds to a new entity.

When using mixed lists of objects, [Expand](#) carries out the expansion per input category. Table 1 shows the types of objects created when expanding certain objects. After all categories have been expanded, the combined list of created objects is returned.

Input Category	Types of objects created after expansion
<a href="#">MarcPoint</a>	<a href="#">MarcCurve</a> , <a href="#">MarcPoint</a>
<a href="#">MarcCurve</a>	<a href="#">MarcPoint</a> , <a href="#">MarcSurface</a>
<a href="#">MarcNode</a>	<a href="#">MarcNode</a> , <a href="#">MarcElement</a>
<a href="#">MarcElement</a>	<a href="#">MarcNode</a> , <a href="#">MarcElement</a>

Table 1 – Table of entities created after expansion

MarcTools also implements a number of miscellaneous methods that help with communicating with Marc.

[A11](#)(\*marcObjects): [A11](#) is a class that behaves like a function. Any method can be called with [A11](#), which [A11](#) will try to call on all objects passed to it during initialisation. For example:

```
A11(point1, point2, line1).Make()
```

is the same as:

```
point1.Make()
point2.Make()
line1.Make()
```

`Filter(filterClass,objectList)`: `Filter` returns a tuple of objects in `objectList` that are instances of `filterClass`. For example, the following returns a tuple of all the `MarcCurve` type objects in `mesh`:

```
Filter(MarcCurve,mesh)
```

`SendIds(*idList)`: `SendIds` sends a list of values (typically entity IDs) to Marc using the `py_send` command. If the list is longer than the maximum length allowed in Marc, then `SendIds` will break up the list into smaller chunks. `SendIds` also automatically sends the “#” identifier to signify the end of a list. For example, the following sequence would hide all elements belonging to `mesh`:

```
py_send(*select_elements")
SendIds(*Filter(MarcElement,mesh))
py_send(*invisible_selected")
```



# Appendix B – Fortran Source Code

## Procedure Headers

---

This appendix shows important variable definitions, type definitions, and procedure headers for key functions and subroutines developed for the Fortran program.

```
162! Mathematics definitions
163 #define EPSILON      1E-11  ! The error below which two floating point numbers are
! regarded as equal
164 #define SWEEP        5E-4   ! The distance below which two locations are coincident in the
! mesher
165 #define SWEEP_ALPHA  2E-2   ! If two lines are within this angle, they are colinear
166 #define SWEEP_NURBS  1E-7   ! Used to determine if two NURBS are touching
167 #define COARSE       8E-3
168 #define ALPHA_COLINEAR 6.1E-4 ! 3.8E-5 is approximately 0.5 degrees
169
179! Facet flags
180 #define FF_Default      0      ! Default state
181 #define FF_Disabled     1      ! Disabled facets are ignored by all routines operating
! on facet lists
182 #define FF_Side1Free    2      ! Side 1 free, (direction of cross product of v1-v2
! v1-v3)
183 #define FF_Side2Free    4      ! Opposite of side 1 free (Side 2 must be the left shift
! of Side 1)
184 #define FF_KeepHull     8      ! Keep because this facet is part of the hull
185 #define FF_KeepInterface 16    ! Keep because this facet is part of the
! cutter-workpiece interface
186 #define FF_Occluding    32    ! Facet blocks the visibility of some point, from some
! other facet
187 #define FF_Split        64    ! Facet has split rays associated with it
188 #define FF_Sides        FF_Side1Free + FF_Side2Free
189
190! Point flags
191 #define PF_Default      0      ! Default state
192 #define PF_Disabled     1      ! Disabled points are ignored by all routines operating
! on point lists
193 #define PF_Mating       2      ! Node mates with the parent mesh
194 #define PF_Movable      4      ! Node is not significant to the shape of the mesh and
! can be moved without distorting the shape
195 #define PF_Missing      8      ! Associated with a missing facet
196 #define PF_Required     16    ! Required in mesh
197
198! Ray flags
199 #define RF_Default      0      ! Default state
200 #define RF_MeshTri      1      ! Boundary flag, set in TriTriIntersect to indicate
! termination on mesh facet
201 #define RF_NURBSTri     2      ! Boundary flag, set in TriTriIntersect to indicate
! termination on NURBS facet
202 #define RF_Disabled     4      ! Disabled rays are ignored by all routines operating on
! ray lists
203 #define RF_Side1Free    8      !  $1-P \cdot (1 \rightarrow 2 \times 1 \rightarrow N) > 0$  if P is on side one of
! ray 1-2, on a plane with normal vector N
204 #define RF_Side2Free    16    ! Same as above, but less than zero
205 #define RF_External     32    ! Mesh ray (edge of facet) is external (doesn't mate
! with parent mesh)
206 #define RF_Checked      64    ! Ray has been checked during simplification
207 #define RF_Consumed     128   ! Ray has been assigned to an island
208 #define RF_ErrTooNarrow 256   ! Ray cast from a workpiece node against the cutter does
! not intersect the cutter (in other words, the cutter
! geometry is too narrow for the job)
209 #define RF_Sides        RF_Side1Free + RF_Side2Free
210
211! Element and node flags
212 #define F_Default      0      ! Default state
213 #define F_Workpiece     1      ! Element or node is associated with the workpiece
214 #define F_NotUsed       2      ! Slot in elementIndex/nodeIndex is not occupied
215 #define F_ProximityChecked 4    ! Node has been checked for proximity to the cutter
216 #define F_PosSet       8      ! Cached position is up to date
217 #define F_Required     16    ! Workpiece node is used by at least one element
218 #define F_Remesh       32    ! Must be remeshed
219 #define F_Killed       64    ! Element deactivated on current increment
220 #define F_NonWorkpiece 128    ! Node is shared with a non-workpiece element Nodes can
! have both workpiece and non-workpiece if there are
```

```

221 #define F_CutFace           256      ! shared between two bodies
222 #define F_Expansion         512      ! Associated with part of the mesh touching a cutter
                                         ! Element should be included with sub-mesh elements
                                         ! during selection expansion
223 #define F_VolSet            1024     ! Cached volume is up to date
224 #define F_Adjusted          2048     ! Node has been snapped to another location
                                         ! (NODE%adjustment)
225 #define F_Counted           4096     ! Node has been counted during reindexing
226 #define F_New               8192     ! Element or node has been newly created in a virtual
                                         ! state in SplitMesh()
227 #define F_Perturbed         16384    ! Node is microscopically shifted to overcome EPSILON
                                         ! problem (NODE%perturbation)
228 #define F_FaceChecked       32768    ! Elements faces have been checked during hull discovery
229 #define F_Track             65536    ! Highlight node in post file
230 #define F_ConfirmSplit      131072   ! Confirm that element must be split

391 module JamesMod
398   include 'spacenu'      ! NURBS information          | nurinf, rnuinf, nnurbs
399   include 'spaceco'      ! Rigid body data          | adie
400   include 'dimen'        ! Simulation geometry state | numel, numnp, nnodmx
401   include 'concom'       ! Simulation time/increment state | inc, ncycle
402   include 'spaceset'     ! Sets                    | ndset, isetdat, setnam,
                                         ! lsetdat
403   include 'autoin'       ! Time stepping           | iforcrsta
404   include 'creeps'       ! Simulation time          | cptim
405   include 'elemdata'     ! Element states           | ieltype, nelgroups
406   include 'elmcom'       ! Element types            | igroup, jtype

412   type SPINDLE
414     character(255) :: name          ! Name of this spindle
415     real(8)         :: originPos(3)  ! Spindle origin at definition
416     real(8)         :: originDir(3)  ! Spindle direction cosines at definition
                                         ! (The spindle rotates around this vecotr)
417     real(8)         :: RPM           ! Base RPM
418     real(8)         :: feedRate      ! Feed rate in mm / rev
419     real(8)         :: rapidMultiplier ! In rapid mode, this factor is multiplied
                                         ! by the normal speed to calculate the
                                         ! rapid speed
420     integer         :: originNodeInt, dirNodeInt ! Origin and direction, internal node IDs
421     integer         :: originNodeExt, dirNodeExt ! Origin and direction, external node IDs
422     integer         :: incLastStep    ! Increment number of last update
423     integer         :: nearCount      ! Number of elements near bodies attached
                                         ! to this spindle
424     real(8)         :: proximityRadius ! The distance nodes must be away from
                                         ! bodies attached to this spindle to be
                                         ! considered as far (not near)
425     real(8)         :: contactDistance ! The feed distance until cutters on this
                                         ! spindle make contact with the workpiece

426     logical         :: physFileStarted ! Flag indicating whether or not an output
                                         ! file has been created for this spindle
427     character(255) :: physFileName    ! Name of the spindle data file
428     real(8)         :: physFileLastTime ! Computation time at last write to output
                                         ! file

430
431     ! State tracking
432     real(8) :: curRPM      ! Current RPM (affected by rapid mode state)
433     real(8) :: curAngle    ! Current spindle angle in radians
434     real(8) :: dAngle      ! Last change in angle in radians
435     real(8) :: curPos(3)   ! Current position
436     real(8) :: dPos(3)     ! Change in position since last step
437     real(8) :: curDir(3)   ! Current direction cosines
438     real(8) :: dDir(3)     ! Change in direction cosines since last step
439     real(8) :: dTime       ! Time since last step
440
441
442     logical :: rapidMode
443     real(8) :: angularVelocity
444     real(8) :: velocity(3)
445     real(8) :: dp(6)       ! 6 DOF's for the load on this spindle
446
447     contains
448
449     procedure, public, pass :: Init          => SPINDLE_Init
450     procedure, public, pass :: Reset         => SPINDLE_Reset
451     procedure, public, pass :: Step          => SPINDLE_Step
452     procedure, public, pass :: DumpPhysicsData =>
453     * SPINDLE_DumpPhysicsData
454     procedure, public, pass :: CalculateFirstcontact =>

```

```

455      * SPINDLE_CalculateFirstContact
457  end type SPINDLE

465  type RAY
473    real(8) :: origin(SIM_DIM)      ! Starting coordinate of this ray
474    real(8) :: unitDirection(SIM_DIM) ! Unit direction vector of this ray
475    real(8) :: d                     ! Length of ray
476    integer :: boundPnts(2)          ! If this Ray is bounded by some real Node in
                                        ! nodeIndex, or a coordinate in a
                                        ! coordinates list, then the index of that boundary
                                        ! node/coordinate can be stored
                                        ! in this property. Offsets 1 and 2 refer to
                                        ! boundary objects at the start and
                                        ! end of the Ray, respectively.
479    integer :: boundFlags(2)         ! 'RF' bitwise encoded status flags
480                                        ! Populated during Ray%TriIntersect intersection
481                                        ! detection if this Ray starts or ends on the edge
                                        ! of a triangle. Offsets 1 and 2 refer to boundary
                                        ! objects at the start and end of the Ray,
                                        ! respectively.
482    integer :: facetIdxs(3)          ! facetIdxs(1): Primary facet (the facet on which
                                        ! this split ray is based)
483                                        ! The split ray is always entirely inside this facet
484                                        ! facetIdxs(2): Adjacent facet that is affected by
                                        ! the start of this ray
485                                        ! facetIdxs(3): Adjacent facet that is affected by
                                        ! the end of this ray
486    logical :: intersected           ! General purpose flag to indicate intersection
487    integer :: elemInt               ! An element associated with this ray gets removed
                                        ! if an intersection occurs
488    integer :: nodeInt
489    integer :: nurbsIdx              ! General purpose storage for a NURBS index if this
                                        ! ray is associated with a NURBS surface
490    integer :: pntIdx
491    integer :: status                ! 'RF' bitwise encoded status flags
492    integer :: loopPosition(2)       ! General purpose storage for next and previous ray
                                        ! indicies if this ray is in a chain
493
494    integer :: island                ! Island index that this ray is associated with
495
496    contains
497
498    procedure, public, pass :: EndPoint => RAY_EndPoint
499    procedure, public, pass :: Vector  => RAY_Vector
500    procedure, public, pass :: Transform => RAY_Transform
501    procedure, public, pass :: Make     => RAY_Make
502    procedure, public, pass :: MakeFromIds
503      *      => RAY_MakeFromIds
504    procedure, public, pass :: MakeFromTransform
505      *      => RAY_MakeFromTransform
506    procedure, public, pass :: Print    => RAY_Print
507    procedure, public, pass :: PrintX  => RAY_PrintX
508    procedure, public, pass :: VectorD => RAY_VectorD
509    procedure, public, pass :: Angle   => RAY_Angle
510    procedure, public, pass :: Flip    => RAY_Flip
511    procedure, public, pass :: NormalD => RAY_NormalD
512    procedure, public, pass :: RayIntersect
513      *      => RAY_RayIntersect
514    procedure, public, pass :: PerpDistance
515      *      => RAY_PerpDistance
516    procedure, public, pass :: PointOnRay=> RAY_PointOnRay
517    procedure, public, pass :: Collinear=> RAY_Collinear
518    procedure, public, pass :: RotateTransform
519      *      => RAY_RotateTransform
520    procedure, public, pass :: TriIntersect
521      *      => RAY_TriIntersect
522    procedure, public, pass :: PlaneIntersect
523      *      => RAY_PlaneIntersect
525  end type RAY

531  type ELEMENT
533    integer :: status                ! 'F' bitwise encoded status flags
534    integer :: nearCutterIdx         ! cutterIdx of the cutter that this element is near
535    integer :: nodesInt(8)           ! Internal node IDs associated with this element
536                                        ! corresponding to offsets in nodeIndex.
537    integer :: firstInc              ! The first increment this element was present on
538    integer :: nodeCount             ! The number of nodes associated with this element
539    integer :: type                  ! Element type number according to MARC 2016 Volume B
                                        ! Element Library

```

```

540     real(8) :: dp(SIM_DIM)      ! Volume pressure direction vector
541     real(8) :: press            ! Volume pressure (N m-3)
542     real(8) :: vol              ! Cached volume of this element, invalidated at the end
543                               ! of each increment and set by a call to Volume()
544     real(8) :: UserQuantityS(ELEMENT_SCALAR_COUNT)
545
546     real(8) :: pos(SIM_DIM)      ! Cached position of this element, populated by a call to
547                               ! Position
548
549     integer :: intId             ! The MARC internal ID of this element
550 #ifdef DEBUGMODE
551     integer :: extId             ! The MARC external (user) ID of this element
552 #endif
553
554     contains
555     procedure, public, pass :: Position => ELEMENT_Position
556     procedure, public, pass :: Volume  => ELEMENT_Volume
557     procedure, public, pass :: Active  => ELEMENT_Active
558 end type ELEMENT
559
560
561 type NODE
562     real(8) :: dp(SIM_DOF)        ! Loads associated with this node
563     integer :: status              ! 'F_' bitwise encoded status flags
564     integer :: nearCutterIdx       ! Cutter index of cutter that this node is
565                               ! near. Nodes can never be near two or more
566                               ! cutters at the same time.
567
568     ! The user is expected to specify a near field
569     ! search radius in the configuration file
570     ! sufficient to meet this requirement.
571
572     integer, allocatable :: elementsInt(:) ! Array of element offsets in elementIndex
573                               ! associated with this node. This array is
574                               ! used to give effect to the elements-at-node
575                               ! cache.
576
577     ! References get broken if a list containing
578     ! this object gets reallocated.
579
580     real(8) :: pos(SIM_DIM)        ! Cached position of this node
581     real(8) :: perturbation(3)     ! Perturbation vector
582     real(8) :: adjustment(3)       ! Snap coordinate that this node should be
583                               ! moved to
584
585     ! <User quantity definitions>
586     real(8) :: UserQuantityS(NODE_SCALAR_COUNT)
587     real(8) :: UserQuantityV(NODE_VECTOR_COUNT, SIM_DIM)
588
589     integer :: firstInc            ! The first increment this node was present on
590     integer :: intId              ! MARC internal ID
591 #ifdef DEBUGMODE
592     integer :: extId              ! Marc external ID
593 #endif
594
595     integer :: seqId              ! Sequence ID to assist with sorting lists of nodes
596     integer :: mergeId            ! If a merge ID is set, then this node is just a place holder
597                               ! that will be substituted by another node
598                               ! MergeID uses the +/- numbering convention
599
600     contains
601
602     procedure, public, pass :: Position =>
603     * NODE_Position
604     procedure, public, pass :: Reset =>
605     * NODE_Reset
606     procedure, public, pass :: Perturb =>
607     * NODE_Perturb
608 end type NODE
609
610
611 type NURBS_SURFACE
612     integer :: uNtrlPnt,          ! Number of control points in the parametric u and v axis
613     * vNtrlPnt                  ! respectively
614     integer :: uDiv, vDiv         ! Recommended number of divisions in the parametric u and v
615                               ! axes respectively
616     integer :: uOrder, vOrder     ! NURBS entity order in the parametric u and v axes
617                               ! respectively
618     integer :: minU, maxU,        ! If set, these values limit the minimum and maximum
619     * minV, maxV                 ! parametric u and v values to use when calculating the
620                               ! effective surface of this NURBS surface entity during
621                               ! cutter-workpiece intersection detection.
622
623     integer :: marcId             ! The Marc ID of this NURBS in Marc's internal database
624     logical :: disabled           ! NURBS can be flagged as disabled if no part of them is
625                               ! involved in cutting (not front facing)
626 end type NURBS_SURFACE

```

```

640  real(8) :: crudeNormal(3)  ! A normal vector assembled from the cross product
641                                ! of any three non-collinear points on this NURBS surface
642                                ! entity.
643
644  ! The following information deals with mapping this NURBS surface in relation to
645  ! neighbour NURBS surfaces
646  logical :: mapped           ! Used to signal if this NURBS surface has been mapped
647  real(8) :: transform(3,3)   ! An affine transform for transforming this NURBS
648                                ! entity's parametric coordinates from local to
649                                ! relative parametric coordinates (relative to adjacent
650                                ! NURBS entities in the same contact body). This
651                                ! transform is used when meshing across multiple NURBS
652                                ! surfaces.
653  real(8) :: invTransform(3,3) ! The inverse transform of transform (above).
654  real(8) :: scales(2)         ! Required scaling factors required to make this NURBS
655                                ! surface fit in the map. 1 = u scale, 1 = v scale
656  logical :: uvFlip(2)        ! 1 = flipped on U, 2 = flipped on V
657  real(8) :: uOrigin, vOrigin ! Parametric origin in global map
658  real(8) :: angleOffset      ! Angle offset in relation to neighbours
659  integer :: mapI(4,5)        ! Map data describing surrounding NURBS surfaces
660                                ! (x,:), where x=1=top, x=2=right, x=3=bottom, x=4=left
661                                ! (:,1) = nurbId (relative to the cutter this NURBS is
662                                ! associated with)
663                                ! (:,2) = edgeId on neighbour
664                                ! (:,3) = orientation, 1=same (clockwise), 2=opposite
665                                ! (:,4-5) = 4&5=u&v
666  real(8) :: mapR(4,3)        ! Real coordinates of corners
667
668  real(8), allocatable :: uKnotVector(:), vKnotVector(:) ! Parametric u and v knot
669                                ! vectors respectively
670  real(8), allocatable :: homoCoords(:,:,:)              ! (u,v,XYZ) Homogenous
671                                ! control point coordinate grid
672  real(8), allocatable :: weights(:,:,:)                 ! (u,v,1) The control point
673                                ! weights grid
674  real(8), allocatable :: uGridVector(:), vGridVector(:) ! The parametric u and v
675                                ! vectors (used for seeding
676                                ! surfaceGrid. Unlike
677                                ! surfaceGrid, these don't need
678                                ! to be a grid as all columns
679                                ! have the same u value and
680                                ! likewise all rows have the
681                                ! same v value)
682  real(8), allocatable :: surfaceGrid(:,:,:)              ! A cached matrix of
683                                ! precomputed real coordinates
684                                ! on the surface of this NURBS
685                                ! surface entity. Used in a
686                                ! number of places, for
687                                ! example, to accelerate the
688                                ! calculation of discrete
689                                ! surface triangles in
690                                ! TriIntersect intersection
691                                ! detection.
692
693  contains
694
695  procedure, public, pass :: Configure      => NURBSSURFACE_Configure
696  procedure, public, pass :: Populate       => NURBSSURFACE_Populate
697  procedure, public, pass :: Centre         => NURBSSURFACE_Centre
698  procedure, public, pass :: RayIntersect   =>
699  *   NURBSSURFACE_RayIntersect
700  procedure, public, pass :: TriIntersect   =>
701  *   NURBSSURFACE_TriIntersect
702  procedure, public, pass :: DumpToMARC     =>
703  *   NURBSSURFACE_DumpToMARC
704  procedure, public, pass :: GetUV          => NURBSSURFACE_GetUV
705  procedure, public, pass :: S              => NURBSSURFACE_S
706  end type NURBSSURFACE
707
708  type CUTTER
709  integer :: spindleIdx          ! Spindle index used to drive this cutter
710  integer :: incLastUpdate      ! Increment number of last position update
711  integer :: bodyId             ! MARC internal contact body ID
712  integer :: nodeTransInt, nodeRotInt ! Marc internal IDs for this cutters
713                                ! translation and rotation nodes
714                                ! respectively
715  integer :: nodeTransExt, nodeRotExt ! Marc external IDs for this cutters
716                                ! translation and rotation nodes
717                                ! respectively
718  logical :: selfCheck          ! Set to true if this cutters control node

```

```

692     character(128) :: cutterName           ! is seen in FORCDT
693     real(8)        :: originPosition(SIM_DIM) ! The name of this Cutter (used when
694     real(8)        :: position(SIM_DIM)      ! printing summary information)
695     real(8)        :: positionDelta(SIM_DIM) ! The position of this Cutter at the start
696     logical        :: engaged               ! of the simulation.
697     type(RAY)      :: proximityRay          ! The position of this Cutter in the
698                                           ! current increment.
699     integer        :: localCysNodes(2)      ! Change in position vector of this cutter
700     real(8)        :: volumeNear           ! from the previous increment
701     real(8)        :: sliceArea            ! A flag indicating if this Cutter is in
702                                           ! contact with the workpiece or not.
703                                           ! A representative Ray that runs the width
704                                           ! of this Cutter, which is used to test
705                                           ! for nodal proximity.
706
707     integer        :: localCysNodes(2)
708     real(8)        :: volumeNear
709     real(8)        :: sliceArea            ! Area of intersection between this cutter
710                                           ! and the workpiece on this increment
711
712     integer        :: nurbCount             ! Number of NURBS surfaces
713     type(NURBSSURFACE), allocatable :: nurbsSurfaces(:) ! associated with this cutter
714                                           ! A list of NURBSSurface objects
715                                           ! that make up the geometry of
716                                           ! this Cutter,
717                                           ! including two invisible
718                                           ! NURBSSurface objects at index 1
719                                           ! and 2, that assist with mesh
720                                           ! splitting.
721
722     ! Coordinate systems
723     ! Dimension 1 : X, Y, Z, Homogenous coordinate (1)
724     ! Dimension 2 : Rake, Feed, Radial
725     real(8), dimension(4,SIM_DIM) :: originalCS ! Coordinate system at the start of
726                                           ! simulation (see table below)
727     real(8), dimension(4,SIM_DIM) :: currentCS  ! Coordinate system in the current
728                                           ! increment (see table below)
729
730     ! Depth Reference: This is the vector along the feed direction
731     ! from the control node to the lowest point of the cutter.
732     ! This defines the depth of cut
733     real(8) :: pathRadius
734     real(8) :: depthOffset
735
736     real(8) :: maxDOC, avgDOC              ! Approximate maximum and average depths
737                                           ! of cut
738     real(8) :: WOC                        ! Approximate width of cut
739
740     real(8) :: dShape(3,2)                ! The min and max d values in all three
741                                           ! local axis that create
742                                           ! rays from the cutter origin to the min
743                                           ! and max points
744
745     real(8) :: dp(6)                      ! Cutter loads in global coordinates
746     real(8) :: dpLocal(6)                 ! Cutter loads in local coordinates
747
748     ! Affine Transforms:
749     real(8) :: affineTransform(4,4)        ! Transform from position in previous
750                                           ! increment to current increment position
751     real(8) :: affineTransformInv(4,4)      ! Inverse of affineTransform describing
752                                           ! the reverse action
753     real(8) :: affineTransformAbs(4,4)      ! Absolute transform from starting
754                                           ! position to current position
755
756     real(8), allocatable :: VolHistory(:)  ! Array of volumes removed per increment.
757
758     integer :: meshErrorCount              ! Number of mesh errors in SplitMesh
759                                           ! associated with this cutter
760
761     integer :: NURBSID_Start               ! The index of the first NURBS in nurinf
762     integer :: NURBSID_Finish              ! The index of the last NURBS in nurinf
763
764     logical :: physFileStarted              ! Switch indicating whether or not a
765                                           ! physics output file has been started for
766                                           ! this cutter
767     character(255) :: physFileName          ! Name of this cutter physics file
768     real(8)        :: physFileLastTime      ! Last time data was written to this
769                                           ! cutters physics file
770     logical        :: profileFileStarted    ! Switch indicating whether or not a
771                                           ! profile output file has been started for

```



```

743         character(255) :: profileFileName           ! this cutter
744                                                     ! Name of this cutters profile file
745     contains
746     procedure, public, pass :: UpdatePosition =>
747     * CUTTER_UpdatePosition
748     procedure, public, pass :: Title => CUTTER_Title
749     procedure, public, pass :: SetToolSurfaces =>
750     * CUTTER_SetToolSurfaces
751     procedure, public, pass :: DumpPhysicsData =>
752     * CUTTER_DumpPhysicsData
754 end type CUTTER
986
987 ! Global indexes
988 type(ELEMENT),      allocatable, target :: elementIndex(:)
989 type(NODE),          allocatable, target :: nodeIndex(:)
990 type(CUTTER),        allocatable, target :: cutterIndex(:)
991 type(SPINDLE),       allocatable, target :: spindleIndex(:)
1060 contains ! In root of JamesMod

1062 subroutine SPINDLE_Init(this,name)
1064     ! Function: Initialise this spindle using default values
1065     ! Inputs:
1066     class(SPINDLE), intent(inout) :: this ! This spindle
1067     character(*),    intent(in)    :: name ! Name of this spindle
1068 end subroutine SPINDLE_Init

1144 subroutine SPINDLE_Reset(this)
1146     ! Function: Resets increment dependent values, e.g. loads and near count
1147     ! Inputs:
1148     class(SPINDLE), intent(inout) :: this ! This spindle
1149 end subroutine SPINDLE_Reset

1155 subroutine SPINDLE_Step(this,dTime)
1157     ! Function: Advances this spindle by a given time step
1158     ! Inputs / Outputs:
1159     class(SPINDLE), intent(inout) :: this ! This spindle
1160     real(8),         intent(in)    :: dTime ! Time step to advance this spindle
1161 end subroutine SPINDLE_Step

1204 subroutine SPINDLE_DumpPhysicsData(this)
1206     ! Function: Writes key data about this spindle out to a data file
1207     ! Inputs:
1208     class(SPINDLE), intent(inout) :: this ! This spindle
1209 end subroutine SPINDLE_DumpPhysicsData

1298 subroutine SPINDLE_CalculateFirstContact(this)
1300     ! Function: Estimates the spindle displacement excepted at the point
1301     ! of first contact between any cutter and the workpiece and
1302     ! stores the result on this spindle for reference.
1303     ! Input:
1304     class(SPINDLE), intent(inout) :: this ! This spindle
1305 end subroutine SPINDLE_CalculateFirstContact

1427 function CUTTER_Title(this)
1429     ! Function: Returns an exact length string containing a description of this cutter
1430     ! Input:
1431     class(CUTTER), intent(in) :: this ! This cutter
1432     ! Returns:
1433     character(len=:), allocatable :: CUTTER_Title ! Cutter description
1434 end function CUTTER_Title

1454 subroutine CUTTER_SetToolSurfaces(this,basePlaneEdge)
1456     ! Function: -Culls non-front facing portions of this cutters NURBS surfaces
1457     ! -Populates map information in associated NURBS to describe their
1458     ! positions relative to one another
1459     ! Inputs:
1460     class(CUTTER), target, intent(in) :: this ! This cutter
1461     real(8),          intent(inout) :: basePlaneEdge(SIM_DIM,2) ! The lowest front
1462                                                                ! -facing edge
1463                                                                ! of this cutter
1464 end subroutine CUTTER_SetToolSurfaces

1838 subroutine CUTTER_UpdatePosition(this)
1840     ! Function: -Updates the cached position of this cutter
1841     ! -Generates additional tool surfaces to prevent saw-tooth cuts
1842     ! -Updates the position of the proximity ray
1843     ! Inputs / Outputs:
1844     class(CUTTER), target, intent(inout) :: this ! This cutter

```

```

2422     end subroutine CUTTER_UpdatePosition

2424     subroutine CUTTER_DumpPhysicsData(this)
2426         ! Function: Writes physics information about this cutter to a file
2427         ! Inputs:
2428         class(CUTTER), intent(inout) :: this ! This cutter
2515     end subroutine CUTTER_DumpPhysicsData

2552     function ELEMENT_Position(this)
2554         ! Function: -Approximates the centre of an element by taking an average of all it's
                ! nodes
2555         ! -Caches the calculated position for future calls
2556         ! Input / Output:
2557         class(ELEMENT), intent(inout) :: this ! This element
2558         ! Returns:
2559         real(8) :: ELEMENT_Position(SIM_DIM) ! The position
2560                                                ! coordinate of
2561                                                ! this element
2580     end function ELEMENT_Position

2582     function ELEMENT_Volume(this)
2584         ! Function: -Returns the volume of this element (tetrahedral elements only)
2585         ! -Caches the calculated volume for future calls
2586         ! Input / Output:
2587         class(ELEMENT), intent(inout) :: this ! This element
2588         ! Returns:
2589         real(8) :: ELEMENT_Volume ! The volume of this element
2654     end function ELEMENT_Volume

2657     function ELEMENT_Active(this)
2659         ! Function: Checks to see if this element is active or not
2660         ! Inputs:
2661         class(ELEMENT), intent(in) :: this ! This element
2662         ! Returns:
2663         logical :: ELEMENT_Active ! True if active, else False
2673     end function ELEMENT_Active

2678     recursive function NODE_Position(this)
2680         ! Function: -Caches the position of this node from the MARC database if no
                ! cached position currently exists
2681         ! -Returns the cached position of this node.
2682         ! Note: Unless the position is cached, this function cannot return
2683         ! the correct coordinates during certain phases of simulation
2684         ! Inputs / Outputs:
2685         class(NODE), intent(inout) :: this ! This Node
2686         ! Returns:
2687         real(8) :: NODE_Position(SIM_DIM) ! Coordinate of this node
2731     end function NODE_Position

2733     subroutine NODE_Perturb(this,optDeltaMax)
2735         ! Function: Creates some perturbation in this node
2736         ! position to help overcome EPSILON problems.
2737         ! The perturbation created here is undone by
2738         ! clearing the F_Perturbed flag
2739         ! Inputs / Outputs:
2740         class(NODE), intent(inout) :: this ! This Node
2741         real(8), optional, intent(in) :: optDeltaMax ! Optional maximum perturbation limit
2764     end subroutine NODE_Perturb

2766     subroutine NODE_Reset(this,optDefaultState)
2768         ! Function: -Resets the cached user quantities, IDs and position of this node
2769         ! -Optionally sets a default state
2770         ! Input / Outputs:
2771         class(NODE), intent(inout) :: this ! This node
2772         integer, intent(in), optional :: optDefaultState ! Optional default 'F_'
2773                                                         ! status flag to set
2786     end subroutine

2788     subroutine NURBSSURFACE_Configure(this, nurbId)
2790         ! Function: Queries the MARC database and caches information about
2791         ! this NURBS surface, such as control points, homogenous
2792         ! coordinates, knot vectors and addresses
2793         ! Inputs / Outputs:
2794         class(NURBSSURFACE), intent(inout) :: this ! This NURBS surface
2795         integer, intent(in) :: nurbId ! Marc NURBS ID
2980     end subroutine NURBSSURFACE_Configure

2985     subroutine NURBSSURFACE_Populate(this)
2987         ! Function: Queries Marc database for NURBS surface information to cache

```



```

2988      ! on this NURBS surface object. This subroutine also generates
2989      ! a cache of real grid coordinates on this NURBS surface
2990      ! Inputs / Outputs:
2991      class(NURBSSURFACE), intent(inout) :: this ! Thus NURBS surface
3021  end subroutine NURBSSURFACE_Populate

3023  function NURBSSURFACE_Centre(this)
3025      ! Function: Returns an average of this NURBS surface's corner coordinates
3026      ! Inputs / Outputs:
3027      class(NURBSSURFACE), intent(inout) :: this ! This NURBS surface
3028      ! Returns:
3029      real(8) :: NURBSSURFACE_Centre(SIM_DIM) ! Centre coordinate
3038  end function NURBSSURFACE_Centre
3475  end function NURBSSURFACE_RayIntersect

3477  function NURBSSURFACE_TriIntersect(this, testTri,
3478      *                               intersectionRayCount,
3479      *                               intersectionRays)
3481      ! Function: Determines the intersection rays that described the intersection
3482      ! between a given triangle and this NURBS surface.
3483      ! Inputs / Outputs:
3484      class(NURBSSURFACE), intent(in) :: this ! This NURBS surface
3485      real(8), intent(in) :: testTri(SIM_DIM,3) ! Triangle to test
3486      integer, intent(inout) :: intersectionRayCount ! Number of intersection
3487      ! rays generated
3488      type(RAY), target,
3489      *      allocatable, intent(inout) :: intersectionRays(:) ! List of intersection
3490      ! Returns:
3491      logical :: NURBSSURFACE_TriIntersect ! True if at least
3492      ! one intersection ray
3493      ! generated, else
3494      ! False

3557  end function NURBSSURFACE_TriIntersect

3559  subroutine NURBSSURFACE_DumpToMARC(this,optResetT)
3561      ! Function: Prints Marc commands to log suitable for copying and pasting
3562      ! into Marc in order to generate this NURBS surface in Marc
3563      ! Inputs:
3564      class(NURBSSURFACE), intent(in) :: this ! This NURBS surface
3565      logical, optional, intent(in) :: optResetT ! Optionally reset the persistent
3566      ! starting Marc entity ID
3617  end subroutine NURBSSURFACE_DumpToMARC

3621  function NURBSSURFACE_GetUV(this,coordinate) result(error)
3623      ! Function: Convergence based technique to recover
3624      ! the parametric coordinates of a given
3625      ! point on a NURBS surface, assuming that
3626      ! point lies on the NURBS surface.
3627      ! Inputs / Outputs:
3628      class(NURBSSURFACE), intent(in) :: this ! This NURBS surface
3629      real(8), intent(inout) :: coordinate(3) ! In as real coordinate to test,
3630      ! out as
3631      ! corresponding parametric
3632      ! coordinate

3631      ! Returns:
3632      real(8) :: error ! Distance between initial real
3633      ! coordinate and real coordinate
3634      ! generated after finding
3635      ! parametric coordinates

3781  end function NURBSSURFACE_GetUV

3786  subroutine NURBSSURFACE_S(this,u,v,output)
3788      ! Function: Calculates the real coordinate on a NURBS
3789      ! surface from parametric coordinates, u and v
3790      ! Inputs / Outputs::
3791      class(NURBSSURFACE), intent(in) :: this ! This NURBS surface
3792      real(8), intent(in) :: u, v ! NURBS parametric coordinates
3793      real(8), intent(out) :: output(SIM_DIM) ! Real coordinate
3841  end subroutine NURBSSURFACE_S

3847  subroutine RAY_Transform(this,affineTransform)
3849      ! Function: Transforms a ray according to a given affine transform
3850      ! Inputs:
3851      class(RAY), intent(inout) :: this ! This ray to transform
3852      real(8), intent(in) :: affineTransform(4,4) ! Affine transform
3869  end subroutine RAY_Transform

3871  function RAY_Vector(this)

```

```

3873      ! Function: Returns this ray as a simple vector
3874      class(RAY), intent(in) :: this          ! This ray
3875      ! Returns:
3876      real(8)                :: RAY_Vector(SIM_DIM) ! This ray as a vector
3877      RAY_Vector = this%d * this%unitDirection
3878  end function RAY_Vector

3880  function RAY_EndPoint(this,optD)
3882      ! Function: Returns the end point coordinate of this ray, or optionally some other
3883      ! coordinate, optD along this ray from the origin
3884      ! Inputs:
3885      class(RAY),          intent(in) :: this ! This Ray
3886      real(8), optional, intent(in) :: optD ! Optional d value to use, else ray's d value is
                                           ! used
3887      ! Returns:
3888      real(8)                :: RAY_EndPoint(SIM_DIM) ! End point coordinate
3889  end function RAY_EndPoint

3900  subroutine RAY_MakeFromIds(this,optPointsReal)
3902      ! Function: Makes a ray from internal node numbers stored in this%boundPnts
3903      ! using the +/- id system, where +ve indicated the position is found in nodeindex
3904      ! and -ve indicating that the position is found in optPointsReal
3905      ! Inputs:
3906      class(RAY),          intent(inout) :: this          ! This ray
3907      real(8), optional, intent(in)    :: optPointsReal(:, :) ! Reference points list for -ve
                                           ! IDs
3930  end subroutine RAY_MakeFromIds

3932  subroutine RAY_Make(this,startPoint,endPoint,optReset)
3934      ! Function: Configures a given ray it such that it has a given
3935      ! given start and end point
3936      ! Inputs / Outputs:
3937      class(RAY),          intent(inout) :: this          ! This ray
3938      real(8),             intent(in)    :: startPoint(SIM_DIM) ! Origin of the ray
3939      real(8),             intent(in)    :: endPoint(SIM_DIM)  ! End point of the ray
3940      logical, optional, intent(in)    :: optReset          ! Optional switch to reset
                                           ! ray properties to defaults
3941  end subroutine RAY_Make

3964  subroutine RAY_MakeFromTransform(this,startPoint,
3965      *                               affineTransform)
3967      ! Function: Creates a ray that describes the displacement of startPoint from it's
3968      ! origin through a given affine transform, affineTransform
3969      ! Inputs / Outputs:
3970      class(RAY), intent(inout) :: this          ! This ray
3971      real(8),    intent(inout) :: startPoint(SIM_DIM) ! Start point
3972      real(8),    intent(inout) :: affineTransform(4,4) ! Affine transform through which
                                           ! to transform affine transform
3973  end subroutine RAY_MakeFromTransform

3986  subroutine RAY_Print(this,optComment)
3988      ! Function: Prints a ray start and end coordinates to the log
3989      ! Inputs:
3990      class(RAY),          intent(in) :: this          ! This ray
3991      character(*), optional, intent(in) :: optComment ! Optional comment to display
                                           ! next to the info
4007  end subroutine RAY_Print

4009  subroutine RAY_PrintX(this,Id,optComment)
4011      ! Function: Prints a ray start and end coordinates to the log, as well as extra
4012      ! information about the ray including flags, end point IDs, loop position etc.
4013      ! Inputs:
4014      class(RAY),          intent(in) :: this          ! This ray
4015      integer,             intent(in) :: Id            ! Calling function's ID of this ray
4016      character(*), optional, intent(in) :: optComment ! Optional comment to display
                                           ! next to the info
4017  end subroutine RAY_PrintX

4051  function RAY_Angle(this,otherRay)
4053      ! Function: Returns the angle between this ray and another ray
4054      ! The angle returned is always the smallest angle
4055      ! The angle is always zero or positive
4056      ! Inputs:
4057      class(RAY), intent(in) :: this, otherRay ! Rays to test
4058      ! Returns:
4059      real(8)                :: RAY_Angle          ! Angle in radians between this ray and
                                           ! otherRay
4076  end function RAY_Angle

```

```

4078  subroutine RAY_Flip(this)
4080      ! Function: Flips the origin and end point of this ray, including
4081      ! all end-sensitive flags.
4082      ! Inputs:
4083      class(RAY), intent(inout) :: this ! This ray
4093  end subroutine RAY_Flip

4095  function RAY_VectorD(this,newDirection)
4097      ! Function: Returns the end point of this ray as d value along a another
4098      ! given direction vector originating at the origin of this ray
4099      class(RAY), intent(in) :: this ! This ray
4100      real(8), intent(in) :: newDirection(3) ! New director vector
4101      ! Returns:
4102      real(8) :: RAY_VectorD ! d value of this rays end point along
4111                               ! newDirection
4111  end function RAY_VectorD

4114  function RAY_NormalD(this,point)
4116      ! Function: Returns the distance along this ray, to the closest point on this ray to
4117      ! point
4118      ! Inputs:
4118      class(RAY), intent(in) :: this ! This ray
4119      real(8), intent(in) :: point(3) ! Coordinate to test
4120      ! Returns:
4121      real(8) :: RAY_NormalD ! Distance along this ray to closest point to
4129                               ! point
4129  end function RAY_NormalD

4131  function RAY_RayIntersect(this,other,d1,d2,error,optInfiniteRay)
4133      ! Function: Tests two rays for intersection and returns the respective distances
4134      ! from each rays origin, along the ray to the intersection point
4135      ! Inputs / Outputs:
4136      class(RAY), intent(in) :: this ! This ray
4137      class(RAY), intent(in) :: other ! Other ray to test against this
4138      real(8), intent(out) :: d1, d2 ! D values to intersection along
4139      real(8), intent(out) :: error ! this ray and other respectively
4140      logical, optional, intent(in) :: optInfiniteRay ! Distance between ray and other
4141      ! at thier closest point
4142      ! Returns:
4143      logical :: RAY_RayIntersect ! Optional flag to treat the rays
4144      ! as infinite
4145      ! Returns True if the rays
4146      ! intersect, else False
4238  end function RAY_RayIntersect

4240  function RAY_PointOnRay(this,point)
4242      ! Function: Test to see if a point is on a ray
4243      class(RAY), intent(in) :: this ! Ray to test
4244      real(8), intent(in) :: point(3) ! Point to test
4245      ! Returns:
4246      logical :: RAY_PointOnRay ! True if the point is on the ray, else False
4258  end function RAY_PointOnRay

4260  function RAY_PerpDistance(this,point)
4261      ! Function: Returns the length of the shortest possible line that joins a given
4262      ! point to any other point on this infinite ray
4263      implicit none
4264      ! Inputs:
4265      class(RAY), intent(in) :: this ! This ray
4266      real(8), intent(in) :: point(3) ! Point to test
4267      ! Returns:
4268      real(8) :: RAY_PerpDistance ! Shortest distance between point and ray
4297  end function RAY_PerpDistance

4300  function RAY_Collinear(this,other,error)
4302      ! Function: Determines if this ray is collinear with another ray, other
4303      ! by testing the perpendicular distance of the end point of the
4304      ! shortest ray to the axis of the longest ray. The distance
4305      ! is tested against a given error, error
4306      ! Inputs / Outputs:
4307      class(RAY), intent(in) :: this, other ! This and other ray to test
4308      real(8), intent(inout) :: error ! In as error limit for collinear,
4309      ! out as actual error
4310      ! Returns:
4311      logical :: RAY_Collinear ! True if collinear, else false
4333  end function RAY_Collinear

4335  subroutine RAY_RotateTransform(this,alpha,coordinates)

```

```

4337      ! Function: Uses this ray as a rotation axis to rotate a coordinates
4338      ! list, coordinates, about a given angle, alpha
4339      ! Inputs:
4340      class(RAY), intent(in) :: this           ! This Ray
4341      real(8),    intent(inout) :: coordinates(:, :) ! Coordinates to rotate around this ray
4342      real(8),    intent(in) :: alpha          ! Angle in radians to rotate coordinates
4343                                           ! about this ray
4378  end subroutine RAY_RotateTransform

4383  function RAY_TriIntersect(this, triangle, intersection,
4384  *             infiniteRay, refineRecommended, d)
4386      ! Function: Finds the intersection coordinate between a given ray and triangle
4391      ! Inputs / Outputs:
4392      class(RAY),    intent(in) :: this           ! This Ray
4393      real(8),       intent(in) :: triangle(SIM_DIM, 3) ! Triangle defined by three
4394                                           ! vertices
4395      real(8),       intent(out) :: intersection(SIM_DIM, 1) ! Intersection coordinate
4396      logical, value, intent(in) :: infiniteRay      ! Switch to enable infinite ray
4397                                           ! mode
4398      logical,       intent(out) :: refineRecommended ! If the ray narrowly stops
4399                                           ! before hitting the triangle,
4400                                           ! then recommend refine
4401      real(8),       intent(out) :: d              ! Distance along ray to
4402                                           ! intersection point
4403
4404      ! Returns:
4405      logical :: RAY_TriIntersect ! True if intersect, else false
4476  end function RAY_TriIntersect

4478  function RAY_PlaneIntersect(this, planeOrigin, planeNormal,
4479  *             intersection, infiniteRay, d, alpha)
4481      ! Function: Calculates the intersection coordinate between a ray and plane
4482      ! Inputs / Outputs:
4483      class(RAY), intent(in) :: this           ! This Ray
4484      real(8),    intent(in) :: planeOrigin(SIM_DIM) ! The origin coordinate of the plane
4485                                           ! to test
4486      real(8),    intent(in) :: planeNormal(SIM_DIM) ! The normal vector of the plane to
4487                                           ! test
4488      real(8),    intent(out) :: intersection(SIM_DIM) ! Output for intersection coordinate
4489      logical,    intent(in) :: infiniteRay          ! Flag for infinite ray mode
4490      real(8),    intent(out) :: d                   ! Output for distance along ray until
4491                                           ! intersection
4492      real(8),    intent(out) :: alpha                ! Angle between ray and plane normal
4493                                           ! vector
4494
4495      ! Returns:
4496      logical :: RAY_PlaneIntersect ! True if intersection occurred,
4497                                           ! else False
4519  end function RAY_PlaneIntersect

4521  recursive subroutine NURBSMapWalker(pCutter, nurbsAIdx)
4522      ! Function: The aim of NURBS mapping is to allow coordinates specified on a global
4523      ! plane to be mapped to their appropriate NURBS surface and vice versa.
4524      ! This function walks NURBS surface belonging to pCutter, calculating, scale, rotation
4525      ! and translation factors and the accompanying affine and inverse affine transforms.
4526      ! Inputs / Outputs:
4527      type(CUTTER), target, intent(inout) :: pCutter ! Pointer to cutter
4528      integer,      intent(in) :: nurbsAIdx ! Next NURBS surface to position in
4529                                           ! map
4563  end subroutine

4566  subroutine NURBSMapWalker_TransZero(pNurbs, dUB, dVB)
4567      ! Function: Applies a translation affine transform to this NURBS surface's
4568      ! internal transform, according to a given change in parametric
4569      ! coordinates
4570      ! Inputs / Outputs:
4571      type(NURBS_SURFACE), pointer, intent(inout) :: pNurbs ! Pointer to NURBS surface
4572      real(8),              intent(inout) :: dUB, dVB ! Translation coordinates
4589  end subroutine NURBSMapWalker_TransZero

4591  subroutine NURBSMapWalker_Scale(pNurbs)
4592      ! Function: Applies a scale affine transform to this NURBS surface's
4593      ! internal transform, according to NURBS surface scale factors
4594      ! stored in %scale(1:2)
4595      ! Inputs / Outputs:
4596      type(NURBS_SURFACE), intent(inout), pointer :: pNurbs ! Pointer to NURBS surface
4706  end subroutine NURBSMapWalker_Scale

4590  subroutine DumpElements(listElements)
4591      ! Function: Print a list of all elements in listElements.
4592      ! Information contains associated nodes, flags,

```

```

5094      ! type and contact body id
5096      ! Inputs:
5097      type(ELEMENT), intent(inout), allocatable,
5098      *          target :: listElements(:) ! List of elements to print
5149  end subroutine

5151  subroutine DumpNodes(listNodes)
5153      ! Function: Print a list of all nodes in listNodes.
5154      ! Information contains flags,
5155      ! special designations and workpiece status
5157      ! Inputs:
5158      type(NODE), intent(inout), allocatable,
5159      *          target :: listNodes(:) ! List of nodes to print
5186  end subroutine

5188  subroutine DumpRays(listRays,optResetT)
5190      ! Function: Writes a list of Marc commands to the log to generate
5191      ! all rays in listRays as nodes and 1d elements
5192      ! Inputs / Outputs
5193      type(RAY), intent(inout), allocatable,
5194      *          target :: listRays(:) ! List of rays to print
5195      logical, optional, intent(in) :: optResetT ! Optionally reset the persistent
                                                ! starting Marc entity ID
5229  end subroutine DumpRays

5231  subroutine DumpTri3D(pt1, pt2, pt3, optResetT)
5233      ! Function: Dumps triangle vertices to log, for copy and paste into Marc user
5234      ! interface
5235      ! Inputs:
5236      real(8), dimension(3), intent(in) :: pt1, pt2, pt3 ! Triangle vertex coordinates
5237      logical, value, optional, intent(in) :: optResetT ! Optionally reset an output ID
                                                ! offset
5254  end subroutine DumpTri3D

5256  subroutine DumpMesh(pointsReal, nPoints, facetsInt,
5257  *          nFacets, optSideFlags)
5259      ! Function: Generate Marc commands necessary to recreate a point and triangle
5260      ! list in MSC Marc
5261      ! Inputs / Outputs:
5262      real(8), allocatable, intent(in) :: pointsReal(:, :) ! Points to dump
5263      integer, intent(in) :: nPoints ! Number of points in
                                                ! pointsReal
5264      integer, allocatable, intent(in) :: facetsInt(:, :) ! Triangles to dump
5265      integer, intent(in) :: nFacets ! Number of triangles in tris
5266      logical, optional, value, intent(in) :: optSideFlags ! Optional side flag mask
                                                ! filter
5299  end subroutine DumpMesh

5301  subroutine Transform3D_SINGLE(coordinate,affineTransform)
5303      ! Function: Transforms a single R3 coordinate by a given affine transform
5304      ! Inputs / Outputs:
5305      real(8), intent(inout) :: coordinate(SIM_DIM) ! R3 coordinate to transform
5306      real(8), intent(in) :: affineTransform(4,4) ! Affine transform
5313  end subroutine Transform3D_SINGLE

5315  subroutine Transform3D_MULTI(coordinates,affineTransform)
5317      ! Function: Transforms a list of R3 coordinates by a given affine transform
5318      ! Inputs / Outputs:
5319      real(8), intent(inout) :: coordinates(:, :) ! (n,3) R3 coordinates to transform
5320      real(8), intent(in) :: affineTransform(4,4) ! Affine transform
5329  end subroutine Transform3D_MULTI

5332  subroutine Transform2D_SINGLE(coordinate,affineTransform)
5334      ! Function: Transforms a single R2 coordinate by a given affine transform
5335      ! Inputs / Outputs:
5336      real(8), intent(inout) :: coordinate(2) ! Coordinate to transform
5337      real(8), intent(in) :: affineTransform(3,3) ! Affine transform
5344  end subroutine Transform2D_SINGLE

5346  subroutine Transform2D_MULTI(coordinates,affineTransform)
5348      ! Function: Transforms a R2 coordinate list by a given affine transform
5349      ! Inputs / Outputs:
5350      real(8), intent(inout) :: coordinates(:, :) ! (n,2) coordinates to transform in
                                                ! place
5351      real(8), intent(in) :: affineTransform(3,3) ! Affine transform
5360  end subroutine Transform2D_MULTI

5402  subroutine Raise(lineNumber,strE)
5404      ! Function: Raise allows exceptions that occur during execution to be handled

```

```

5405      ! in a safe way that outputs a message to the console, making sure it is
5406      ! written (and not stuck in a buffer) and then terminating execution
5407      ! Inputs:
5408      integer,          intent(in) :: lineNumber ! Line number error was raised on
5409      character(*), optional, intent(in) :: strE      ! Warning message
5444  end subroutine Raise

5446  subroutine Warn(lineNumber,strE)
5449      ! Functions: -Presents a warning message but doesn't halt execution
5450      ! -Writes warning message to log file
5451      ! -Sets warning indicator so all future prints indicate a
5452      ! historical warning
5454      ! Inputs:
5455      integer,          intent(in) :: lineNumber ! Line number warning was raised
5456      character(*), optional, intent(in) :: strE      ! Warning message
5484  end subroutine Warn

5487  subroutine LogException(strError)
5490      ! Function: Logs an error message to a dedicated error log file
5492      ! Input:
5493      character(2048), intent(in) :: strError ! The error string to log
5517  end subroutine LogException

5520  function SingleShot(lineNumber)
5522      ! Function: Used to detect whether or not a previous call to SingleShot
5523      ! was made from a given line number
5525      ! Input:
5526      integer, intent(in) :: lineNumber ! The calling line
5528      ! Returns:
5529      logical              :: SingleShot ! True if previous call was from another line, else
5535      ! false
5535  end function SingleShot

5537  subroutine LogStat(msg,tier)
5539      ! Function: Records a timestamped event.
5540      ! The information is used to later to work out the time between events
5541      ! and therefore the time each section of code took to execute.
5542      ! The information can be dumped as a hierarchical table of operation durations
5543      ! Inputs:
5544      character(*),          intent(in) :: msg      ! Message to log
5545      integer, optional, intent(in) :: tier      ! Hierarchy level
5578  end subroutine LogStat

5581  subroutine FlushStat()
5582      ! Function: Prints out the statistics tracking timestamps and durations
5694  end subroutine FlushStat

5700  subroutine TimerStart()
5702      ! Function: Starts the timer
5703      call SystemClock(timeStart)
5704      timeSet = True
5705  end subroutine TimerStart

5707  function TimerNow()
5709      ! Function: Returns the current value of timer
5711      ! Returns:
5712      real(8) :: TimerNow ! Current timer value in seconds
5720  end function TimerNow

5722  subroutine TimerPrint()
5723      ! Function: Prints the current timer value in seconds to the console
5726  end subroutine TimerPrint

5741  subroutine SystemClock(seconds)
5743      ! Function: Sets seconds since this function was first called
5746      ! Outputs:
5747      real(8), intent(out) :: seconds ! System time in seconds since some unspecified time
5759      ! in the past
5759  end subroutine SystemClock

6260  subroutine GetModelPathAndName(path,modelName)
6262      ! Function: Gets the path to this model and returns the model name
6263      ! Outputs:
6264      character(*), intent(out) :: path      ! The path to the folder containing the model
6265      character(*), intent(out) :: modelName ! The base name of the model
6287  end subroutine GetModelPathAndName

6289  subroutine PrintToLog(line, msg, optChannel)

```



```

6291      ! Function: Writes the characters in msg to unit file 6 (or optionally optChannel)
6292      ! The message is prefixed by an information string that includes the line
6293      ! number, line
6294      ! Inputs:
6295      integer,          intent(in) :: line          ! Originating line
6296      character(*),    intent(in) :: msg           ! Message to write to log
6297      integer, optional, intent(in) :: optChannel ! Optional channel ID (used by tail
                                                ! script)
6317  end subroutine PrintToLog

6319  function Bool(str)
6321      ! Function: Interprets a user defined string and returns a best guess
6322      ! for its boolean meaning
6323      ! Inputs:
6324      character(*), intent(in) :: str    ! String to test
6325      ! Returns:
6326      logical                :: Bool ! True of string looks like True
6327                                ! (e.g. 'yes', 'enabled'), else false
6345  end function

6350  function itoa(intVal,optFixedLength)
6352      ! Function: Utility function to convert an integer value to an exact length string
6353      ! Inputs:
6354      integer,          intent(in) :: intVal          ! Value to format as integer
6355      integer, optional, intent(in) :: optFixedLength ! Optional length to pad
6356      ! Returns:
6357      character(len=:), allocatable :: itoa          ! Exact length string containing
                                                ! formatted value
6377  end function itoa

6379  function etoa(floatVal,optDecimalPlaces)
6381      ! Function: Utility function to convert an scientific value to an exact length string
6382      ! Inputs:
6383      real(8),          intent(in) :: floatVal          ! Value to format as scientific
                                                ! number
6384      integer, optional, intent(in) :: optDecimalPlaces ! Optional number of decimal places
                                                ! to use
6385      ! Returns:
6386      character(len=:), allocatable :: etoa          ! Exact length string containing
                                                ! formatted value
6407  end function etoa

6409  function ftoa(floatVal,optDecimalPlaces)
6411      ! Function: Utility function to convert an float value to an exact length string
6412      ! Inputs:
6413      real(8),          intent(in) :: floatVal          ! Value to format as floating point
6414      integer, optional, intent(in) :: optDecimalPlaces ! Optional number of decimal places
                                                ! to use
6415      ! Returns:
6416      character(len=:), allocatable :: ftoa          ! Exact length string containing
                                                ! formatted value
6437  end function ftoa

6495  subroutine Sort(realArray,nRealArray,optReverse)
6497      ! Function: Sorts an array from high to low
6498      ! Inputs / Outputs:
6499      real(8), allocatable, intent(inout) :: realArray(:) ! Array of data to sort in
                                                ! place
6500      integer,          intent(in)  :: nRealArray          ! Number of items in array
6501      logical, optional, value, intent(in) :: optReverse    ! Optional switch to reverse
                                                ! the sort order
6525  end subroutine Sort

6529  function Circumcircle(tri, centre, radius)
6531      ! Function: Returns the circumcircle origin and radius of a triangle in R3
6533      ! Inputs / Outputs:
6536      real(8), intent(in)  :: tri(3,3) ! (3 points, XYZ) Triangle vertex coordinates
6537      real(8), intent( out) :: centre(3) ! Centre coordinate of circumcircle
6538      real(8), intent( out) :: radius    ! Radius of circumcircle
6540      ! Returns:
6541      logical                :: Circumcircle ! True if valid circumcircle, else False
6542                                ! (points are colinear)
6561  end function Circumcircle

6563  function Circumsphere(tet, centre, radius)
6565      ! Function: Returns the circumsphere of a tetrahedral
6571      ! Inputs / Outputs:
6572      real(8), intent(in)  :: tet(4,3) ! (4 points, XYZ)
6573      real(8), intent( out) :: centre(3) ! Coordinate of the centre of the circumcircle

```

```

6574     real(8), intent( out) :: radius      ! Radius of the circumcircle
6575     ! Returns:
6576     logical :: Circumsphere ! This function can fail if points are colinear
6646 end function Circumsphere
6652 end function

6657 recursive function CDBR(i,degree,knotVector,x,Cache) result(Nout)
6659     ! Function: Cox-de Boor Recursion Formula (accelerated by buffers)
6661     ! Inputs:
6662     integer, value,      intent(in)  :: i, degree      ! See CDBR equation
6663     real(8), allocatable, intent(in)  :: knotVector(:) ! See CDBR equation
6664     real(8),             intent(in)  :: x              ! See CDBR equation
6665     real(8), allocatable, intent(inout) :: Cache(:, :) ! (i, degree) Cache to store the
                                                ! results of prior calls with
                                                ! identical parameters

6667     ! Returns:
6668     real(8) :: Nout ! Product of CDBR equation
6705 end function CDBR

6711 subroutine Setup()
6713     ! Function: -Loads in settings from configuration file
6714     ! -Initialises elementIndex and nodeIndex
6715     ! -Discovers and sets up cutters
6716     ! -Discovers and sets up spindle
7446 end subroutine Setup

7448 subroutine GetSetNodes(setName,setNodes,setLength)
7450     ! Function: Queries the Marc database and gets the nodes associated
7451     ! with a given set, specified by name
7453     ! Inputs / Outputs:
7454     character(*), intent(in)  :: setName      ! The name of the set to query
7455     integer,       intent(inout) :: setNodes(*) ! Node IDs in the set
7456     integer,       intent(inout) :: setLength  ! Number of nodes in the set
7497 end subroutine GetSetNodes

7499 function ElementType(elemInt)
7502     ! Function: Returns the element type of element with internal ID, elemInt
7504     ! Inputs:
7505     integer, intent(in) :: elemInt ! Internal element ID
7507     ! Returns:
7508     integer :: ElementType ! Element type ID (See Marc 2016 Volume B)
7521 end function ElementType

7523 subroutine IndexMesh(optReset)
7525     ! Function: Creates and populates the nodeIndex and elementIndex arrays
7527     ! Inputs:
7528     logical, optional, intent(in) :: optReset ! Optionally reset node flags is present and
                                                ! True

7702 end subroutine IndexMesh

7704 subroutine PrintIndexSummary()
7706     ! Function: Prints brief summary information about elementIndex
7707     ! and nodeIndex
7708     print("      Number of elements: " // itoa(numel))
7709     print("      of which indexed: " //
7710           iff(numel == ubound(elementIndex,1),
7711             "(all)",
7712             itoa(ubound(elementIndex,1))))
7713     print("      Number of nodes: " // itoa(numnp))
7714     print("      of which indexed: " //
7715           iff(numnp == ubound(nodeIndex,1),
7716             "(all)",
7717             itoa(ubound(nodeIndex,1))))
7718 end subroutine PrintIndexSummary

7759 subroutine CompileEquation(instructionBuffer,equationIn)
7761     ! Function: Generates an equation instruction sequence compatible with 'Evaluate()'
7762     ! from a string equation
7764     ! Inputs / Outputs:
7765     real(8),      intent( out) :: instructionBuffer(:, :) ! Equations are limited to
7766                                                         ! 255 instructions
7767     character(*), intent(in)  :: equationIn              ! Input string equation
8102 end subroutine CompileEquation

8104 function Evaluate(instructions,varTable)
8106     ! Function: Evaluates a given precompiled equation with reference to
8107     ! variables stored in varTable
8131     ! Inputs:
8132     real(8),      intent(in) :: instructions(:, :) ! Instruction list to evaluate

```



```

8133     real(8), optional, intent(in) :: varTable(:)           ! Variables table for reference by
                                                                ! instructions
8135     ! Returns:
8136     real(8)                               :: Evaluate       ! Value of evaluated equation
8286 end function Evaluate

8496 function Cross(a,b)
8498     ! Function: Calculates and returns the cross product of vectors a and b
8500     ! Inputs:
8501     real(8), intent(in)   ) :: a(3), b(3) ! Vectors to operate on
8502     ! Returns:
8503     real(8)               :: Cross(3)    ! Cross product vector
8509 end function Cross

8511 function TriTriIntersect(tri1,tri2,pRay)
8513     ! Function: Tests two triangles for intersection and generates a ray at their line
8514     ! of intersection
8515     ! Inputs / Outputs:
8516     real(8),               intent(in)   ) :: tri1(3,XYZ)    ! Triangle 1
8517     real(8),               intent(in)   ) :: tri2(3,XYZ)    ! Triangle 2
8518     type(RAY), target, intent(inout) :: pRay                ! Intersection ray output
8519     ! Returns:
8520     logical                :: TriTriIntersect ! True if intersection
                                                                ! occurred, else False
8521
8750 end function TriTriIntersect

8752 function TriPointDistance(testTri,point,optNormDir)
8754     ! Function: Returns the distance above of below the plane of a given triangle
8756     ! Inputs:
8757     real(8),               intent(in) :: testTri(3,XYZ) ! Triangle to test
8758     real(8),               intent(in) :: point(3)      ! Point to test
8759     real(8), optional, intent(in) :: optNormDir(3)    ! Optional chance to provide the
                                                                ! normal dir of the triangle to reduce
                                                                ! computation effort

8760     ! Returns:
8761     real(8)                               :: TriPointDistance ! Perpendicular distance of point to
                                                                ! plane of testTri

8778 end function TriPointDistance
8832 end function GetBodyType

8836 subroutine DumpPhysicsDataWorkpiece(contactStatus,dp,
8837     *,                               thisVol, totalVol)
8839     ! Function: Writes incremental physics data about the workpiece
8840     ! to a local csv file
8842     ! Outputs:
8843     logical, intent(in) :: contactStatus ! True if workpiece is in contact with any cutter
8844     real(8), intent(in) :: dp(SIM_DOF)  ! Total load acting on workpiece
8845     real(8), intent(in) :: thisVol       ! Volume removed this increment
8846     real(8), intent(in) :: totalVol      ! Total volume removed this simulation
8918 end subroutine DumpPhysicsDataWorkpiece

8922 subroutine DumpCutterFaceProfile(pCutter,
8923     *,                               profilePoints,nProfilePoints)
8925     ! Function: Writes out the cutter face intersection profile data to a file
8928     ! Inputs / Outputs:
8929     type(CUTTER), pointer, intent(inout) :: pCutter        ! This cutter
8930     real(8), allocatable, intent(in)   ) :: profilePoints(:, :) ! Profile line points
8931     integer,               intent(in)   ) :: nProfilePoints ! Number of profile points
8983 end subroutine DumpCutterFaceProfile

8985 subroutine DeleteFile(fileName)
8988     ! Function: Attempts to delete a file, retrying every second until success
8990     ! Input:
8991     character(*), intent(in) :: fileName ! The path to the file to delete
9023 end subroutine DeleteFile

9025 subroutine ProtectedWrite(fileName,strData,optFileUnit)
9027     ! Function: Attempts to write given data to a specified file
9028     ! using an optional unit number.
9029     ! If it fails to open the file or fails to write to
9030     ! the file, it will keep trying every second to infinity
9032     ! Inputs:
9033     character(*),               intent(in) :: fileName    ! Path to file to write to
9034     character(*),               intent(in) :: strData      ! Data to write to file
9035     integer, optional, value, intent(in) :: optFileUnit ! Optional unit file number to use
9109 end subroutine ProtectedWrite

9143 function TetMesh(points,
9144     *,                               facetsInt,    nFacets,

```

```

9145      *      protoTets,      nTets,
9146      *      keepFacets) result (errCode)
9148      ! Function: Generates a tetrahedral mesh inside a given hull of facets
9149      ! Inputs / Outputs:
9150      real(8), allocatable, intent(inout) :: points(:, :) ! points(nPoints,3): A list of n
! coordinates, that may or may
! not be referenced by
! triangles.
9151      ! If the TetMesh generates
! additional points, they will
! be added to points.
9152      integer, allocatable, intent(inout) :: facetsInt(:, :) ! facetsInt(nFacets,3)
9153      ! A list of corner nodes that
! make up triangles. If the
! value is +ve, the number
! refers to nodeIndex, else the
! absolute number refers to an
! offset in points
9154      ! There is guaranteed to be one
! and only one tet that shares a
! face with each
9155      ! of the supplied triangles.
! Many will generate naturally
! during the meshing
9156      ! process, but if one fails to
! generate, the tets occupying
! the contested volume will be
! removed and the
9157      ! remaining hull will be mesh
! with a different strategy
! ("Gift Wrapping")
9158      ! New triangles formed by the
! addition of tets or points
! will NOT be appended to
! triangles.
9159      integer, intent(inout) :: nFacets ! Number of facets in facet list
9160      integer, allocatable, intent( out) :: protoTets(:, :) ! List of tetrahedrals (may
! include disabled tets)
9161      integer, intent(inout) :: nTets ! Number of tets in protoTets
9162      integer , intent(in ) :: keepFacets(:) ! List of facets that must
! appear in the final mesh
9164      ! Returns:
9165      integer :: errCode ! Error code (see WARN_* flags)
11796 end function TetMesh

11798 function Visible(pntIdx, rootFacet,
11799      *      facetsInt, facetsReal,
11800      *      nFacets, pointsReal,
11801      *      optMarkOccluded)
11803      ! Function: Test to see if a point in pointsReal can be seen from a facet in facetsInt
11805      ! Inputs / Outputs:
11806      integer, intent(in ) :: pntIdx ! Point to test
11807      integer, intent(in ) :: rootFacet ! Facet to test
11808      integer, allocatable, intent(inout) :: facetsInt(:, :) ! Facet list
11809      real(8), allocatable, intent(in ) :: facetsReal(:, :) ! Facet list real data
11810      integer, intent(in ) :: nFacets ! Number of facets in
! facet list
11811      real(8), intent(in ) :: pointsReal(:, :) ! Points list real data
11812      logical, optional, value, intent(in ) :: optMarkOccluded ! Optional flag to have
! this
11813      ! function flag occluding
! facets
11814
11816      ! Returns:
11817      logical :: Visible ! True if visible, else
! False
11890 end function Visible

11892 function TetMesh_DeleteTet(tetIdx, protoTets,
11893      *      facetsInt, facetsReal, nFacets,
11894      *      illegalTets, nIllegalTets,
11895      *      pointsReal,
11896      *      nExternalFacets, forceRecycle)
11897      *      result (errCode)
11899      ! Function: Deletes a tetrahedral from a mesh and frees facet sides that were occupied
11900      ! by the tetrahedral.
11901      ! Inputs / Outputs:
11902      integer, intent(in ) :: tetIdx ! Index of tet to delete
11903      ! in protoTets
11904      integer, allocatable, intent(inout) :: protoTets(:, :) ! Tetrahedral list

```

```

11905     integer, allocatable, intent(inout) :: facetsInt(:, :) ! Facet list
11906     real(8), allocatable, intent(inout) :: facetsReal(:, :) ! Facet coordinate list
11907     integer, intent(in) :: nFacets ! Number of facets in
11908                                     ! facet list
11909     integer, allocatable, intent(inout) :: illegalTets(:, :) ! List of prohibited
11910                                     ! tetrahedral formulations
11911     integer, intent(inout) :: nIllegalTets ! Number of prohibited
11912                                     ! tetrahedral formulations
11913     real(8), allocatable, intent(in) :: pointsReal(:, :) ! Point list coordinates
11914     integer, intent(in) :: nExternalFacets ! Number of external facets
11915     logical, intent(inout) :: forceRecycle ! Output to notify calling
11916                                     ! function (TetMesh) that
11917                                     ! DeleteMesh has done
11918                                     ! something
11919                                     ! that could result in new
11920                                     ! elements being required
11921                                     ! (e.g. exposed a new face)
11921     ! Returns:
11922     integer :: errCode ! Error code
11923 end function TetMesh_DeleteTet

12062 subroutine MakeTAPCache(TAPCacheMap, TAPCache,
12063 *      protoTets, nTets,
12064 *      nCachePoints
12065 *      )
12066     ! Function: Create the Tetrahedral At Point Cache.
12067     ! The result is TAPCacheMap whose indices correspond
12068     ! to point IDs and whose values indicate the start address
12069     ! in TAPCache of a subarray of tetrahedral IDs associated with
12070     ! the point
12071     ! The length of the subarray is limited by the address associated
12072     ! with the next point (or the end of TAPCache)
12073     ! Inputs / Outputs:
12074     integer, allocatable, intent(out) :: TAPCacheMap(:) ! Map of nodes IDs to
12075                                     ! TAPCache start offsets
12076     integer, allocatable, intent(out) :: TAPCache(:) ! List of tetrahedrals
12077     integer, intent(in) :: protoTets(:, :) ! Tetrahedrals to map
12078     integer, intent(in) :: nTets ! Number of tetrahedrals to map
12079     integer, intent(in) :: nCachePoints ! Number of points to map
12080 end subroutine MakeTAPCache

12140 subroutine MakeFAPCache(FAPCacheMap, FAPCache,
12141 *      facetsInt, nFacets,
12142 *      optNCachePoints
12143 *      )
12144     ! Function: Create the Facets at Point Cache
12145     ! The result is FAPCacheMap whose indices correspond
12146     ! to point IDs and whose values indicate the start address
12147     ! in FAPCache of a subarray of facet IDs associated with the
12148     ! the point
12149     ! The length of the subarray is limited by the address associates with
12150     ! the next point (or the end of FAPCache)
12151     ! Inputs / Outputs:
12152     integer, allocatable, intent(out) :: FAPCacheMap(:) ! Map of point IDs to FAPCache
12153                                     ! start offsets
12154     integer, allocatable, intent(out) :: FAPCache(:) ! List of facets
12155     integer, intent(in) :: facetsInt(:, :) ! Facets to map
12156     integer, intent(in) :: nFacets ! Number of facets
12157     integer, optional, intent(in) :: optNCachePoints ! Optional number of points to
12158                                     ! map if omitted, determined
12159                                     ! automatically from highest ID
12160                                     ! references by facetsInt
12161 end subroutine MakeFAPCache

12241 subroutine MakeSAPCache(SAPCacheMap, SAPCache,
12242 *      segments, nSegments,
12243 *      nCachePoints
12244 *      )
12245     ! Function: Create the Segment (Edge) At Point Cache. The result is SAPCacheMap whose
12246     ! indices correspond to point IDs and whose values indicate the start address in
12247     ! SAPCache of a subarray of segment IDs associated with the point.
12248     ! The length of the subarray is limited by the address associated with the next point
12249     ! (or the end of SAPCache)
12250     ! Inputs / Outputs:
12251     integer, allocatable, intent(out) :: SAPCacheMap(:) ! Map of nodes IDs to
12252                                     ! SAPCache start offsets
12253     integer, allocatable, intent(out) :: SAPCache(:) ! List of segment IDs
12254     integer, intent(in) :: segments(:, :) ! Segments to map
12255     integer, intent(in) :: nSegments ! Number of segments in segments
12256     integer, intent(in) :: nCachePoints ! Number of points to map
12257 end subroutine MakeSAPCache

```

```

12318 recursive subroutine KeepKillWalker(rootFacet,
12319 *                                     FAPCache, FAPCacheMap,
12320 *                                     facetsInt, nFacets)
12322 ! Function: Walks facets in facetsInt, starting from rootFacet without
12323 ! crossing an edge thats shared by 3 or more facets.
12324 ! Stepped on facets are marked as Keep
12325 ! Inputs / Outputs:
12326 integer,          intent(in)    :: rootFacet      ! Known keep facet to start from
12327 integer, allocatable, intent(inout) :: FAPCacheMap(:) ! Facets at point cache map
12328 integer, allocatable, intent(inout) :: FAPCache(:)  ! Facets at point cache
12329 integer, allocatable, intent(inout) :: facetsInt(:, :) ! Facet list
12330 integer,          intent(in)    :: nFacets        ! Number of facets in facet list
12408 end subroutine KeepKillWalker

12410 recursive subroutine SideWalker(rootFacet, paintBrush,
12411 *                               FAPCache, FAPCacheMap,
12412 *                               facetsInt, nFacets)
12414 ! Function: Designates a side to facets in facetsInt such that all contiguous
12415 ! facets forming a hull have a common external and internal side.
12416 ! For some facets, side 1 is internal, for others side 2 is internal
12417 ! A seed facet and initial internal/external definition in paintBrush is
12418 ! used to designate the first side. Further facets are walked and the
12419 ! paintBrush is inverted every time the algorithm steps on to a facet
12420 ! that defines a common edge in the same order as the departing facet
12421 ! Inputs / Outputs:
12422 integer,          intent(in)    :: rootFacet      ! Seed facet with known outside
12423 integer, value,    intent(in)    :: paintBrush     ! Which side to paint side one
12424 integer, allocatable, intent(inout) :: FAPCache(:) ! Facets at point cache
12425 integer, allocatable, intent(inout) :: FAPCacheMap(:) ! Facets at point cache map
12426 integer, allocatable, intent(inout) :: facetsInt(:, :) ! Facet list
12427 integer,          intent(in)    :: nFacets        ! Number of facets in facet list
12504 end subroutine SideWalker

12506 subroutine GiftWrap(points,
12507 *                  triangles, nTriangles,
12508 *                  meshRays, nMeshRays,
12509 *                  planeNormal )
12511 ! Function: Generates a 2D mesh in R3 using the Gift Wrapping algorithm and a list
12512 ! of rays(edges) and points.
12513 ! Thesis note: We used to use the Bowyer Watson algorithm
12514 ! https://en.wikipedia.org/wiki/Bowyer%E2%80%93Watson\_algorithm
12515 ! https://www.mathopenref.com/trianglecircumcircle.html
12516 ! Inputs / Outputs:
12517 real(8), allocatable, intent(inout) :: points(:, :) ! (n:3) XYH or XYZ
12518 integer, allocatable, intent(inout) :: triangles(:, :) ! (n:3) Vertices
12519 integer,          intent(out)    :: nTriangles      ! Number of triangles
12520 integer,          intent(inout) :: meshRays(:)      ! Edges are dealt
12521 type(RAY), allocatable, target, intent(inout) :: meshRays(:) ! with as rays
12522 integer,          intent(inout) :: nMeshRays        ! Number of mesh rays
12523 real(8),          intent(in)    :: planeNormal(3) ! Normal vector of
12524 integer,          intent(inout) :: nMeshRays        ! Number of mesh rays
12525 real(8),          intent(in)    :: planeNormal(3) ! Normal vector of
12526 integer,          intent(inout) :: meshRays(:)      ! Edges are dealt
12527 type(RAY), allocatable, target, intent(inout) :: meshRays(:) ! with as rays
12846 end subroutine GiftWrap

12848 subroutine SplitMesh()
12852 ! Functions: -Main function in this program. Split mesh does the following:
12853 ! -Identifies a sub-mesh of elements in the path of the cutter
12854 ! -Splits the sub-mesh and removes the chip side
12855 ! -Simplifies the modified sub-mesh
12856 ! -Creates a new volume mesh for the sub-mesh
12857 ! -Calculates the cutting forces and torques according to
12858 ! the cutter-workpiece intersection profile
12859 ! -Applies the workpiece reaction load
12860 ! -Modified elementIndex and nodeIndex to contain the new mesh
12861 end subroutine SplitMesh

12862

12863

12864

12865

12866

12867

12868

12869

12870

12871

12872

12873

12874

12875

12876

12877

12878

12879

12880

12881

12882

12883

12884

12885

12886

12887

12888

12889

12890

12891

12892

12893

12894

12895

12896

12897

12898

12899

12900

12901

12902

12903

12904

12905

12906

12907

12908

12909

12910

12911

12912

12913

12914

12915

12916

12917

12918

12919

12920

12921

12922

12923

12924

12925

12926

12927

12928

12929

12930

12931

12932

12933

12934

12935

12936

12937

12938

12939

12940

12941

12942

12943

12944

12945

12946

12947

12948

12949

12950

12951

12952

12953

12954

12955

12956

12957

12958

12959

12960

12961

12962

12963

12964

12965

12966

12967

12968

12969

12970

12971

12972

12973

12974

12975

12976

12977

12978

12979

12980

12981

12982

12983

12984

12985

12986

12987

12988

12989

12990

12991

12992

12993

12994

12995

12996

12997

12998

12999

13000

13001

13002

13003

13004

13005

13006

13007

13008

13009

13010

13011

13012

13013

13014

13015

13016

13017

13018

13019

13020

13021

13022

13023

13024

13025

13026

13027

13028

13029

13030

13031

13032

13033

13034

13035

13036

13037

13038

13039

13040

13041

13042

13043

13044

13045

13046

13047

13048

13049

13050

13051

13052

13053

13054

13055

13056

13057

13058

13059

13060

13061

13062

13063

13064

13065

13066

13067

13068

13069

13070

13071

13072

13073

13074

13075

13076

13077

13078

13079

13080

13081

13082

13083

13084

13085

13086

13087

13088

13089

13090

13091

13092

13093

13094

13095

13096

13097

13098

13099

13100

13101

13102

13103

13104

13105

13106

13107

13108

13109

13110

13111

13112

13113

13114

13115

13116

13117

13118

13119

13120

13121

13122

13123

13124

13125

13126

13127

13128

13129

13130

13131

13132

13133

13134

13135

13136

13137

13138

13139

13140

13141

13142

13143

13144

13145

13146

13147

13148

13149

13150

13151

13152

13153

13154

13155

13156

13157

13158

13159

13160

13161

13162

13163

13164

13165

13166

13167

13168

13169

13170

13171

13172

13173

13174

13175

13176

13177

13178

13179

13180

13181

13182

13183

13184

13185

13186

13187

13188

13189

13190

13191

13192

13193

13194

13195

13196

13197

13198

13199

13200

13201

13202

13203

13204

13205

13206

13207

13208

13209

13210

13211

13212

13213

13214

13215

13216

13217

13218

13219

13220

13221

13222

13223

13224

13225

13226

13227

13228

13229

13230

13231

13232

13233

13234

13235

13236

13237

13238

13239

13240

13241

13242

13243

13244

13245

13246

13247

13248

13249

13250

13251

13252

13253

13254

13255

13256

13257

13258

13259

13260

13261

13262

13263

13264

13265

13266

13267

13268

13269

13270

13271

13272

13273

13274

13275

13276

13277

13278

13279

13280

13281

13282

13283

13284

13285

13286

13287

13288

13289

13290

13291

13292

13293

13294

13295

13296

13297

13298

13299

13300

13301

13302

13303

13304

13305

13306

13307

13308

13309

13310

13311

13312

13313

13314

13315

13316

13317

13318

13319

13320

13321

13322

13323

13324

13325

13326

13327

13328

13329

13330

13331

13332

13333

13334

13335

13336

13337

13338

13339

13340

13341

13342

13343

13344

13345

13346

13347

13348

13349

13350

13351

13352

13353

13354

13355

13356

13357

13358

13359

13360

13361

13362

13363

13364

13365

13366

13367

13368

13369

13370

13371

13372

13373

13374

13375

13376

13377

13378

13379

13380

13381

13382

13383

13384

13385

13386

13387

13388

13389

13390

13391

13392

13393

13394

13395

13396

13397

13398

13399

13400

13401

13402

13403

13404

13405

13406

13407

13408

13409

13410

13411

13412

13413

13414

13415

13416

13417

13418

13419

13420

13421

13422

13423

13424

13425

13426

13427

13428

13429

13430

13431

13432

13433

13434

13435

13436

13437

13438

13439

13440

13441

13442

13443

13444

13445

13446

13447

13448

13449

13450

13451

13452

13453

13454

13455

13456

13457

13458

13459

13460

13461

13462

13463

13464

13465

13466

13467

13468

13469

13470

13471

13472

13473

13474

13475

13476

13477

13478

13479

13480

13481

13482

13483

13484

13485

13486

13487

13488

13489

13490

13491

13492

13493

13494

13495

13496

13497

13498

13499

13500

13501

13502

13503

13504

13505

13506

13507

13508

13509

13510

13511

13512

13513

13514

13515

13516

13517

13518

13519

13520

13521

13522

13523

13524

13525

13526

13527

13528

13529

13530

13531

13532

13533

13534

13535

13536

13537

13538

13539

13540

13541

13542

13543

13544

13545

13546

13547

13548

13549

13550

13551

13552

13553

13554

13555

13556

13557

13558

13559

13560

13561

13562

13563

13564

13565

13566

13567

13568

13569

13570

13571

13572

13573

13574

13575

13576

13577

13578

13579

13580

13581

13582

13583

13584

13585

13586

13587

13588

13589

13590

13591

13592

13593

13594

13595

13596

13597

13598

13599

13600

13601

13602

13603

13604

13605

13606

13607

13608

13609

13610

13611

13612

13613

13614

13615

13616

13617

13618

13619

13620

13621

13622

13623

13624

13625

13626

13627

13628

13629

13630

13631

13632

13633

13634

13635

13636

13637

13638

13639

13640

13641

13642

13643

13644

13645

13646

13647

13648

13649

13650

13651

13652

13653

13654

13655

13656

13657

13658

13659

13660

13661

13662

13663

13664

13665

13666

13667

13668

13669

13670

13671

13672

13673

13674

13675

13676

13677

13678

13679

13680

13681

13682

13683

13684

13685

13686

13687

13688

13689

13690

13691

13692

13693

13694

13695

13696

13697

13698

13699

13700

13701

13702

13703

13704

13705

13706

13707

13708

13709

13710

13711

13712

13713

13714

13715

13716

13717

13718

13719

13720

13721

13722

13723

13724

13725

13726

13727

13728

13729

13730

13731

13732

13733

13734

13735

13736

13737

13738

13739

13740

13741

13742

13743

13744

13745

13746

13747

13748

13749

13750

13751

13752

13753

13754

13755

13756

13757

13758

13759

13760

13761

13762

13763

13764

13765

13766

13767

13768

13769

13770

13771

13772

13773

13774

13775

13776

13777

13778

13779

13780

13781

13782

13783

13784

13785

13786

13787

13788

13789

13790

13791

13792

13793

13794

13795

13796

13797

13798

13799

13800

13801

13802

13803

13804

13805

13806

13807

13808

13809

13810

13811

13812

13813

13814

13815

13816

13817

13818

13819

13820

13821

13822

13823

13824

13825

13826

13827

13828

13829

13830

13831

13832

13833

13834

13835

13836

13837

13838

13839

13840

13841

13842

13843

13844

13845

13846

13847

13848

13849

13850

13851

13852

13853

13854

13855

13856

13857

13858

13859

13860

13861

13862

13863

13864

13865

13866

13867

13868

13869

13870

13871

13872

13873

13874

13875

13876

13877

13878

13879

13880

13881

13882

13883

13884

13885

13886

13887

13888

13889

13890

13891

13892

13893

13894

13895

13896

13897

13898

13899

13900

13901

13902

13903

13904

13905

13906

13907

13908

13909

13910

13911

13912

13913

13914

13915

13916

13917

13918

13919

13920

13921

13922

13923

13924

13925

13926

13927

13928

13929

13930

13931

13932

13933

13934

13935

13936

13937

13938

13939

13940

13941

13942

13943

13944

13945

13946

13947

13948

13949

13950

13951

13952

13953

13954

13955

13956

13957

13958

13959

13960

13961

13962

13963

13964

13965

13966

13967

13968

13969

13970

13971

13972

13973

13974

13975

13976

13977

13978

13979

13980

13981

13982

13983

13984

13985

13986

13987

13988

13989

13990

13991

13992

13993

13994

13995

13996

13997

13998

13999

14000

14001

14002

14003

14004

14005

14006

14007

14008

14009

14010

14011

14012

14013

14014

14015

14016

14017

14018

14019

14020

14021

14022

14023

14024

14025

14026

14027

14028

14029

14030

14031

14032

14033

14034

14035

14036

14037

14038

14039

14040

14041

14042

14043

14044

14045

14046

14047

14048

14049

14050

14051

14052

14053

14054

14055

14056

14057

14058

14059

14060

14061

14062

14063

14064

14065

14066

14067

14068

14069

14070

14071

14072

14073

14074

14075

14076

14077

14078

14079

14080

14081

14082

14083

14084

14085

14086

14087

14088

14089

14090

14091

14092

14093

14094

14095

14096

14097

14098

14099

14100

14101

14102

14103

14104

14105

14106

14107

14108

14109

14110

14111

14112

14113

14114

14115

14116

14117

14118

14119

14120

14121

14122

14123

14124

14125

14126

14127

14128

14129

14130

14131

14132

14133

14134

14135

14136

14137

14138

14139

14140

14141

14142

14143

14144

14145

14146

14147

14148

14149

14150

14151

14152

14153

14154

14155

14156

14157

14158

14159

14160

14161

14162

14163

14164

14165

14166

14167

14168

14169

14170

14171

14172

14173

14174

14175

14176

14177

14178

14179

14180

14181

14182

14183

14184

14185

14186

14187

14188

14189

14190

14191

14192

14193

14194

14195

14196

14197

14198

14199

14200

14201

14202

14203

14204

14205

14206

14207

14208

14209

14210

14211

14212

14213

14214

14215

14216

14217

14218

14219

14220

14221

14222

14223

14224

14225

14226

14227

14228

14229

14230

14231

14232

14233

14234

14235

14236

14237

14238

14239

14240

14241

14242

14243

14244

14245

14246

14247

14248

14249

14250

14251

14252

14253

14254

14255

14256

14257

14258

14259

14260

14261

14262

14263

14264

14265

14266

14267

14268

14269

14270

14271

14272

14273

14274

14275

14276

14277

14278

14279

14280

14281

14282

14283

14284

14285

14286

14287

14288

14289

14290

14291

14292

14293

14294

14295

14296

14297

14298

14299

14300

14301

14302

14303

14304

14305

14306

14307

14308

14309

14310

14311

14312

14313

14314

14315

14316

14317

14318

14319

14320

14321

14322

14323

14324

14325

14326

14327

14328

14329

14330

14331

14332

14333

14334

14335

14336

14337

14338

14339

14340

14341

14342

14343

14344

14345

14346

14347

14348

14349

14350

14351

14352

14353

14354

14355

14356

14357

14358

14359

14360

14361

14362

14363

14364

14365

14366

14367

14368

14369

14370

14371

14372

14373

14374

14375

14376

14377

14378

14379

14380

14381

14382

14383

14384

14385

14386

14387

14388

14389

14390

14391

14392

14393

14394

14395

14396

14397

14398

14399

14400

14401

14402

14403

14404

14405

14406

14407

14408

14409

14410

14411

14412

14413

14414

14415

14416

14417

14418

14419

14420

14421

14422

14423

14424

14425

14426

14427

14428

14429

14430

14431

14432

14433

14434

14435

14436

14437

14438

14439

14440

14441

14442

14443

14444

14445

14446

14447

14448

14449

14450

14451

14452

14453

14454

14455

14456

14457

14458

14459

14460

14461

14462

14463

14464

14465

14466

14467

14468

14469

14470

14471

14472

14473

14474

14475

14476

14477

14478

14479

14480

14481

14482

14483

14484

14485

14486

14487

14488

14489

14490

14491

14492

14493

14494

14495

14496

14497

14498

14499

14500

14501

14502

14503

14504

14505

14506

14507

14508

14509

14510

14511

14512

14513

14514

14515

14516

14517

14518

14519

14520

14521

14522

14523

14524

14525

14526

14527

14528

14529

14530

14531

14532

14533

14534

14535

14536

14537

14538

14539

14540

14541

14542

14543

14544

14545

14546

14547

14548

14549

14550

14551

14552

14553

14554

14555

14556

14557

14558

14559

14560

14561

14562

14563

14564

14565

14566

14567

14568

14569

14570

14571

14572

14573

14574

14575

14576

14577

14578

14579

14580

14581

14582

14583

14584

14585

14586

14587

14588

14589

14590

14591

14592

14593

14594

14595

14596

14597

14598

14599

14600

14601

14602

14603

14604

14605

14606

14607

14608

14609

14610

14611

14612

14613

14614

14615

14616

14617

14618

14619

14620

14621

14622

14623

14624

14625

14626

14627

14628

14629

14630

14631

14632

14633

14634

14635

14636

14637

14638

14639

14640

14641

14642

14643

14644

14645

14646

14647

14648

14649

14650

14651

14652

14653

14654

14655

14656

14657

14658

14659

14660

14661

14662

14663

14664

14665

14666

14667

14668

14669

14670

14671

14672

14673

14674

14675

14676

14677

14678

14679

14680

14681

14682

14683

14684

14685

14686

14687

14688

14689

14690

14691

14692

14693

14694

14695

14696

14697

14698

14699

14700


```

```

15992      *      segmentIdx,altSegment)
15994      ! Function: Calculates the counter segment of two adjacent facets
15995      ! Inputs / Outputs:
15996      integer, allocatable, intent(in) :: facetsInt(:, :) ! Facet list
15997      integer, allocatable, intent(in) :: FASCacheInt(:, :) ! Facet cache
15998      integer, intent(in) :: segmentIdx ! ID of current segment
15999      integer, intent(inout) :: altSegment(2) ! Counter segment node Ids
16017  end subroutine CounterSegment
16163  end module JamesMod

16169  subroutine MOTION(x,f,v,time,dTime,nsurf,inc_X)
16170  use JamesMod
16172  ! Function: MARC User Subroutine entry point
16173  ! Used to set the positions / velocities of rigid bodies
16174  ! See page 118 - MARC 2016 Volume D
16176  ! Inputs / Outputs:
16177  real(8), intent(inout) :: x(*) ! Array defining body coordinates
16178  real(8), intent(in) :: f(*) ! Array of current body loads
16179  real(8), intent(inout) :: v(*) ! Array of current body velocities
16180  real(8), intent(in) :: time ! Time at which data is requested
16181  real(8), intent(in) :: dTime ! Current time increment
16182  integer, intent(in) :: nsurf ! Surface number for which data is requested
16183  integer, intent(in) :: inc_X ! Increment number
16225  end subroutine MOTION

16232  subroutine FORCEM(press,th1,th2,nn,n)
16233  use JamesMod
16239  ! Intro: MARC User Subroutine entry point
16240  ! Used to assign element distributed loads
16241  ! See page 58 - MARC 2016 Volume D
16243  ! Function: MARC User Subroutine entry point
16244  ! Used to set elemental pressure loads
16245  ! See page 58 - MARC 2016 Volume D
16252  ! Inputs / Outputs:
16253  real(8), intent(out) :: press ! Magnitude of distributed load to be applied
16254  real(8), intent(in) :: th1(SIM_DIM) ! First coordinate of the integration point
16255  real(8), intent(in) :: th2(SIM_DIM) ! Second coordinate of the integration point
16256  integer, intent(in) :: nn ! Integration point number
16257  integer, intent(in) :: n(10) ! (mixed parameters - see user guide)
16259  ! Outputs:
16260  real(8) :: prnorm ! (special output via common var) Direction
16261  ! of distributed load
16306  end subroutine FORCEM

16312  subroutine UMAKNET(ido,iflag,ncrdmx_X,ndegmx_X,
16313  *      numnp_X,numel_X,ndeg_X,ncrd_X,
16314  *      iel_type,nnodmx_X,numelmx_X,neltab_X,
16315  *      xord,disp,ielcon_X,ieltab_X,fileName)
16316  use JamesMod
16318  ! Intro: MARC User Subroutine entry point
16319  ! Used to write out new contact body mesh to local
16320  ! file for MARC to read back in at the start of next inc
16321  ! See page 410 - MARC 2016 Volume D
16323  ! Function: On increment zero:
16324  ! -Return iflag=0 to allow MARC to generate it's own start mesh
16325  ! On all other increments:
16326  ! -Reset element and node states in element and node indexes
16327  ! -Update the cached positions of cutters
16328  ! -Call SplitMesh
16329  ! -Generate and write out new mesh from the product of SplitMesh
16330  ! -Reorder the contents of local element and node indexes to match
16331  ! how MARC will organise it's own lists after reading in the new
16332  ! mesh file
16336  ! Inputs / Outputs:
16337  integer, intent(in) :: ido ! 2=3D, 3=3D
16338  integer, intent(out) :: iflag ! 1 If subroutine used, else 0
16339  integer, intent(in) :: ncrdmx_X ! Max number of coordinate
! components
16340  integer, intent(in) :: ndegmx_X ! Max number of degrees of freedom
16341  integer, intent(in) :: numnp_X ! Total number of nodes
16342  integer, intent(in) :: numel_X ! Total number of elements
16343  integer, intent(in) :: ndeg_X ! Number of degrees of freedom
16344  integer, intent(in) :: ncrd_X ! Number of coordinate components
16345  ! (2D=2, 3D=3)
16346  integer, intent(in) :: iel_type ! Element type
16347  integer, intent(in) :: nnodmx_X ! Max number of nodes per element
16348  integer, intent(in) :: numelmx_X ! Max number of elements
16349  integer, intent(in) :: neltab_X ! Size of ieltab
16350  real(8), intent(in) :: xord(ncrdmx_X,*) ! Nodal coordinates

```

```

16351     real(8),      intent(in  ) :: disp(ndegmx_X,*)      ! Nodal displacements
16352     integer,      intent(in  ) :: ielcon_X(nnodmx_X,*)   ! Current element connectivity
16353     integer,      intent(in  ) :: ieltab_X(neltab_X,*)   ! Element group information
16354     character(*), intent(inout) :: fileName              ! Remeshing file name
16963 end subroutine UMAKNET

16968
16969 *      subroutine PLOTV(v,s,sp,etot,eplas,ecreep,t,m,nn,
16970 *                      kcus,ndi_X,nshear_X,jplbcd)
16971     use JamesMod
16972     ! Intro: MARC User Subroutine entry point
16973     ! Used to define element variables to be written to post file
16974     ! See page 451 - MARC 2016 Volume D
16975     ! Function: Used to set element variables from cached values stored in
16976     ! element index
16977     ! Inputs / Outputs:
16978     real(8), intent( out) :: v          ! Variable to be written to post file
16979     real(8), intent(in  ) :: s(*)       ! Array of stresses at this integration point
16980     real(8), intent(in  ) :: sp(*)      ! Array of stresses in preferred direction
16981     real(8), intent(in  ) :: etot(*)    ! Total strain at this integration point
16982     real(8), intent(in  ) :: eplas(*)   ! Total plastic strain at this integration point
16983     real(8), intent(in  ) :: ecreep(*)  ! Total creep strain at this integration point
16984     real(8), intent(in  ) :: t(*)       ! Array of state variables at this integration
16985                                         ! point
16986     integer, intent(in  ) :: m(2)      ! m(1/2): External/internal element number
16987     integer, intent(in  ) :: nn        ! Integration point number
16988     integer, intent(in  ) :: kcus(2)   ! kcus(1/2): User/internal layer number
16989     integer, intent(in  ) :: ndi_X     ! Number of direct stresses
16990     integer, intent(in  ) :: nshear_X  ! Number of shear stresses
16991     integer, intent(in  ) :: jplbcd    ! Absolute value of user's entered code
17050 end subroutine PLOTV

17055
17056 *      subroutine UPSTNO(nqcode,nodeExt,verno,nqncomp,nqtype,
17057 *                      ngaver,nqcomptype,nqdatatype,
17058 *                      nqcompname)
17059     use JamesMod
17060     ! Intro: MARC User Subroutine entry point
17061     ! Used to define nodal scalars and vectors
17062     ! to be written to post file
17063     ! See page 455 - MARC 2016 Volume D
17064     ! <OPTIMISATION REPORT>
17065     ! CALL SCALE: S M [L]
17066     ! CALL TYPE: For all nodes, for all scalar and vector quantities
17067     ! Inputs:
17068     integer, intent(in  ) :: nqcode     ! User nodal post code, e.g. -1
17069     integer, intent(in  ) :: nodeExt    ! External node id
17070     real(8), intent( out) :: verno(*)   ! nodal values
17071                                         ! real/imag verno(      1: nqncomp) real
17072                                         ! verno(nqncomp+1:2*nqncomp) imag
17073                                         ! magn/phas verno(      1: nqncomp) magn
17074                                         ! verno(nqncomp+1:2*nqncomp) phas
17075     integer, intent( out) :: nqncomp    ! Number of values in verno
17076     integer, intent( out) :: nqtype     ! 0 = scalar
17077                                         ! 1 = vector
17078     integer, intent( out) :: ngaver     ! only for DDM 0 = sum over domains
17079                                         ! 1 = average over domains
17080     integer, intent( out) :: nqcomptype ! 0 = global coordinate system (x,y,z)
17081                                         ! 1 = shell (top,bottom,middle)
17082                                         ! 2 = order (1,2,3)
17083     integer, intent( out) :: nqdatatype ! 0 = default
17084                                         ! 1 = modal
17085                                         ! 2 = buckle
17086                                         ! 3 = harmonic real
17087                                         ! 4 = harmonic real/imaginary
17088                                         ! 5 = harmonic magnitude/phase
17089     integer, intent(in  ) :: nqcompname ! not used (future expansion)
17169 end subroutine UPSTNO

17175
17176 *      subroutine UBGINC(inc_X,incsub_X)
17177     use JamesMod
17178     ! Intro: MARC User Subroutine entry point
17179     ! Called at the beginning of every increment
17180     ! See page 470 - MARC 2016 Volume D
17181     ! Function: -Calls Setup on increment 0
17182     ! -If in debug mode, validates element and node indices every increment thereafter
17183     ! <OPTIMISATION REPORT>
17184     ! CALL SCALE: [S] M L
17185     ! CALL TYPE: Once per increment start
17186     ! Inputs:
17187     integer, intent(in) :: inc_X      ! Increment number

```

```

17190      integer, intent(in) :: incsub_X ! Sub-increment number
17427 end subroutine UBGINC

17432 subroutine UEDINC(inc_X,incsub_X)
17433   use JamesMod
17435   ! Intro: MARC User Subroutine entry point
17436   ! Called at the start of each increment
17437   ! See page 471 - MARC 2016 Volume D
17438   ! Function: Used to print statistics to log
17444   ! Inputs:
17445   integer, intent(in) :: inc_X      ! Increment number
17446   integer, intent(in) :: incsub_X ! Sub-increment number
17458 end subroutine UEDINC

```