Diploma Thesis

FE - simulation of chip formation in inhomogenous materials

Author:
Stefan Distlberger

Supervisor:
Dr. Werner Ecker

A thesis submitted in fulfilment of the requirements for the degree of Master of Science

at the

Montanuniversität Leoben
Materials Center Leoben

April 2014

Affidavit

I, Stefan Distlberger, declare that this thesis titled 'FE - simulation of chip formation in inhomogenous materials' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ____________________________________

Date: ____________________________________

"Thanks to my solid academic training, today I can walk around the earth, and enjoy the wonders and the complexity of nature, science and life itself."

Stefan Distlberger

MONTANUNIVERSITAET LEOBEN

Abstract

Institute of Mechanics
Materials Center Leoben

Master of Science

FE - simulation of chip formation in inhomogenous materials

by Stefan Distlberger

The chip formation in today's machining is a complex process that has not been completely understood yet. The process is defined by friction, chip formation, material, cutting force, hole inclusions in the cut material and many more aspects. Correlations between some of these aspects are analyzed in this thesis by using the commercial finite-element software ABAQUS$^{\mathrm{TM}}$/Standard. This thesis presents a survey of today's simulation techniques and ways to facilitate chip formation simulations. A semi automated tool is developed which allows to simulate a 2D chip formation of a homogeneous material or an inhomogeneous material in ABAQUS$^{\mathrm{TM}}$/Standard. The tool automates the remeshing process to avoid severe element distortion. It also automates the time consuming manual process of the modeling itself. The pertaining routines are coded in Python and Fortran. To visualize results, the author provides a further tool for rendering videos as well as utilities for investigating certain aspects of the process, such as the change of the chip's width or the contact forces over time. The influences of three different element sizes are discussed: A decreasing element size results in a decrease of the chip's width, an increased rake angle of the chip and a decrease of the tool's contact force. A homogeneous specimen with decreasing chip thickness is simulated to investigate the decrease of the tool's force and the decrease of the chip's width over time. Two models are compared: The first one shows chip formation with the chip getting in contact with itself. The second model shows chip formation where the chip is trimmed before self contact can occur. Both models show a change in the chip's width and contact length between tool and workpiece. An inhomogeneous model is presented to study the effects of an inclusion moving through the primary deformation zone of the chip. The size of the inclusion and the position are varied. It is confirmed that the size and the position of an inclusion have a great impact on the chip's formation. The results show the formation of a sharp notch in the chip when a large inclusion is positioned near the workpiece's surface.

Kurzfassung

FE - Simulation der Zerspanung inhomogen aufgebauter Materialien

by Stefan Distlberger

Der Spanbildungsprozess ist ein komplexer-, nicht vollständig verstandener Vorgang. Aspekte wie Reibung, die Bildung des Spans, Schnittkraft und Einschlüsse im Material definieren den Prozess. In dieser Arbeit werden einige dieser Aspekte mit Hilfe der kommerziellen FE-Softwarelösung ABAQUS^{TM}/ Standard analysiert und ein Überblick über gängige Methoden der Spanbildungssimulation gegeben. Im Zuge dessen werden Python- und Fortran-Skripts präsentiert, mit deren Hilfe man die Spanbildung von homogenen und inhomogenen Materialien in ABAQUS^{TM}/ Standard automatisieren und analysieren kann. Die dafür entwickelte Routine ermöglicht ein automatisches Neuvernetzen um kritischer Elementverzerrung vorzubeugen, sowie eine Automatisierung der ansonsten manuellen, zeitaufwändigen Modellierung.

Analyse- und Visualisierungsskripts ermöglichen es dem Benutzer beispielsweise Videos zu rendern oder die Spandickenänderung und Werkzeugkräfte als Funktion der Zeit zu beobachten. Des Weiteren wird der Einfluss unterschiedlicher Elementgrößen auf die Simulation untersucht. Es zeigt sich, dass sich bei kleiner werdenden Elementgrößen auch die Spandicke und Werkzeugkraft reduziert und der Span eine stärkere Krümmung aufweist. Eine Simulation mit homogenem Material, welches über die Schnittlänge an Spantiefe verliert, gibt Aufschluss über die Abnahme von Spandicke und Schnittkraft. Es werden zwei Modelle verglichen, wobei im ersten der Span ungehindert in Selbstkontakt treten kann, während im zweiten der Span getrimmt wird, um diesen Selbstkontakt gerade zu verhindern. Um Effekte von Einschlüssen beim Durchschreiten der primären Deformationszone zu zeigen, wird eine Parameterstudie mit drei unterschiedlichen Einschlussgrößen an unterschiedlichen Positionen durchgeführt. Die Studie zeigt, dass sowohl Größe und Position des Einschlusses einen Einfluss auf die Spanbildung haben. Beispielsweise bildet sich eine scharfe Kerbe, wenn ein großer Einschluss nahe der Werkstoffoberfläche positioniert wird.

Acknowledgements

I would like to express my gratitude to the persons who have supported me and contributed to this work. In particular, I owe a great deal of thanks to:

Prof. Dr. Thomas Antretter, the supervisor of this thesis, who always found optimistic ways to look at impossible problems.

Dr. Werner Ecker, my project advisor, who had the idea for this topic and guided me when I took a wrong turn.

Prof. Dr. Reinhold Ebner, for the opportunity to write my thesis at the Materials Center Leoben Forschung GmbH.

My office colleagues and friends, Daniel Kiener, Martin Schloffer, Martin Pletz, Markus Mikl-Resch, Thomas Wlanis, Christian Posch, Hans-Peter Krückl, Martin Krobath, Anatol Drlicek, Richard Tichy for their help, countless discussions and a lot of fun.

My best friend Harald Zlattinger, who always had an ear for my programming issues.

My family and friends for their support and friendship.

Financial support by the Austrian Federal Government (in particular from Bundesministerium für Verkehr, Innovation und Technologie and Bundesministerium für Wirtschaft, Familie und Jugend) represented by Österreichische Forschungsförderungsgesellschaft mbH and the Styrian and the Tyrolean Provincial Government, represented by Steirische Wirtschaftsförderungsgesellschaft mbH and Standortagentur Tirol, within the framework of the COMET Funding Programme is gratefully acknowledged.

Contents

List of Figures

List of Tables

Abbreviations

2D	**2 D**imensional
3D	**3 D**imensional
ALE	**A**rbitrary **L**agrangian **E**ulerian Method
AS	**A**utomator **S**cript
EP	**E**lastic **P**lastic
FE	**F**inite **E**lement
FEA	**F**inite **E**lement **A**nalysis
FEM	**F**inite **E**lement **M**ethod
HSC	**H**igh **S**peed **C**utting
JCM	**J**ohnson **C**ook **M**aterial Model
JCD	**J**ohnson **C**ook **D**amage Model
KS	**K**ernel **S**cript
PDZ	**P**rimary **D**eformation **Z**one
PEEQ	Accumulated equivalent plastic strain
PL	**P**ower **L**aw
UL	**U**pdated **L**agrangian
SDZ	**S**econdary **D**eformation **Z**one
SHPB	**S**plit **H**opkinson **P**ressure **B**ar test

Chapter 1

Introduction

1.1 Overview

The machining of materials is one of the most important types of manufacturing. Thus, a deep understanding is indispensable for controlling the process. In the last two decades, the research using FEM modeling techniques has increased rapidly [1]. Investigations of the effects on chip formation, cutting forces, temperature distribution and tool wear are constantly improving the efficiency and quality of cutting operations. This chapter presents a review of the current state of the art of FE cutting simulation. Critical issues and obstacles including constitutive laws, thermal interactions and different modeling approaches with their advantages and limitations are discussed. Turning, milling, planing and many more of the used cutting methods can be simulated with FE-programs like ABAQUS$^{\mathrm{TM}}$. Their commonality is a tool with a cutting edge, which is in contact with and penetrates the surface of a workpiece, resulting in a chip formation. Usually, the real contact is three dimensional, and therefore demands a solution in the same dimension. To simplify the contact problem, Astakhov and Outeiro [2] suggest to look at plane contact problems, where it is assumed that displacements are restricted to the x-y plane. Regarding Klocke and König [3], the machining process is categorized by the free orthogonal cut. As is in most of the 2D-simulations, the free, orthogonal cut is used to simplify the process, because only the main cutting edge is in contact. This geometry provides a good representation of the chip formation on the main cutting edge of many machining processes such as turning, milling, drilling, sawing and grinding, etc.

1.2 The cutting process

A cutting process is characterized by its unique features such as extensive deformation, high strain rates and high temperatures. During a cutting process, most of the chip's deformation occurs in the contact region between the tool's cutting edge and the chip as well as in a band, which forms from the tool's tip to the uncut surface of the workpiece. Those areas are called primary deformation zone (PDZ) and secondary deformation zone (SDZ). Astakhov and Outeiro [2] divide the secondary deformation zone into two distinctive parts of approximately equal length. A plastic zone, which reaches from the cutting edge to the middle of the contact length, and an elastic zone, which covers the rest of the contact length. Klocke and König [3] call these regions the *sticking region* and the *sliding region* (Figure 1.1).

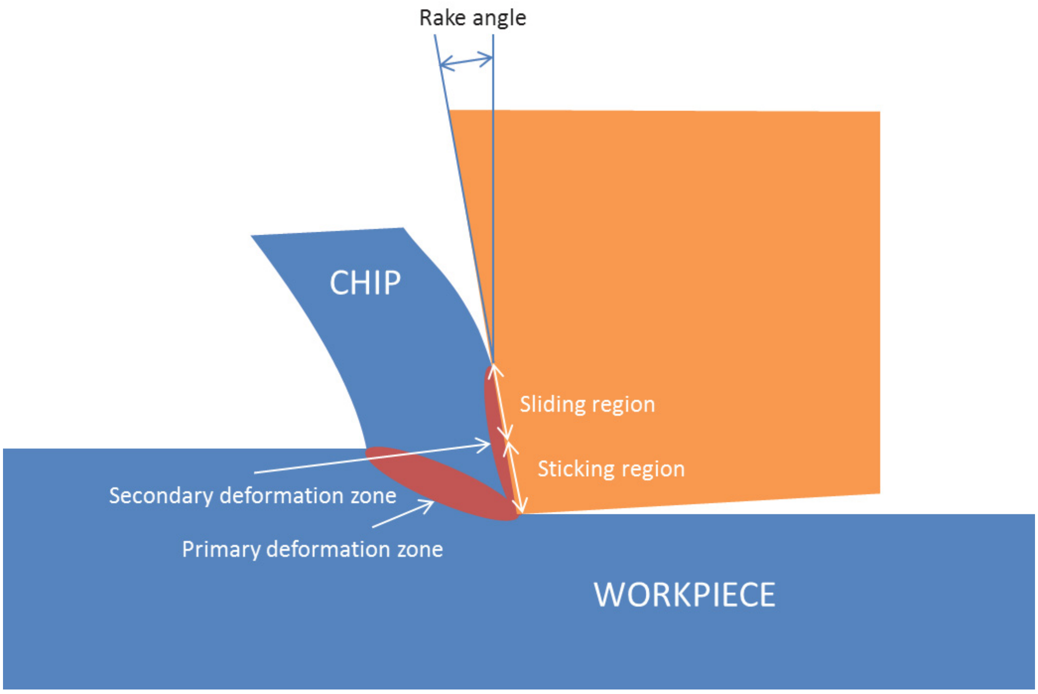


FIGURE 1.1: Regions on chip and tool.

The formation of the chip is highly depending on the material properties of the workpiece and can be separated into two different ways of cutting through a material. First, when cutting through a non ductile material, a forward propagating crack will form in front of the tool's tip. The upper flank of the crack then glides along the cutting edge and a chip forms. The amount of plastification due to the cutting process is material dependent. The second type of chip formation occurs, when a ductile material is cut. Instead of forming a crack path propagating forward, the material plastifies and flows around the tool's tip. According to Shi and Attia [1], the reliability of a finite element analysis (FEA) is mostly depending on:

1. Material: An accurate constitutive law within the simulation is necessary to describe the unique features mentioned above.
2. Process: The contact between the cutting edge and the chip has to be described by accurate friction and heat transfer models.

1.2.1 Homogeneous cutting

Depending on the tool's rake angle, the friction between the cutting surface and the velocity of the tool, the chip forms differently. In general, the chip forms gradually and the material starts to buckle. When the tool has a negative rake angle, the cut surface will experience compressive residual stresses. A positive rake angle like in Figure 1.1 produces tensile residual stresses on the cut surface. The severe deformations in the primary and secondary deformation zone cause an increase in temperature. Also the contact's friction between tool and workpiece introduces heat. The cutting force increases up to a point where a steady state is reached. This state is defined by a constant contact length between the tool and the workpiece, a stable shear band in the secondary deformation zone and a constant chip thickness. If the chip's temperature causes a rapid increase in the flow stress, the chip may show segmentation effects causing the appearance of a so called lamellar chip. Without the possibility of breaking, the chip continues to grow. This is called a flow chip. As the chip keeps growing, it bends and at some point the chip will get in contact with itself or the workpiece. This circumstance can cause an instability and furthermore it can cause the chip to break. Another way that causes a chip to break or affects its formation is the presence of inhomogeneities.

1.2.2 Inhomogeneous cutting

Inclusions such as carbides have a great impact on the chip formation. These inhomogeneities typically entail a sudden change of the mechanical properties. To account for this behavior in the simulations one has to look closely into the cutting process. Quan et al. [4] report on the consequences of different silicon carbides (SiC) in an aluminum matrix. SiC particles can only be deformed elastically or break. The split particles cause a rougher surface and an increase of the friction. Due to the higher friction at the interface, the tool flank's lifespan is reduced drastically. Therefore, the machinability of SiC reinforced materials is highly dependent on the particle size. Coarse particles cause a poor machinability, which leads to a higher tool-wear than in the case of smaller particles. When cutting a coarse particle composite, high plastic deformations of the matrix around the particles will occur, and hence the surrounding matrix becomes strongly hardened. Now the matrix combined with the particles show an even higher resistance and it becomes increasingly difficult for the tool to press the particles into the matrix. The consequence is a higher cutting force and the change of the shear angle. This may cause fracture of the particle across its crystal boundaries. In any case, the stress field will be non-uniform, and the thermal stress will increase caused by the friction between the workpiece and the tool. When the SiC breaks, the cutting force drops instantly and the SiC parts could be either pushed into the machined surface, or they could fall off the surface. When the cutting proceeds, the shear angle decreases until another particle appears. The remaining fragments are very hard and have abrasive effects on the tool's surface.

On the other hand, there are effects caused by small particles. When the tool cuts a small particle reinforced material, the indentation of the small particles consumes less energy. The particles will either flow along with the chip or along the interface. When particles reach the surface of a weak matrix material, they may fall out. Quan et al. [4] wrote that an increase in the fraction of reinforcement particles leads to a decrease in the deformation coefficient, and an increase in the shear angle.

1.3 Finite Element Analysis

There are numerous defining factors when building a FE model, such as material behavior or contact properties. Shi and Attia [1] state that only a few accurate predictions regarding the cutting force or chip morphology can be found in literature. This is referred to as the unique features observed in the cutting process, including the complex occurrence of extreme deformations and localized temperatures in the primary and secondary deformation zones.

For a successful simulation, one has to know all the influences caused by these features and use them properly. Some of the methods found in the literature to define those features are discussed in this chapter.

1.3.1 Constitutive models

As mentioned in chapter 1.2, the constitutive model has a big impact on the quality of the FE results. When modeling the cutting-process a constitutive model should represent the material's behavior for large strains, high strain rates and high temperatures.

Constitutive models found in the literature have mostly been developed based on experimental data to describe the material behavior inside the simulation. These data are mostly generated in a direct way using the Split Hopkinson Pressure Bar test (SHPB) [3], or by an inverse modeling from instrumented orthogonal milling tests [5]. SHPB offers a range of data necessary to formulate a constitutive model, including the flow stress $\bar{\sigma}$, the effective strain $\bar{\epsilon}$, the effective strain rate $\dot{\bar{\epsilon}}$ and the temperature T during the plastic deformation.

Shi and Attia [1] summarize many of the used constitutive models. These models describe the relationship between stress and strain with respect in temperature and strain rate. They claim that all of these models are accurate when used with slow strain rates and show a good agreement with the experimental data. Unfortunately, this no longer applies for high strain rates or at high temperatures. This is caused by the limitations of SHPB, which allows strains of up to 1.0 and strain rates of less than 10^4 s^{-1}, only.

The formulation used in most of the surveyed articles such as [6–8] has been developed by Johnson and Cook in 1983 [9, 10]. Their constitutive model includes isotropic strain hardening, strain-rate hardening and thermal softening, where the yield stress $\bar{\sigma}$ is expressed in equation 1.1. The model was evaluated by comparing it to different experimental data, such as a cylinder impact at high velocity, since the cutting process itself is performed at high cutting speeds. Very similar to the experimental impact conditions at about 130 ms^{-1} to 343 ms^{-1}, the Johnson Cook Material (JCM) formulation is suitable also for cutting simulations.

$$\bar{\sigma} = \left[A + B\left(\bar{\epsilon}^{pl}\right)^n\right]\left[1 + C\ln\left(\frac{\dot{\bar{\epsilon}}^{pl}}{\dot{\epsilon}_0}\right)\right]\left(1 - \hat{\theta}^m\right) \tag{1.1}$$

$$\hat{\theta} = \left(\frac{T - T_0}{T_m - T_0}\right) \tag{1.2}$$

The coefficient A is the yield strength (MPa), B is the hardening modulus (MPa), C is the strain rate sensitivity coefficient, n is the hardening exponent and m is the thermal softening exponent. $\hat{\theta}$ is a non-dimensional factor, which is formed by using the material's current temperature T, the room temperature T_0 and the melting temperature T_m (eq. 1.2).

Similar to the JCM the constitutive law can be described by a power law in the form 1.3, but uses the initial flow stress σ_0 and the thermal softening coefficient ν. Further information can be found in [1, 11].

$$\bar{\sigma} = \sigma_0\bar{\epsilon}^n\left(\frac{\dot{\bar{\epsilon}}}{\dot{\epsilon}_0}\right)^m\left(\frac{T}{T_0}\right)^{-\nu} \tag{1.3}$$

In the Vinh model 1.4, the thermal softening is written in an exponential form.

$$\bar{\sigma} = \sigma_0\bar{\epsilon}^n\left(\frac{\dot{\bar{\epsilon}}}{\dot{\epsilon}_0}\right)^m \exp\left(\frac{G}{T}\right) \tag{1.4}$$

Shi and Attia [1] evaluate commonly used constitutive models they found in the open literature and their behavior in comparison to the experimental discoveries of machining procedures. They report that some of these models cannot correctly describe the experimental behavior of flow stress variations in the primary deformation zone. They therefore developed a new model which is a combination of the JCM, PL and Vinh model, named V-JC-PL model (eq. 1.5).

$$\bar{\sigma} = [a - b\exp\left(-c\,\bar{\epsilon}\right)]\left[1 + c\,\ln\left(\frac{\dot{\bar{\epsilon}}}{\dot{\epsilon}_0}\right)\right]\left(\frac{T}{T_0}\right)^{-\nu} \tag{1.5}$$

Example values for a high-strength, corrosion-resistant nickel chromium material are: $\dot{\bar{\epsilon}} = 10^{-3}\ \mathrm{s}^{-1}$, $T_0 = 293$ K and the material constants $a = 1233$ MPa, $b = 4455$ MPa, $c = 31$, $C = 0.023$ and $\nu = 0.21$ [1]. In their tests with a SHPB they achieved a much better predictability than only by using each single model individually. The correct selection of the constitutive law and its parameters is a critical step. It will define the chip's formation and morphology, the accuracy of the cutting force, the feed force, temperature and tool wear (residual stress, surface roughness, micro-structure, etc.) [5].

Another formulation of the Johnson Cook model can be used to introduce damage initiation in materials. Different modeling techniques like the use of a predefined crack path (see chapter 1.3.4) or the simulation of chip segmentation depend on a damage model like the Johnson Cook Damage model (JCD) [12, 13]. ABAQUS$^{\mathrm{TM}}$/Explicit provides a special case of this criterion, which differs from the original formula published by Johnson and Cook [10]. The damage initiation is triggered by the equivalent plastic damage strain $\bar{\epsilon}_D^{pl}$ as expressed in equation 1.6.

$$\bar{\epsilon}_D^{pl} = [d_1 + d_2\exp\left(-d_3\eta\right)]\left[1 + d_4\ln\left(\frac{\dot{\bar{\epsilon}}}{\dot{\epsilon}_0}\right)\right]\left(1 + d_5\hat{\theta}\right) \tag{1.6}$$

$\hat{\theta}$ is the same non-dimensional factor as in equation 1.2, $d_1 - d_5$ are failure parameters and $\eta = -\frac{p}{q}$ is the stress triaxiality, where p is the pressure stress and q is the Mises equivalent stress [9].

For example, Ng and Aspinwall [14] presented an ABAQUS$^{\mathrm{TM}}$/Explicit model to simulate continued and segmental chip formation by introducing an element deletion technique with JCD.

1.3.2 Contact of the tool- chip interface, friction and heat

The most important contact occurs between the main cutting edge and the chip. The complexity of the contact situation makes it hard to obtain accurate results. The meshed surface of the workpiece is in contact with the tool's mesh. The convergence of the simulation is highly depending on the quality of the mesh, especially in the contact zone. For example, Bäker et al. [15] simulated the effect of chip segmentation. They used ABAQUS^{TM}/Standard with a self-developed remeshing algorithm, where their mesh changes periodically (more about remeshing see chapter 1.3.4). It has to be noted that the remeshing disturbs the force equilibrium which must be restored at the beginning of the new increment. This may cause initial deformations that prevent the restart of the simulation. Therefore, they introduced so called convergence controls which they adjusted to typical forces found in the shear zone, and used artificial damping to keep the initial deformations small and achieve convergence. They ensured that the amount of energy caused by the artificial damping is less than 0.1% of the total work, and can therefore be neglected.

Along the contact surface shear forces and normal forces are transmitted as stresses. The friction model and its coefficient are responsible for the relationship between these shear and normal stresses. ABAQUS^{TM}offers many different models including the Coulomb friction model, which allows to define the friction coefficient in terms of slip rate, contact pressure, average surface temperature and provides options to define static and kinetic friction coefficients with a smooth transition zone.

Some of the articles use the Coloumb friction law with a friction coefficient μ of 0 to 0.5 [1, 2, 11, 16–18]. Tests from Astakhov and Outeiro [2] show that a friction coefficient up to $\mu = 2$ can be found on the tool tip. They state that the normal stress at the tool-chip interface is zero at the point where the chip separates from the tool, and has its maximum at the cutting-edge.
The shear stress distribution can be described with the Coulomb friction μ.

P is the normal force and F is the friction force. When the stress normal to the interface $\sigma = \frac{P}{A}$ and the shear stress $\tau = \frac{F}{A}$ then the friction coefficient is defined as:

$$\mu = \frac{F}{P} = \frac{\tau}{\sigma} \tag{1.7}$$

The Coulomb friction model allows to define a shear stress limit $\bar{\tau}_{max}$, to enable sliding in the contact zone regardless of the contact pressure stress.

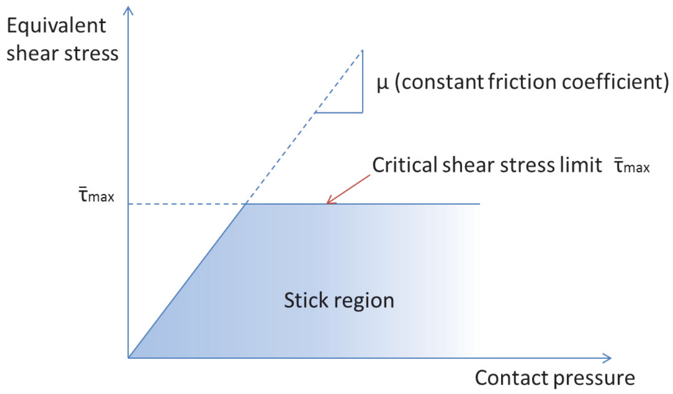


FIGURE 1.2: Coulomb friction model with shear stress limit.

Astakhov and Outeiro [2] state that there is no relative motion in most of the contact area. Thus, they calculate a minimum friction coefficient of 0.577 for sticking conditions. Therefore, when the friction model does not distinguish between the sticking and sliding area and the sticking area covers most of the contact area, a friction coefficient greater than 0.577 should be used for the simulation. Umbrello et al. [5] evaluate their friction model by comparing the simulated cutting force with the experimental cutting force. They adjust the friction coefficient until both forces match to examine the friction conditions.

In simulations where heat is produced due to mechanical deformation, an adiabatic thermal-stress analysis is favorable, where heat exchange can only occur inside the system. This is only possible when the deformation of the material is faster than the time the heat would need to diffuse through the material. The cutting process meets those conditions due to its high strain rates, where heat is produced by the material's deformation. In ABAQUS^{TM}a fully coupled thermal-stress analysis must be used to simulate the cutting process since the stress solution is depending on the temperature and vice versa. Due to the inelastic heat fraction the deformation of the material produces a heat flux adding to the thermal energy balance. By default, the amount of inelastic heat is 90% of the product between stress and rate of plastic straining. Additionally, the contact conditions may include properties for heat generation due to friction as well as thermal conductance.

1.3.3 Eulerian and Lagrangian view of the continuum

When building a 2D or 3D model for a FEA the workpiece is a discretized continuum. Klocke and König [3] explain two possible ways to discretize a continuum. These two formulations, the Lagrangian formulation and the Eulerian formulation are derived from the basic principle of virtual work and are valid for linear and non-linear material behavior.

The Lagrangian formulation describes the movement of an element's node with the material, see Fig. 1.3. It can be split into two different formulations. The formulation describing the system's current configuration in relation to its initial configuration is called "Total Lagrange formulation" (TL). That means, that the nodal-coordinates always refer to their initial position. This formulation can be used to analyze linear or weakly non-linear material behavior. A more suitable way to deal with large strains and non-linear material behavior occuring in the simulation of a chip formation is the "Updated Lagrange formulation" (UL), where the current configuration of the system is described relative to the foregoing configuration step. In contrast to the TL the UL is an approximation procedure whose predictive quality starts to degrade when the step size increases.

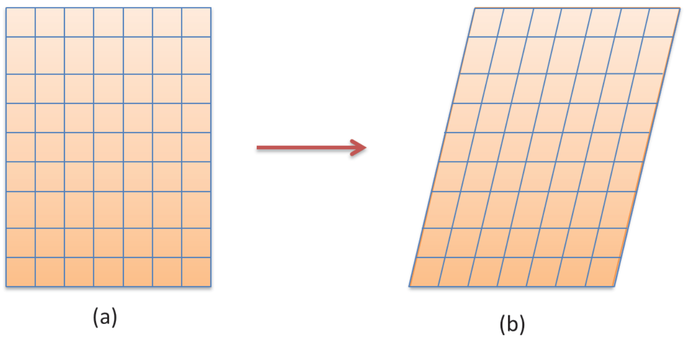


FIGURE 1.3: Lagrangian formulation. The nodes move with the material when the geometry changes form (a) to (b).

Conversely, the Eulerian formulation describes the movement of the continuum through a stationary mesh, see Fig. 1.4.

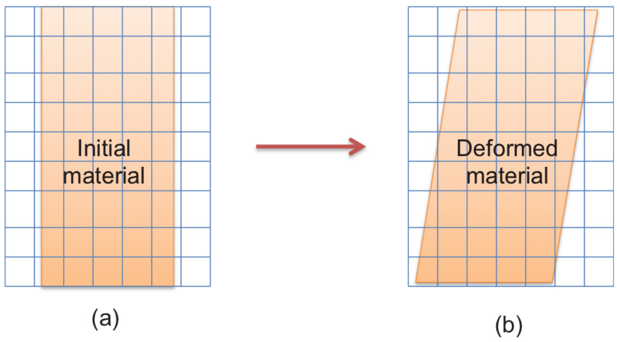


FIGURE 1.4: Eulerian formulation. (a) The material is covered by a static mesh (b) the material deforms while the mesh remains constant.

1.3.4 Methods to avoid excessive mesh distortion

As mentioned in chapter 1.2 the chip can form with a forward slipping crack path or it flows around the tool's tip. These two effects are also used as modeling method to avoid excessive mesh distortion.

By using a predefined crack path along the cutting layer, the tool splits the material into two separate parts. The upper part can move along the tool's surface without experiencing excessive mesh distortions. This method can be used for ductile and non-ductile materials. For example, Kalhori [16] builds an UL model in ABAQUS$^{\mathrm{TM}}$/Standard, which he compares with experimental results using a pre-defined crack path (see Fig. 1.5).

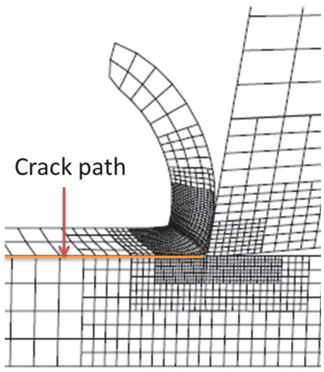


FIGURE 1.5: Predefined crack path.

The remeshing method (see Fig. 1.6) represents another viable approach using the UL formulation with the material flowing around the tool's tip. Here, the mesh often happens to become extensively distorted. There are two possible ways to generate a new mesh. A manually generated mesh can perfectly be adjusted to its purpose, which can help to avoid mesh related convergence problems. On the other hand, an automated approach is helpful when a new mesh has to be generated frequently.

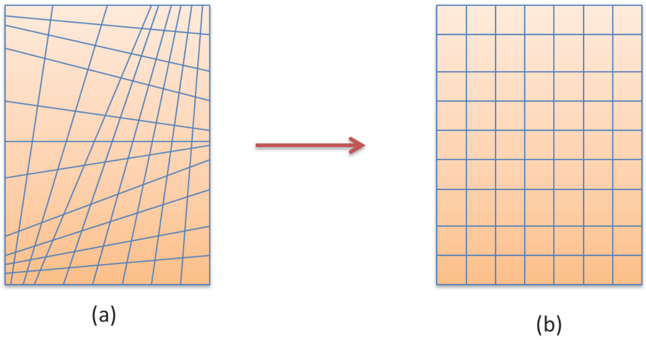


FIGURE 1.6: (a) before the geometry gets remeshed. (b) after remeshing.

Klocke and König [3] state that if huge plastic deformations occur, the distorted mesh could lead to numerial problems and instability. To allow large degrees deformation of the material, they use the remeshing method to obtain a new undistorted mesh. After the remeshing process, the simulated results of the old mesh have to be interpolated onto the new mesh. The interpolation can cause a loss of accuracy when it is performed frequently, or when the mesh-size difference is too large. Furthermore, an excessive distortion of the primary mesh can cause numerical artefacts, or even a smoothing effect of the interpolated data when performed repeatedly. For example, Bäker et al. [15] used a transfinite interpolation scheme, combined with an inverted Laplacian equation and a full grid algorithm, to reduce the residual of the solution to a value less than the discretisation error to remesh their model. After the new mesh is successfully generated, they use a technique called MAP SOLUTION, which is integrated in ABAQUS^{TM}/Standard, to map the old data onto the new mesh (more information about MAP SOLUTION in chapter 2.2.4). The influence of the used interpolation method in ABAQUS^{TM}/Standard is evaluated in this thesis.

The most commonly used method found in the literature combines the UL and the Eulerian formulation forming the Arbitrary Lagrangian Eulerian formulation (ALE). With an ALE formulation, the nodes move with the material in the same way as they would in an UL formulation, but can at the same time adjust their position as if they were not connected to the material. This enables the chip's mesh to be updated freely without distortion and without the need of any predefined crack as a-priori existing separation line. To this end the adaptive mesh parameters have to be adjusted judiciously. ALE is very suitable for the cutting conditions in which a rounded or chamfered-edge cutting tool is used. With ALE, coarser elements can still produce an acceptable chip thickness and cutting forces, etc. because no failure criterion is required. In addition a broader selection of material models is available [19]. The adjustment of the nodes can be done before distortion effects can occur (see Figure 1.7).

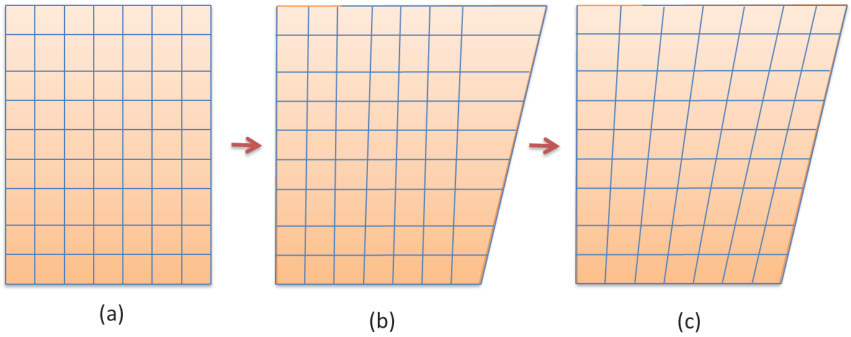


FIGURE 1.7: (b) The material gets deformed, resulting in a distortion of the mesh. (c) Repositioning of the nodes avoids distortion.

Guo and Yen [20] show a model using the ALE (adaptive meshing) method and the JCD model to study the discontinuous chip formation in hard machining. When elements fail due to the damage criterion, the nodes along the interface between the failed and the remaining elements will become non-adaptive since the failed elements have already been deleted. Thus they periodically use an adaptive meshing to control the mesh quality in order to maintain computational efficiency. In the same year, Gadala [21] presented a fully coupled ALE formulation for large deformation in static and dynamic problems. Their detailed formulation and possible ways to modify ALE show its advantage for dynamic metal forming, crack propagation and orthogonal metal cutting simulations.

Concluding, there is no general opinion on the advantages or disadvantages of UL vs. ALE. Taking into consideration the UL with a JCD model the expected results can be very similar to the experimental results of Vaziri et al. [22]. Moreover, this resolves the problem of a highly distorted mesh and related software crashes by deleting elements above a given limit thereby eliminating the numerical problems related to an adaptive meshing procedure. On the other hand, ALE in ABAQUS^{TM}/Explicit offers a fast and safe procedure to reliably obtain results. However the magnitudes of temperature, strain rate or material flow stress may be higher than those of an UL approach, also compared to the experimental data [23]. Nevertheless, using ALE in ABAQUS^{TM}/Explicit is preferred in most of the reviewed articles.

1.4 Common issues reported in the literature

Most of the articles state that the simulated cutting force does not match the experimental data.
Ducobu et al. [24] recognize certain variations of the cutting forces, while the feed forces almost do not show any variations. Taking into consideration differences in the metallurgical state and after refining their constitutive law, the quality of their results improved in that the cutting force became similar to the measured cutting force. However, then the feed force was slightly overestimated. Mohammed et al. [25] also showed variations in the cutting forces. Especially at low cutting speeds, the simulated cutting force was more than 30% smaller than the experimental cutting force. Also Zanger and Schulze [18] report differences between the simulated and experimental forces.
In 2004, Soo et al. [26] presented a 3D ABAQUS^{TM}Explicit model using a Lagrangian approach. Their results show almost no differences when they compare the predicted tangential force and feed force with the measured forces. Soehner and Joerg [27] present a study which compares the results of Kalhori [16] simulation with ABAQUS^{TM}/Standard, and the results from ABAQUS^{TM}/Explicit. They conclude that the cutting forces, pressure and temperature simulated by the finite element software DEFORM^{TM}-2D, ABAQUS^{TM}/Standard and ABAQUS^{TM}/Explicit show only insignificant differences.

In this work the software ABAQUS^{TM}/Standard with an UL is chosen for the following reasons:

- Nodal data can be accessed with the user subroutine URDFIL, that is limited to ABAQUS^{TM}/Standard.
- To simulate large time increments that would be unstable in ABAQUS^{TM}/Explicit.
- To implement a self developed automation procedure using the MAP SOLUTION technique that is limited to ABAQUS^{TM}/Standard.
- To be able to use fully integrated elements. Generally, only first order elements with reduced integration can be used in ABAQUS^{TM}/Explicit.

Chapter 2

Methods

When analyzing a cutting process with UL with periodic remeshing, one has to follow certain steps. Fig. 2.1 shows the simplified flow diagram of these steps. First an initial model has to be created. It contains all information such as geometry, material and step definition. After the model has been set up the user has to start the analysis job. The resulting data is stored inside the output database (*.odb). Then, the deformed parts can be imported into ABAQUS^{TM}/Standard and all model parameters have to be reapplied onto the new parts. After rebuilding the new analysis job has to be started but this time the field output data from the old output database is mapped onto the new parts. After successful completion of one such cycle, the procedure starts over again.

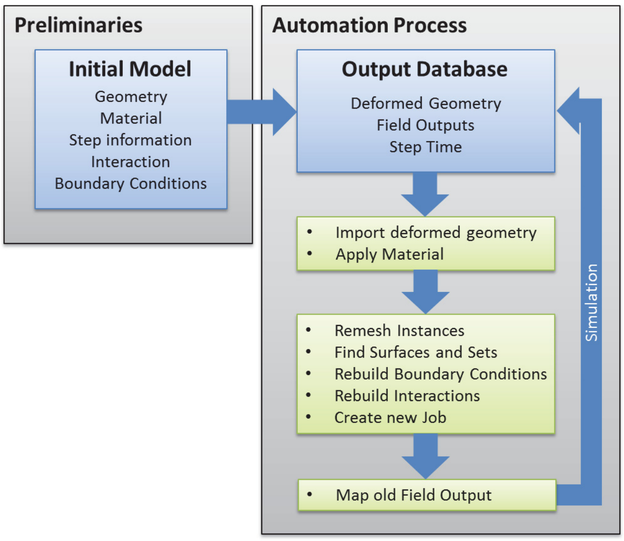


FIGURE 2.1: Analysis routine.

A chip simulation can consist of thousands of output databases and a manual approach would be inefficient. Therefore an automatic procedure has been developed to automate the steps done manually before. Fig. 2.2 shows a more detailed road map through the chip formation procedure, valid for the manual and automated approach.

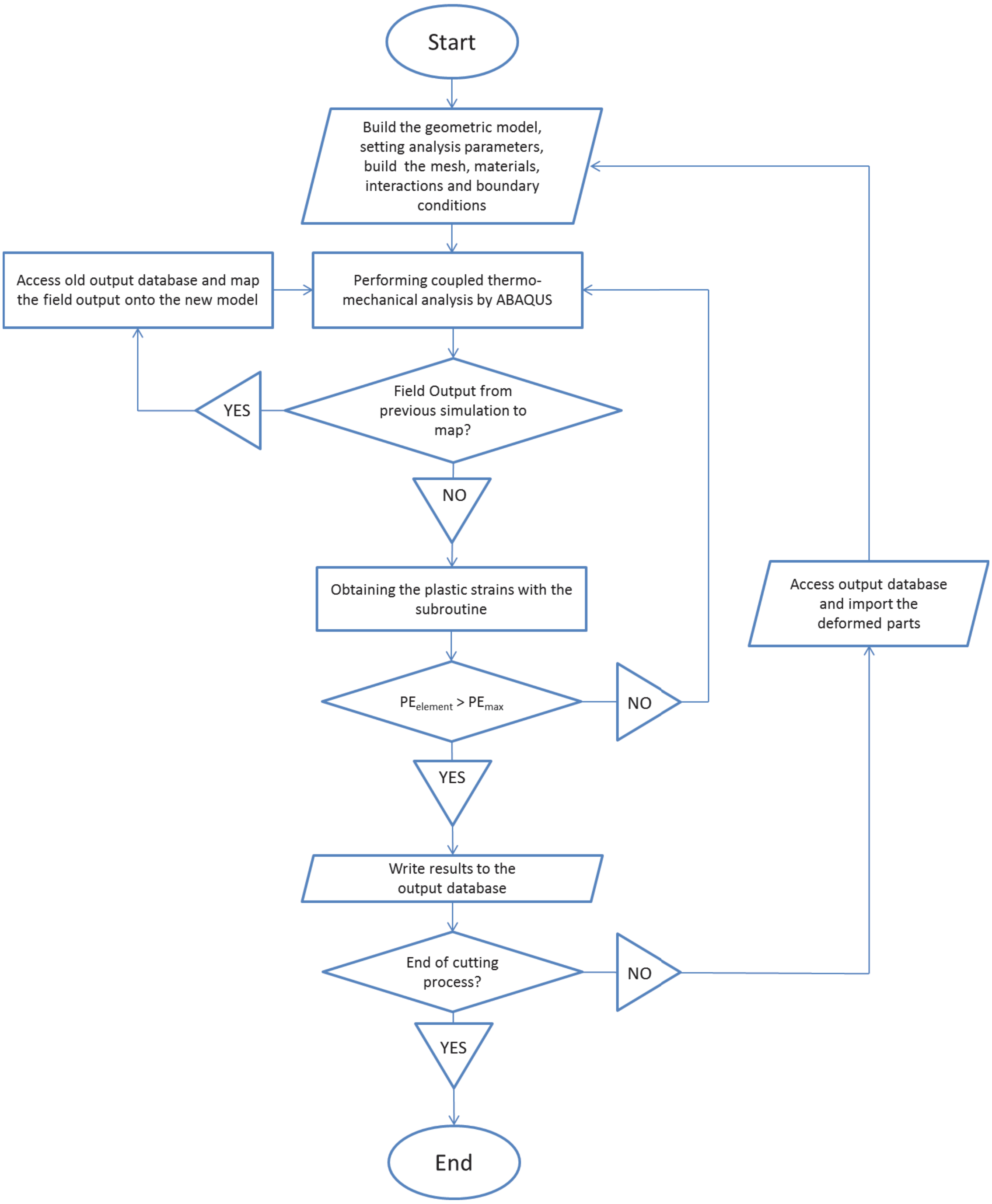


FIGURE 2.2: Detailed cutting process using UL and a remeshing routine.

2.1 The finite element model

By definition, the cutting process involves large geometry-changes, which requires the use of UL or ALE. The need of a wide variety of contact conditions as well as different material definitions suggest to use an implicit approach, with UL and a remeshing procedure. Nevertheless, many of the cutting simulations found in the literature are using ABAQUS^{TM}/Explicit, mostly for convergence reasons (see for example [22]).
The implicit code checks for convergence during the iterative search for a solution. This may not always be successful. However, one of the advantages of an implicit approach is the availability of various user-subroutines which cannot be accessed by ABAQUS^{TM}/Explicit. Furthermore, the chip formation exhibits localized deformations, which demand a dense mesh to ensure good results. The computational time of ABAQUS^{TM}/Explicit increases strongly with a decrease of the mesh size. Moreover, often numerical tricks like density changes or artificial viscosity are necessary to improve performance. Therefore, the implicit approach is preferred as long as convergence can be reached. The simulations in this thesis were carried out using a fully coupled thermo-mechanical finite element model in order to consider the influence of temperature related effects inside the chip.

2.1.1 The model

The model consists of two parts. The workpiece has an overall length of 52.5mm and a height of 1mm (see Figure 2.3). Since this model is intended to describe the milling process, the uncut chip thickness decreases from 0.4mm to zero, which simulates the cutting edge entering the full material and the decreasing uncut chip thickness. The uncut chip thickness is the difference between the expected cutting path and the workpiece's surface. The cutting edge is split into two different rake angles. The cutting edge near the tip has a rake angle of 10° and the tool itself is rotated by 7°, causing a negative rake angle (see Figure 2.4). At the beginning, the cutting tool's edge-geometry is already cut out of the initial workpiece in order to prevent contact issues at the beginning of the simulation. (see Figure 2.5).

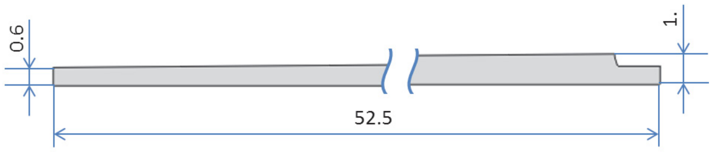
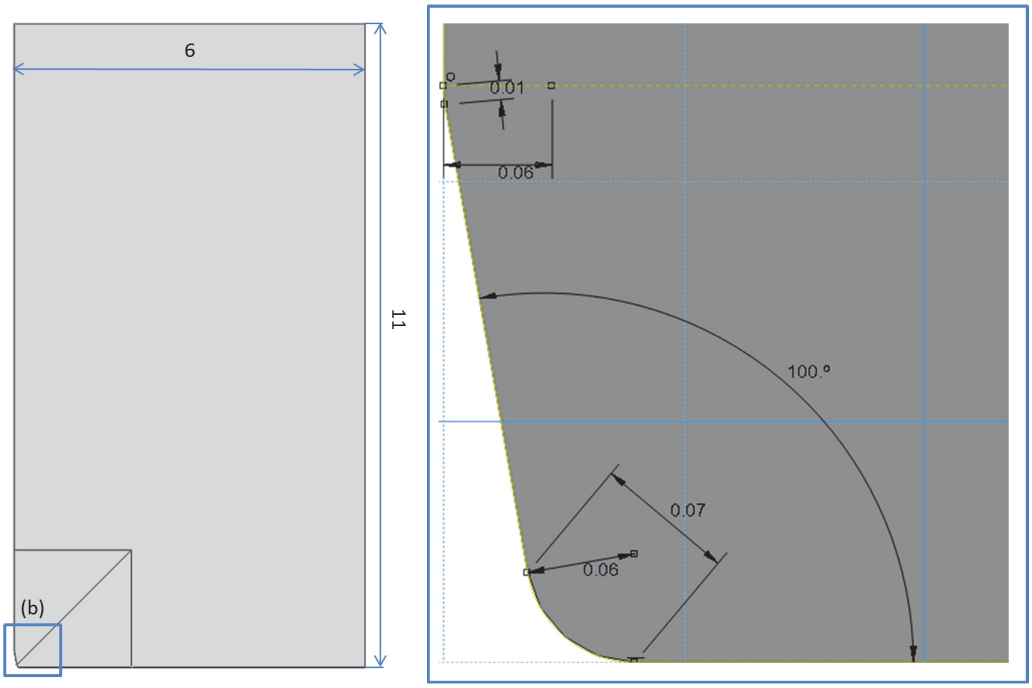


FIGURE 2.3: Geometry of the workpiece. (dimensions in mm)



(a) (b)

FIGURE 2.4: a) Geometry of the tool. b) Tip of the tool's cutting edge.

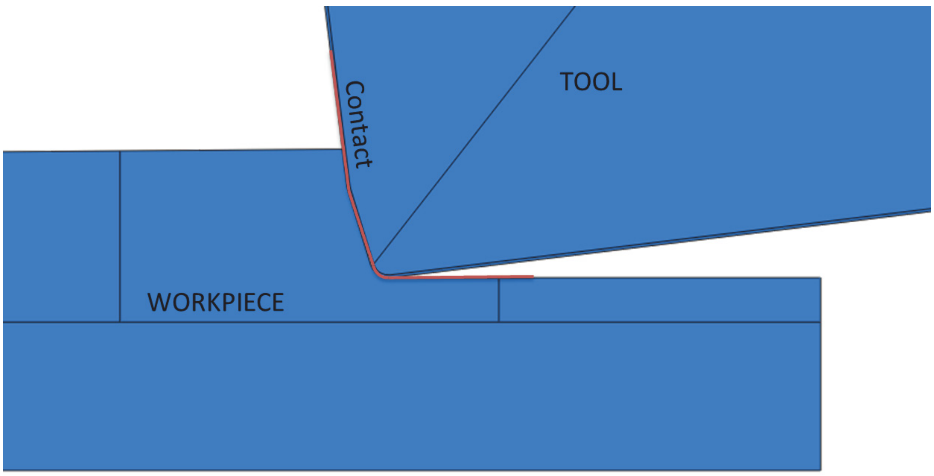


FIGURE 2.5: The cutting edge is in contact with the workpiece.

Depending on the chip's formation, the chip may get in contact with the workpiece surface, causing the simulation to abort. This can be explained by the way the preprocessor stores geometry information: When the orphan mesh which is the deformed mesh of the previous simulation is converted into a geometrical entity, the spline approximating the chip's tip outline overlaps with the uncut workpiece surface. The evolving intersection interferes with the generation of the part and the reassembling encounters an error.

At this point, two possible scenarios can be simulated. When the chip is expected to get in contact with itself, a rigid body is positioned right above the upper edge of the workpiece preventing the chip geometry to penetrate into the workpiece (see Figure 2.6 (b)). If the chip is not supposed to establish contact, the tip of the chip must be removed in periodic intervals (see Figure 2.6 (a)).

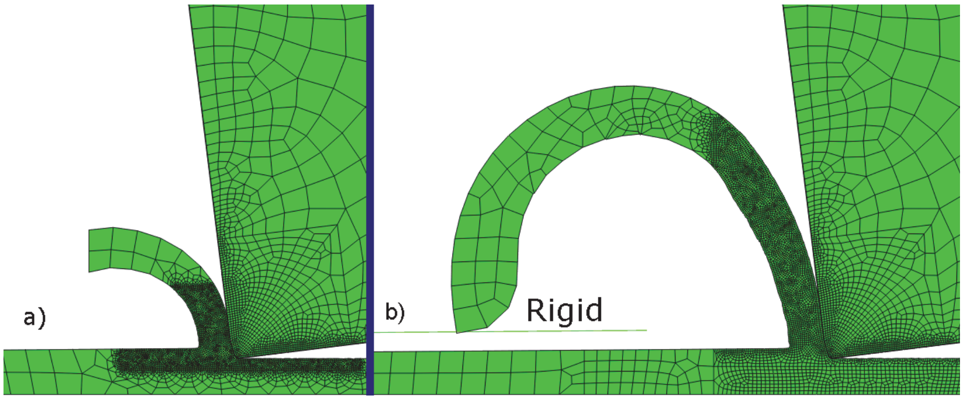


FIGURE 2.6: a) The chip gets cut each cycle or b), the chip is in contact with the rigid.

When simulating an inhomogeneous model the uncut chip thickness does not decrease along the workpiece's surface. This is essential for reaching the steady state condition, where contact force and chip thickness remain constant on the one hand and to investigate the influences of the inhomogeneities on the other hand (see Figure 2.7).

When the model reaches steady state, inclusions are positioned at three different positions inside the workpiece near the primary deformation zone. Depending on the material of the inclusion, the interaction properties are different. If for example the inclusion is the weaker phase, it will experience heavy deformation during the simulation. Therefore, the inclusion must also be remeshed where the matrix is defined as the master surface for the interaction definition. If the inclusion is harder than the matrix material providing the orphan mesh of the inclusion would be sufficient. Then the inclusion's surface is chosen as the master surface instead. The interaction formulation between matrix and inclusion is very important. When the interface between matrix and inclusion is allowed to separate, the matrix shows decohesion effects on the upper and lower pole of the inclusion that may even form a crack whose flanks may for extreme cases overlap during the remeshing. Every time this happens, the user has to rebuild the model manually. Therefore a perfectly bonded inclusion-matrix interface i.e. a tied contact formulation is preferred for an example study on the chip's formation in relation to the inclusion size and position (see chapter 3.3).

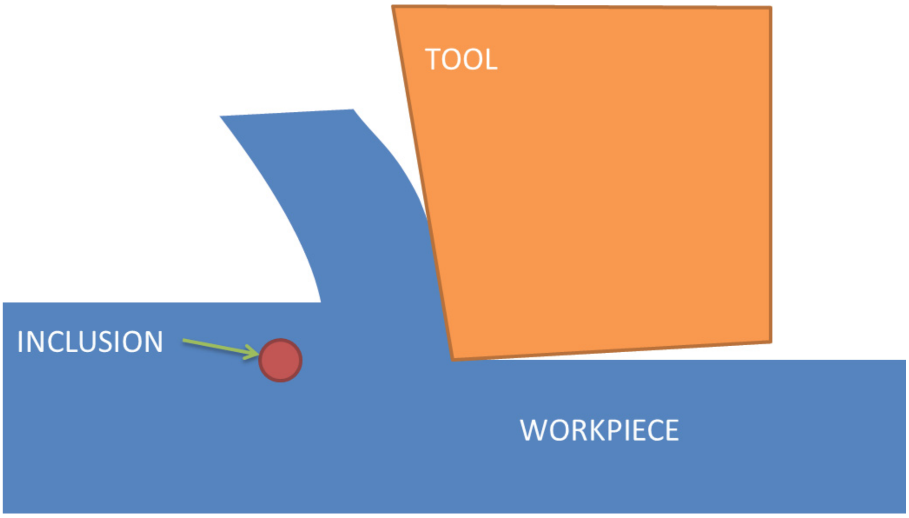


Figure 2.7: Sketch of the inhomogeneous model.

2.1.2 Material data

The following table lists the set of material parameters for the workpiece needed for a fully coupled thermo-mechanical simulation (see table 2.1).

Workpiece		
Young's modulus	210000.	MPa
Poisson's ratio	0.3	
Density	7.8x10^{-9}	kg/m^3
Thermal expansion coefficient	1.2x10^{-5}	K^{-1}
Inelastic heat fraction	0.9	
Conductivity	46. at 20°C	W/mK
Johnson Cook parameters		
A	400.	MPa
B	340.	MPa
m	0.15	
n	1.	
T_m	1520.	°C
T_0	20.	°C
C	0.01	
$\dot{\epsilon}_0$	0.001	

Table 2.1: Material properties and JCM parameters

The homogeneous models and the inhomogeneous models were simulated both with an elastic-plastic tool as well as a rigid tool. The material data of the inclusions and the elastic-plastic tool are listed in appendix C.

2.1.3 Interaction properties used in the simulations

The interaction between the chip and the cutting edge and its properties are essential for a successful simulation. To simulate the mechanical and thermal interactions between the workpiece and the tool, a surface-to-surface contact with a slave adjustment tolerance of 0.001mm is used. To implement inclusions a surface-to-surface contact is used as well, although it is defined as a tied contact, where the surface of the inclusion and the surface of the workpiece or chip, respectively, remain glued together during the entire simulation.

A penalty formulation with a friction coefficient of 0.6 is used to impose a frictional constraint on the tangential behavior. Relative motion of the surfaces will generally occur. When the surfaces are sticking (i.e. $\bar{\tau} < \bar{\tau}_{crit}$) [9], the magnitude of sliding is limited to the elastic slip value which is set to 0.005 in this model. The equivalent shear stress limit $\bar{\tau}_{max}$ is set to 3.6x10^8 MPa in order to allow sliding between the surfaces when the limit is reached.

By default the friction power is completely converted into heat. To establish heat transfer between the elastic-plastic tool and the workpiece an empiric conductance is set to a value of 10^9 W/mK at zero clearance. The conductance decays to zero at a maximum clearance of 0.1 mm.

The simulation consists of two steps. The first step is introduced in order to reestablish the force equilibrium that may have been lost due to the *Map Solution Mapping*-function (see chapter 2.2.4). In this step, the displacement of the workpiece is disabled, and additionally contact controls with a stabilization factor of 10 are introduced to stabilize the contact over the first step. The use of a stabilization factor greater than 10 sometimes leads to convergence problems. The second step where cutting occurs is similar to the first one. Here, the workpiece moves towards the tool with a displacement boundary condition. To preclude any interference of the manual *contact controls* settings with the automatic contact stabilization, the second step is always simulated without contact stabilization.

2.2 Numerical Methods

2.2.1 Common 2D remeshing tool

The use of an algorithm to remesh the deformed model automatically is essential when simulating a cutting process with UL, as element distortions become excessively large. To trigger the remeshing routine, the value of the relative plastic equivalent strain accumulated in the actual simulation has to exceed a user defined critical value. This value, for cutting mostly around 0.7, is stored inside the user-subroutine UVARM. Tests have shown that this value can be of great importance when convergence problems occur. If the value is chosen too high, the simulation may still work but may also entail severely distorted elements. Even though the mesh is renewed in the subsequent simulation the data from the heavily distorted old mesh could cause erroneous stress fields which would then be mapped onto the new mesh making convergence unlikely. If the value is set too low, the number of simulations to reach a certain displacement and hence the calculation time will increase.

The initial CAE File: To use the *common 2D remeshing tool*, an initial CAE-file has to be generated. This file has to contain all the parameters and settings that are needed to run the first simulation manually. Later on, the *common 2D remeshing tool* will always rely on these initial CAE-file parameters. This also allows to start simulations at any point of a valid output database and even to change the model or its parameters. For instance, one could use a coarse mesh to reach a certain state with less calculation time and change it later to a much finer mesh in order to study the process in more detail. Even inclusions such as particles can be added after the chip has reached its steady state to reduce calculation time and to use the chip in its steady state for a parameter study.

Elements: In the used simulations, the favored element type is a 4-node plane strain thermally coupled quadrilateral element with bilinear shape functions in ABAQUS^{TM} denoted as CPE4T. The meshing algorithm needs also an exact specification of the selected triangular element as input. Reduced integrated elements show hourglassing effects at the tool edge, which cannot be completely prevented by introducing hourglass controls. As large plastic strains are to be expected during the simulation, the use of fully integrated elements is suggested to ensure high accuracy. Inside the script *Automator.py* the element type of remeshable parts can be set manually. For the simulations used in this thesis, the following commands were used:

```
elemType1 = mesh.ElemType(elemCode=CPE4T, elemLibrary=STANDARD)
elemType2 = mesh.ElemType(elemCode=CPE3T, elemLibrary=STANDARD)
A.setElementType(regions=(A.instances[PartList[1].cleanName+
    '-'+repr(NextCaeCycle)].faces,),
    elemTypes=(elemType1, elemType2))
```

2.2.1.1 Python scripts

Basically, the *common 2D remeshing tool* comprises two python scripts. The "kernel-script" (KS) contains all functions to process the remeshing procedure. The "automator-script" (AS) contains a sequence of KS-functions and all necessary utilities which can be adapted by the user. When the AS is executed, the initial model is analyzed by the function *startAutomaticProcedure*, which executes the analysis procedure *analyse*.

Analyse: *analyse* is the first function called by the AS. Inside *analyse*, all instances of the initial model are analyzed by the function *analyseInstances*. Their names and attributes get saved in a list named *PartList*. Then, the *analyse*-function executes the functions *analyseSurfaces*, *analyseAnalysisType*, *analyseInteractions*, *analyseBC* and *nodeSetToPartList*. The obtained information is stored inside the *PartList*-list. After a successful analysis the AS continues.

Mesh Windows Class: Mesh windows can be used to define sections with different mesh sizes. One can create as many mesh-windows as needed on a remeshable part. An example how to define a mesh window can be found in the AS in appendix B.

```
mw=MeshWindow('mw1',PartList[1].cleanName+'-'+repr(NextCaeCycle),0.4)
```

This command defines a new mesh-window object, where *'mw1'* is the name of the mesh-window-object, *PartList[1].cleanName+'-'+repr(NextCaeCycle)* is the name of the chosen instance and 0.4 is the global mesh-size of the instance. To find the part's number inside the *PartList*-list, one can type the following command into the CLI:

```
>>> PartList[0].name
"WORKPIECE-1"
>>> PartList[1].name
"TOOL-1"
```

Now, windows can be defined by two different ways. The static way to define a new mesh window is for example:

```
mw.window(1.2,10,3.5,-3.9,0.03)
```

The first four parameters define the two coordinates of a rectangle. The upper left edge is defined by the first two parameters and the lower right edge by the next two parameters. The last parameter sets the seed-size inside this rectangle. Another way to set coordinates is by referring to another part's node. This is only possible if the part referred to does not change its mesh.

```
P1=PartList[1].cleanName+'-'+repr(NextCaeCycle)
mw.Window((P1,800,-2),(P1,800,1.2),(P1,800,20),(P1,800,-1.2),0.01)
```

Here, instead of entering absolute coordinates, only the relative distance to the node 800 of the second part inside the *PartList*-list is set. It is also possible to use absolute and relative coordinates at the same time.

```
P1=PartList[1].cleanName+'-'+repr(NextCaeCycle)
P2=PartList[2].cleanName+'-'+repr(NextCaeCycle)
mw.Window((P1,800,-2),(P2,202,1.2),20,-1.2),0.01)
```

In this example the first x-coordinate refers to the part P1 and the y-coordinate refers to the part P2. The second x and y-coordinate are absolute coordinates. In the AS, the number in the name of the part-instance and the part always match the number of the next cycle. Therefore, the name consists of the clean name (without any numbers in the name) and the number of the next job. To get the clean name, the name of the current cycle, or the name of the part in its initial state one can type the following:

```
>>> PartList[0].cleanName
"WORKPIECE"
>>> PartList[0].name
"WORKPIECE-3"
```

```
>>> PartList[0].initialName
"WORKPIECE-1"
```

Finally, the function *applyMW* creates the mesh windows and generates the mesh based on the user's data.

```
mw.applyMW()
```

Reconstruct the sets: After the deformed parts are imported and remeshed, the old node-sets are no longer valid. To reconstruct those sets, one could use the function *createSetbyName*. By creating set-names with a specified keyword at the name's end, the function *createSetbyName* can recognize those keys and search for nodes in the key-area. There are four possible key-areas to specify an edge and four key-areas to specify a corner:

- __B = bottom
- __L = left
- __R = right
- __T = top

- _PRB = point right bottom
- _PRT = point right top
- _PLB = point left bottom
- _PLT = point left top

If a set-name ends with __B, *createSetbyName* will search for all nodes at the bottom of the mesh. Therefore, when creating the initial-model, the use of straight edges at modeled geometries could be an advantage. If a part in the assembly will not be remeshed, the node-labels are stored and *createSetbyName* automatically restores the set on the part, regardless if it is deformed or not. To do that, the function *createSetbyName* accesses the *PartList*-list to find the relationship between the part and the set-name of the node set, which was analyzed by the function *analyse*.

```
createSetbySetName(PartList)
```

ReconParts: This function uses the methods *orphanImport*, *orphanImportGeo* and *makeInstance*, which all can be found in the KS. *ReconParts* opens the last ODB-file and imports the deformed parts into the current model. When the part is set as a remesh-part, it is imported as an orphan mesh and afterwards the orphan mesh is converted into a geometrical entity. If the part is set as a non-remesh part, only the orphan-mesh will be imported into the current model. If the part's type is rigid, only the name of the part instance will be changed inside the model in order to match the current CAE-cycle (e.g.: *'rigid-12'*).

GetSideEdge: The function *getSideEdge* creates a surrounding surface on a mesh by comparing the connectivities of all the elements. *getSideEdge* needs the part's name and the surface name as parameters. If the provided surface name already exists, it will be overwritten. This is essential when a non-remesh part like a tool is used to update the surface name.

```
getSideEdge(PartList[0].cleanName+'-'+repr(NextCaeCycle),
    SurfaceName='kontakt_werkzeug_s')
```

When the surface of a remesh part is required one can rely on the geometry's edges and easily create a surrounding surface by using the following command:

```
A.Surface(side1Edges=A.instances[PartList[1].cleanName+
    '-'+repr(NextCaeCycle)].edges,
    name='kontakt_werkstoff_s')
```

Reassigning the material: The KS-function *assignSections* automatically reassigns the materials. The reference connecting a part's material to the part is stored in the *PartList*-list.

Adding the MAP SOLUTIONS keyword: The KS-function *AddMapSolutions* writes the following statement into the model-keywords and returns *True* if it has executed successfully:

```
MAP SOLUTION, STEP=step-2,INC=24, UNBALANCED STRESS=RAMP
```

Here *STEP* is the last step of the model and *INC* is the last increment of the last simulation. The UNBALANCED STRESS-key can be either RAMP or STEP. The key can be changed in the KS.

Creating a new job: The KS-function *createJob* creates a new job with the number of the current CAE-cycle.

Starting a job: The KS-function *startJob* creates a subprocess, which is opened in a separate terminal. As long as the job runs, the terminal window will remain open. Additionally the parameters *DEL* and *ChkStep* can be used. *DEL* can be set to *TRUE* to prevent an extensive amount of data. It deletes all job-files that are more than nine jobs behind the current one, except the odb-files. *ChkStep* looks into the last job's status-file and returns *True* if the given step contains a valid increment. This can be useful when slight differences in the mesh cause divergence. In this event the KS-function *startVariation* can be executed to change the size of the global mesh and the mesh windows a little bit and restart the simulation again. After four attempts to alter the mesh window coordinates and the element size in order to converge, the AS terminates.

2.2.2 User file

User.txt is a parameter-file for user-parameters. It allows to specify parameters like the initial CAE-name, step name, import step and many more. The KS function *StartAutomaticProcedure* reads the user-file and updates the predefined values.

```
#sample user input script#
initialCaeName='realmatr_5celsius_biggerMesh_Rigid.cae'
StepName='step-2'
remesh=['werkstoff',]
newBcVal={}
#orphan part properties
deformed=['WERKZEUG','WERKSTOFF']
importStep=1 #1 is the second step
ATV=0.005
deformedShape = DEFORMED
maxCycles=1000
globalAbstractAngle=10
abstractAngle={}
```

2.2.3 Subroutine

ABAQUS^{TM}/Standard provides users with an extensive array of user-subroutines that allow them to adapt ABAQUS^{TM}to their particular analysis requirements. They can interact with the analysis at different points. To include a user-subroutine in an analysis the user parameter can be specified when executing a job. The KS-function *startJob* does that automatically. Usually the command would look like this:

```
abaqus job=Job-2 oldjob=Job-1 user=subroutine
```

UVARM: *UVARM* is used to determine the relative change of the plastic strain during the simulation. *UVARM* accesses all calculation points of elements at each increment in a step.

URDFIL: *URDFIL* reads the data from the results file (*.fil) in order to use the information to make decisions such as when to terminate the analysis. It interacts with the analysis between the point where the output gets written and the end of the increment. *URDFIL* compares the calculated plastic strain value from *UVARM* with a user defined limit (e.g. 0.7). If the relative plastic strain exceeds this limit the LSTOP flag triggers the simulation to stop.

Both subroutines can be found in appendix A.

2.2.4 Interpolation

2.2.4.1 Functionality

The interpolation algorithm in ABAQUS^{TM}is called Mesh-to-Mesh Solution technique and is only available in ABAQUS^{TM}/Standard.
By providing the keyword *MAP SOLUTION the data of the old mesh will be interpolated onto the new mesh. The kernel script automatically looks for the latest step and frame to define the *MAP SOLUTION parameters. The values of the field outputs at the nodes of the old mesh are interpolated onto nodes or integration points of the new mesh. The nodal values can be directly associated with the new nodes. The integration point variables are in a first step extrapolated to the nodes of each element, and

then they are averaged over all similar elements. In a second step, the location of each point in the new mesh in reference to the old mesh is obtained. Afterwards, in case of element variables the nodal data are interpolated onto the integration points of each element and in case of nodal variables the nodal data are interpolated directly onto the new nodes. The performed averaging and extrapolation causes a smoothing of strong gradients (Figure 2.8). Especially when the size of the new mesh in respect to the old one is quite different. This effect can be controlled with a dense mesh in high gradient regions such as the primary deformation zone of the cutting process. As described in chapter 1.3.3 ABAQUS^{TM}/Standard uses a Lagrangian formulation in which the mesh is attached to the deforming material. The discretization can degrade when the elements become severely distorted due to large strains. Therefore an indicator that triggers the remeshing procedure is necessary. In this simulation the indicator is the equivalent plastic strain and when triggered it causes the simulation to terminate and restart with a new mesh (see chapter 2.2.1).

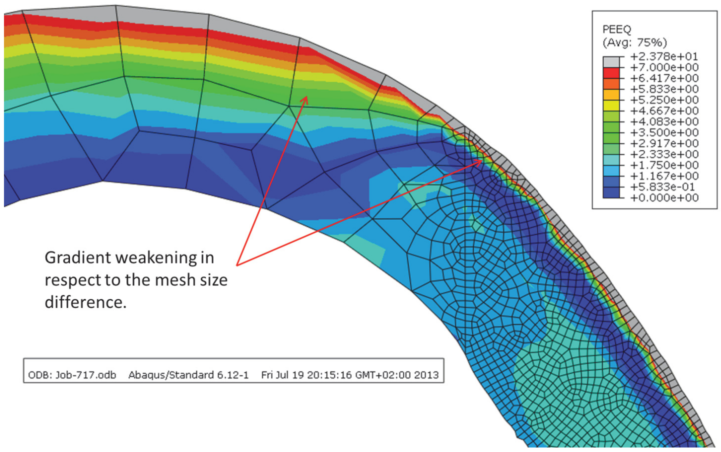


FIGURE 2.8: Smoothing of the solution gradients due to different mesh sizes.

To enable reasonable computation times, the simulations in this thesis show big mesh size gradients. The zones where the mesh size is coarser show a strong smoothing of the gradients as shown in Figure 2.8. A coarse mesh is only used at regions which are of low interest and which do not affect zones of interest. The mesh size in the primary and secondary deformation zone is small enough, that such a gradient flattening can be avoided. This avoids an unacceptable change of the mapped results after the remeshing procedure. The gradient flattening can also be reduced by resetting the trigger condition to a value that remeshes earlier and at a higher similarity of the elements of a less deformed mesh and a new mesh.

Chapter 3

Results

3.1 Evaluation of the mesh dependency

3.1.1 Remeshing versus constant mesh

Simulating the formation of a chip always includes heavy deformations of the elements within the chip. For numerical reasons the results obtained from an excessively distorted mesh start to deteriorate extremely. A model without remeshing and a model using the remeshing algorithm are simulated to demonstrate the differences. Figure 3.1 shows the contact forces of two remeshed models and a model with a constant mesh, when the tool is penetrating the workpiece for the first time. The remeshed models are simulated with a rigid as well as an elastic-plastic tool. The red line represents the results for a constant mesh and an elastic-plastic tool. Although remeshing does not take place even after the elements show extreme distortion the analysis produces results for a surprisingly large tool displacement. However, after three millimeters of cutting, the contact force of the constant mesh starts to stagnate, even though the steady state has not been reached at this point. This is the point, where element deformation is already too severe and numerical aberrations occur. The contact forces of the remeshed models continue to rise even at displacements where the constant mesh model has long failed to deliver reasonable output data. The curves of the remeshed models exhibit a sawtooth-like behavior. This effect is caused by the remeshing procedure itself. After the part has been remeshed and simulation results and the internal variables from the previous simulation such as the accumulated plastic strain, the temperature and the stress have been mapped onto the new mesh, a new simulation cycle starts. In the initial state of the simulation after the remeshing process, the contact has not been established yet and hence no contact forces are acting on the contact surface. Within the first step, the contact forces recover again, but during this procedure the contact pairs slip slightly relative to each other leading to numerical artifacts in the results. These artifacts cause fluctuations in the results of e.g. flickering of the stress distribution and consequently a flickering of the strain rate in the videos (Videos can be seen in the attached CD or online with the QR codes in Figure 3.2). The difference between the rigid tool's contact force (green line) and the elastic-plastic tool's contact force (black line) is due to the different interactions between the tool's and the workpiece's material. In contrast to the rigid tool, the elastic-plastic tool slightly displaces when it gets in contact with the workpiece. This shift causes a change in the tool's rake angle. Hence the material can slip easier underneath the cutting edge which results in a decrease of the contact force. Consequently, the temperature distribution is different which also effects the chip width. The temperature of the workpiece in the model with the elastic-plastic tool reaches about 860°C whereas the rigid tool model exhibits temperatures of about 940°C. However, the higher temperature enables easier shearing, which causes the chip width to increase and therefore the contact force to increase as well. The chip width of the model with the elastic-plastic tool is 5% smaller than the chip width in the model with the rigid tool which results in the difference between the two force curves in Figure 3.1.

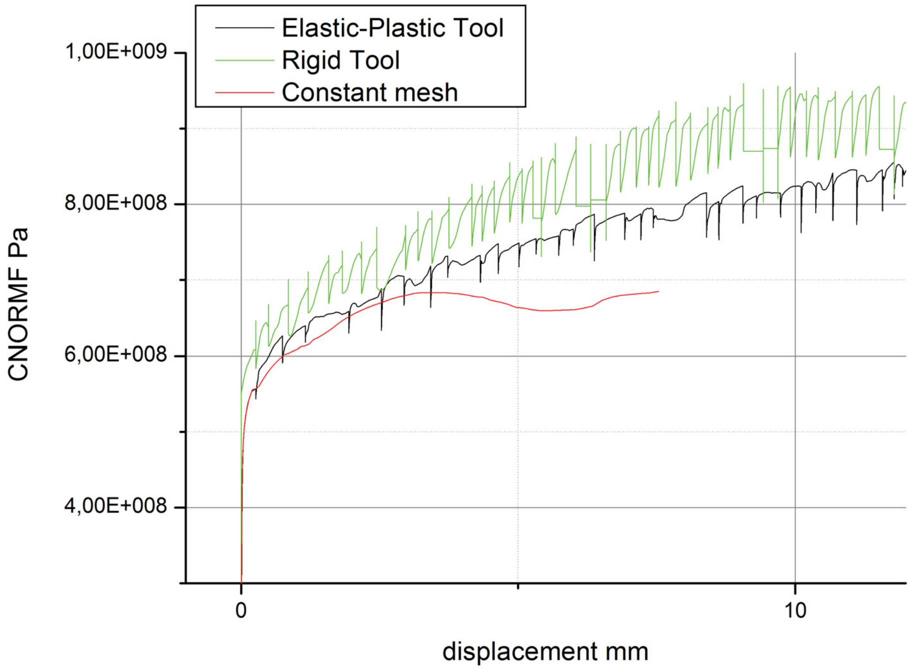


FIGURE 3.1: Remeshing versus one step simulation.

(a) (b) (c)

FIGURE 3.2: QR code for the videos (a) Strain rate of an inhomogeneous chip formation, (b) Nodal temperature and (c) Equivalent plastic strain of a homogeneous chip formation.

3.1.2 Analytical verification of the simulated cutting forces

The cutting force simulations are validated analytically using equation 3.1, where F_s is the cutting force, b the width of the chip, h the chipping thickness, k_s the specific cutting force, K_g the cutting rake correction, K_v the cutting speed correction, K_{st} the chip compression correction and K_v the wear correction. A milling procedure with a hard metal cutting tool is assumed. The steel CK60 with a Young's modulus of 210 GPa and a density of 7.8 g/cm^3 is used to estimate the material parameters z and $k_{s1,1}$

Parameters		
a		0.4 mm
h		0.8 mm
z		0.82
K_v		1.2
g_0		6° (for steels)
K_{st}		1.2
$rakeangle$		17°
v		220 m/min
$k_{s1.1}$		2130 MPa
b	$b = \frac{a}{cos(rakeangle)}$	0.418 mm
K_g	$K_g = 1 - \frac{rakeangle - g_0}{66.7}$	0.835
K_v	$K_v = 1.03 - \frac{3-v}{10^4}$	1.05
k_s	$k_s = k_{s1.1}h^{-z}$	1773.8 MPa

TABLE 3.1: Equations and parameters of the analytical calculation

(The chosen parameters for the cutting force are listed in table 3.1).

$$F_s[N] = bhk_s h^{1-z} K_g K_v K_{st} K_{ver} \tag{3.1}$$

This results in a total cutting force of $F_s = 936\text{x}10^6$ N. The simulation shows a maximum cutting force between $780\text{x}10^6$ N and $850\text{x}10^6$ N. Comparing these two results, the discrepancy between analytical and numerical calculated forces is remarkably low (The formula for the cutting force and the parameters can be found in [28])

3.1.3 Mesh size dependency

In a cutting process the main plastic deformation is concentrated in a shear band reaching from the tool tip to the root of the chip. Within the shear band the high plastic deformations causing dissipative heating lead to a temperature increase in the shear band and consequently to a thermal softening of the material. Temperature dependent softening effects may result in material instabilities and a mesh-dependence of the simulation results. This can be overcome by means of regularization techniques, e.g. by introducing strain rate dependency into the material law. In this thesis, the Johnson Cook model [10], describing strain rate sensitivity, strain-hardening/softening and temperature dependency was used. Therefore an element size related mesh dependency is expected and analyzed. From a theoretical point of view, the shear band width in the primary deformation zone (see Figure 1.1) tends to zero. However, in a numerical analysis the shear band spans over at least one element and furthermore the orientation of the elements also affects the deformation zones and eventually the chip's formation. But also in reality a finite width of the shear band is observed which in the literature is commonly explained by a spatial strain gradient dependency of the material behavior. Introducing that phenomenon in a numerical algorithm regularizes the otherwise discrete nature of the shear band. However, this is not the focus of this thesis and further investigations would go beyond the scope. The work of Hortig [29] elaborates that topic in greater detail. Furthermore, lowering the element size to a point where regularization effects no longer apply was precluded due to the immense consumption of computation time. Therefore, a parameter study with three different mesh sizes is conducted to find out which mesh size is acceptable in relation to its computation time (Figure 3.3). When the tool's cutting edge penetrates the workpiece, the cutting force increases up to a steady state, where the equilibrium between chip thickness, contact length and temperature is reached. These parameters are highly depending on the size of the elements.

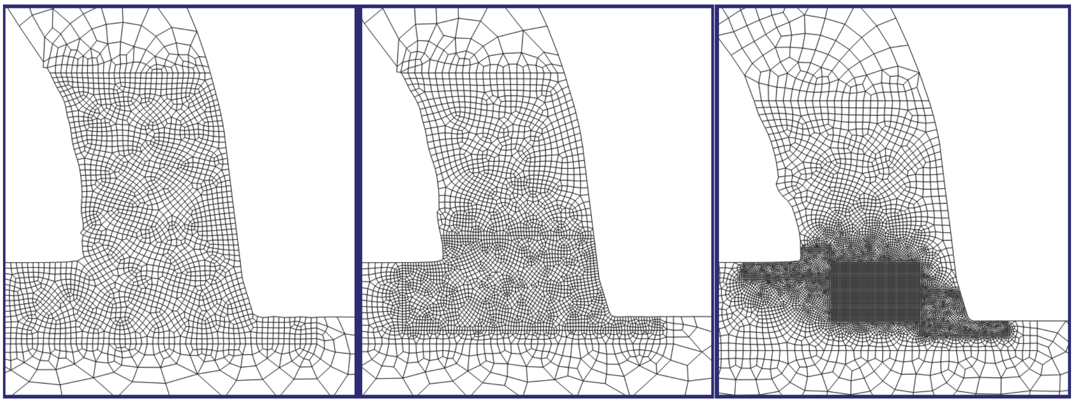


FIGURE 3.3: Primary deformation zone with three different mesh sizes. 0.05 (left), 0.02 (middle) and 0.01 (right).

The chip's formation is simulated with a coarse mesh until it reaches a steady state. Afterwards, the mesh covering the primary deformation zone is refined. The first chip in Figure 3.3 shows no mesh refinement, whereas the second chip shows half the element size of the first chip and the third chip approximately a fifth of the first element size. In all three simulations a steady state is reached. The abrupt change in the primary deformation zone from a coarse mesh size to a small mesh size has an impact on the material's plastic behavior, which causes the strain rate and thus the temperature to increase, leading to temperature dependent softening of the material, and therefore an instantaneous increase of the chip's width. The effect decays completely when steady state is reached again, leaving a small nose on the inner surface of the chip. The chip with the finest mesh shows a smaller chip width compared to the coarse mesh because the shear band is more localized and the shearing zone is more focused and therefore has higher motion constraints than in the case of a coarse mesh.

As mentioned in section 1.2, the contact between tool and workpiece or chip, respectively, can be separated into two distinguished areas. The slipping area, where the material can move relative to the facing material, and the sticking area, where no relative motion is possible. Figure 3.4 shows the beginning of the cutting step. Only the workpiece with the chip is visible. Along the surface, the sticking areas are marked red and the slipping areas are marked green. Areas without contact are marked black. The three plots are arranged in the same way as in Figure 3.3. The coarse mesh shows sticking along the surface except for the indifferent contact situation at the tool's tip, where slipping, sticking and contact opening can be seen. The third plot's sticking area is smaller than the sticking area of the other plots. This is an expected effect, because the finer mesh causes the chip's width to decrease and therefore the contact length decreases as well. Hence, a stronger bending of the chip occurs. Furthermore, smaller elements adapt better to the intricate geometry around the tool's edge, causing a much more accurate representation of the contact situation.

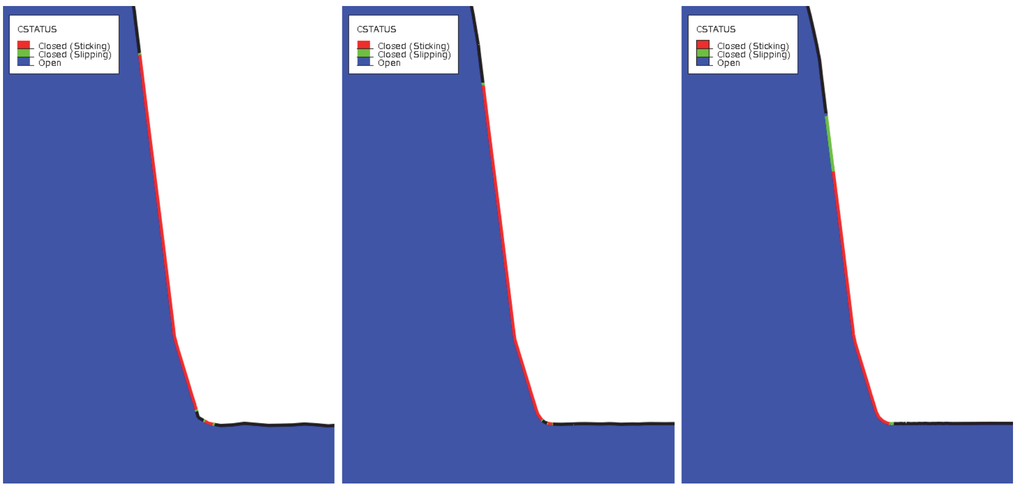


FIGURE 3.4: Contact status at the beginning of the second step.

Figure 3.5 shows the slip stick distribution at the end of a simulated cycle. As the coarse mesh's slip stick zone distribution is judged as rather unrealistic compared to the idealistic conditions visualized in Figure 1.1, attention is drawn to the second and third

Cycles	*Time*	*Elements*	*used disk space*
18	70 min	4150 Elements	1.2 GB
35	$8h45$ min	14300 Elements	7.9 GB
7000	18000 h	54000 Elements	6800 GB

Table 3.2: Computational stats

plot. Comparing the slip stick areas between the three plots, the difference between the second and third plot is much less than between the first and second plot. Two slipping areas (green) can be found inside the sticking region of the second and third plot. This might be caused by the elbow in the tool's geometry, which influences the effects in the upper and lower surfaces on the tool. Table 3.2 shows the time and the amount of remeshing cycles needed to simulate a homogeneous chip formation with 0.7 mm displacement of the tool displacement (see tab. 3.2. Since the computation time of the finest mesh is exponentially higher than the time needed to complete the second mesh, the latter is used for all subsequent simulations.

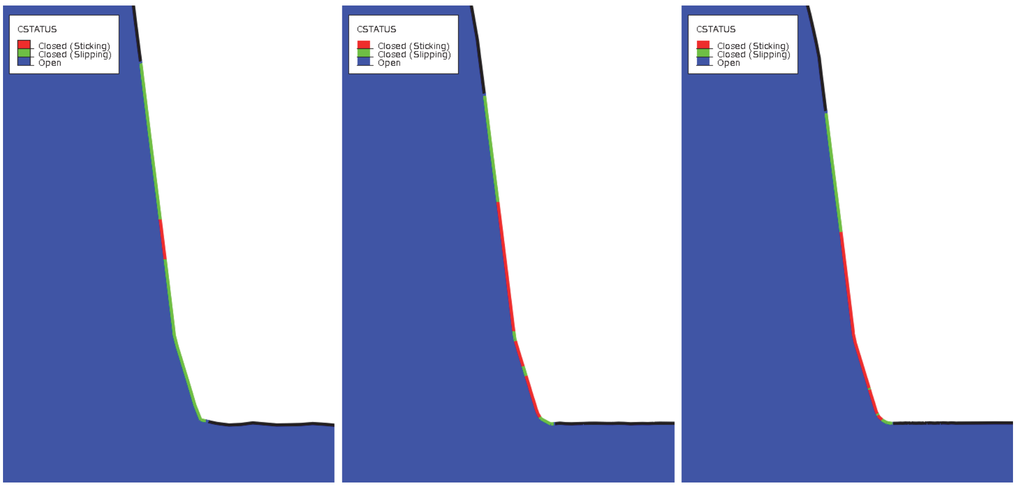


Figure 3.5: Contact status at the end of the second step.

The hardly predictable contact situation also has an effect on the temperature distribution. Only areas where the contact is established are used by the friction model, which in turn has effects on the friction related heat generation. Similar to the friction-induced heat generation, also the inelastic heat fraction defining the heat generation caused by plastic dissipation produces significant amounts of heat. Thus, the third plot in Figure 3.6 shows a much higher nodal temperature at the surface than the other two plots. The maximal temperature in the first plot is about 900°C versus 1400°C in the third plot.

At this temperature a phase transition can be expected. Therefore, the temperature should be less than 1400°C, otherwise some of the heat energy is used for the phase transition. However, phase transitions are not considered in this simulation.

The primary deformation zone's mesh-resolution increases with a decreasing element size. Figure 3.7 shows that the accumulated equivalent plastic strain (PEEQ) is more focused at regions with higher mesh resolution. Thus, even the refinement of the mesh can cause adiabatic shearing due to the size effects mentioned earlier. This trend appears clearly at the refining point in plot three. Compared to the other two plots, PEEQ is much higher in the refining zone (see Figure 3.7), which is caused by the increased strain rate. The increase of the strain rate can be seen in Figure 3.8, where the first plot shows strain rates of about 5×10^4 s^{-1}, while the third plot's strain rates increase to 1×10^5 s^{-1}.

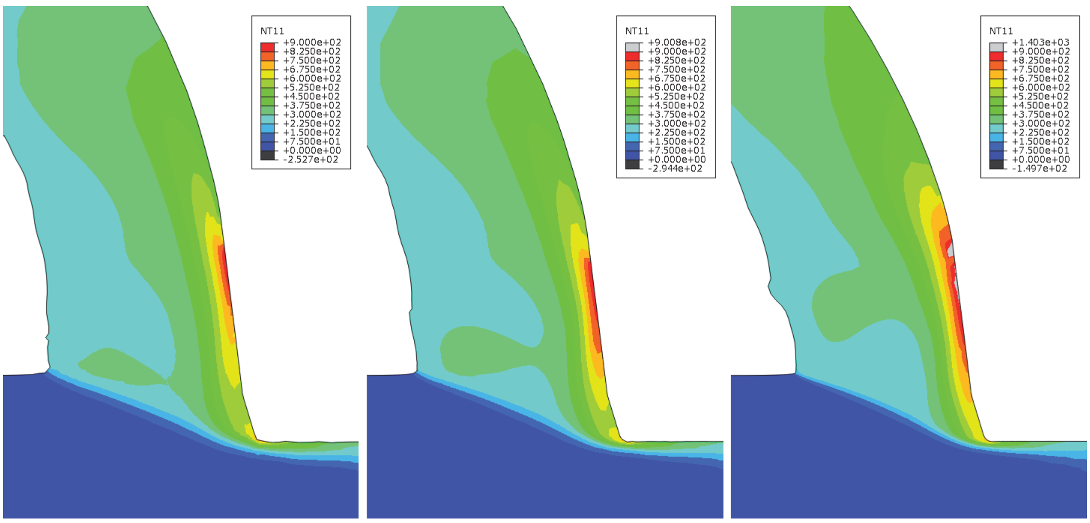


FIGURE 3.6: Chip edge temperature distribution.

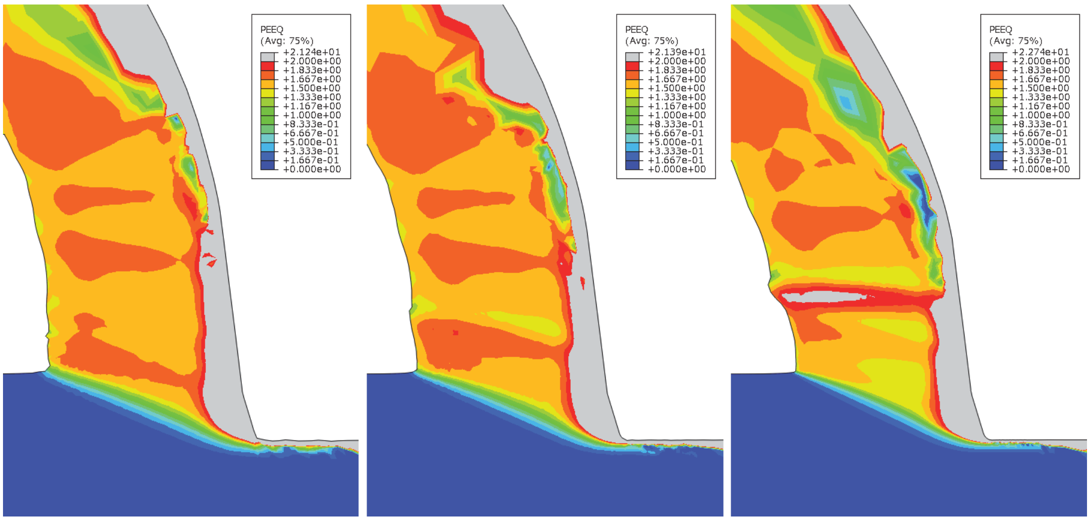


FIGURE 3.7: PEEQ distribution.

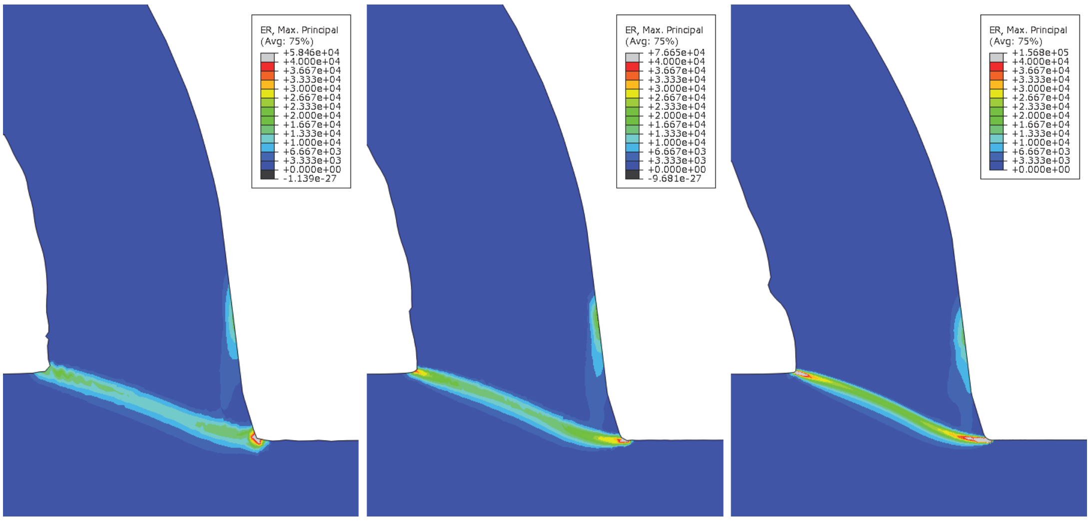
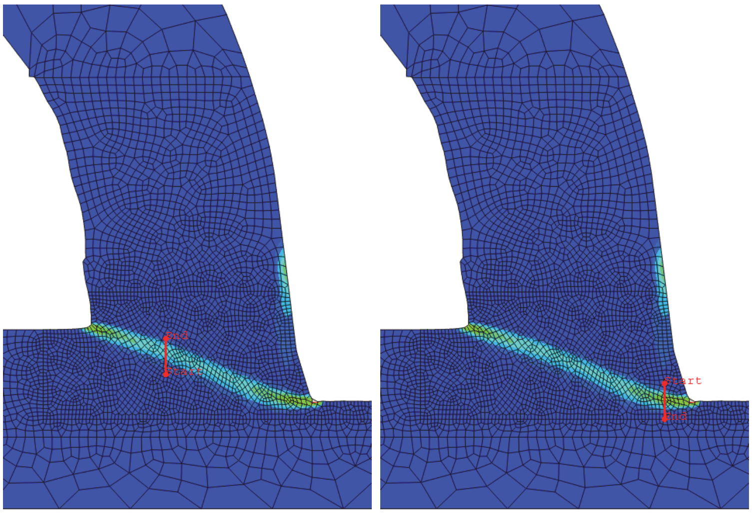


FIGURE 3.8: Strain rate distribution.

3.1.4 Deformation zone profiles

As an alternative indicator visualizing the impact of the different elements the strain rate distribution is evaluated along two different paths in the primary deformation zone revealing significant differences. Figure 3.9 shows the two paths in the chip's primary deformation zone, referred to in Figure 3.10. Mesh 1, 2 and 3 in that figure correlate to the meshes shown in Figure 3.3. As expected, the strain rate of mesh 3 is higher than the strain rate of mesh 2 and mesh 1.



(a) (b)

FIGURE 3.9: a) Path A through the midsection of the primary deformation zone. b) Path B through the edge of the primary deformation zone.

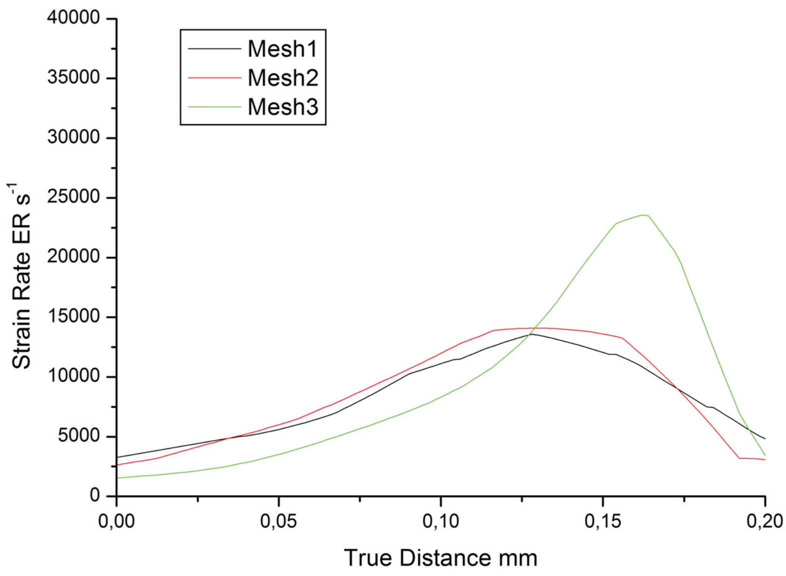
3.2 Homogeneous simulation results

3.2.1 Chip formation

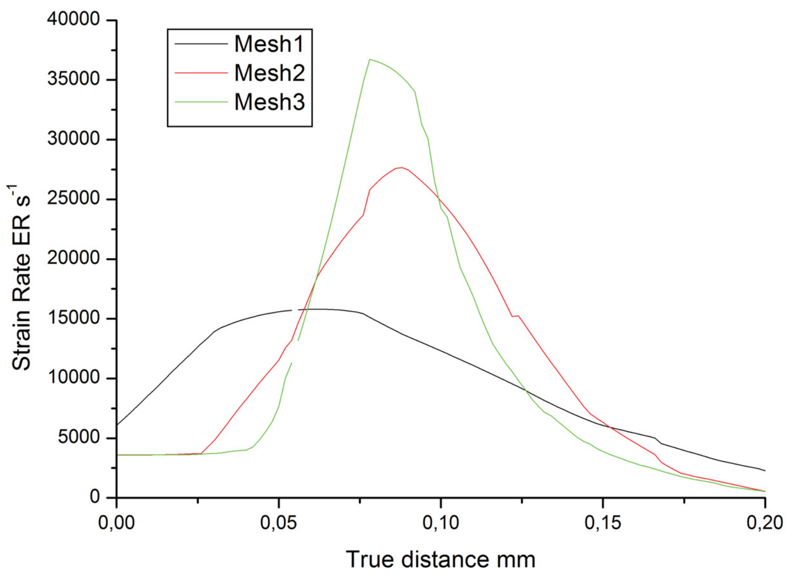
When the chip bends over to the uncut material, two possible scenarios can be simulated: (i) "with self contact": The chip gets in contact with a rigid body, simulating self contact of the chip with the surface of the uncut workpiece as mentioned in chapter 2.1.1, and (ii) "without self contact": the tip of the chip gets trimmed at each remeshing-cycle.

3.2.2 Forces and contact length

In Figure 3.12 (a), the chip-width over time curve shows that the decrease of the chip's width is almost the same within the first six milliseconds for the two considered scenarios described above. The decrease of the chip's width is caused by a constant decrease of the cutting depth, which simulates the cutting in a milling process. The difference of the chips width at the beginning of the curves is due to different mesh sizes at the beginning of the simulation. The reason for the small kink in the curve of the model without self contact (red plot) after four milliseconds is due to changes of the mesh for simulation stability reasons. After six milliseconds, the chip width in the case "with self contact" ascends by about 0.05 mm but remains parallel to "without self contact"



(a)



(b)

FIGURE 3.10: a) Strain rate along path A. b) Strain rate along path B.

curve afterwards. This is where the chip gets in contact with the rigid body. When in contact, the chip's neck erects causing the contact length between the tool and the chip to increase (see Figure 3.12 (b)). This increase also affects the primary deformation zone and causes the chip's width to increase. Comparing Figure 3.12 (a) with Figure 3.12 (b) suggests that a change of the contact length always results in a change of the chip's width.

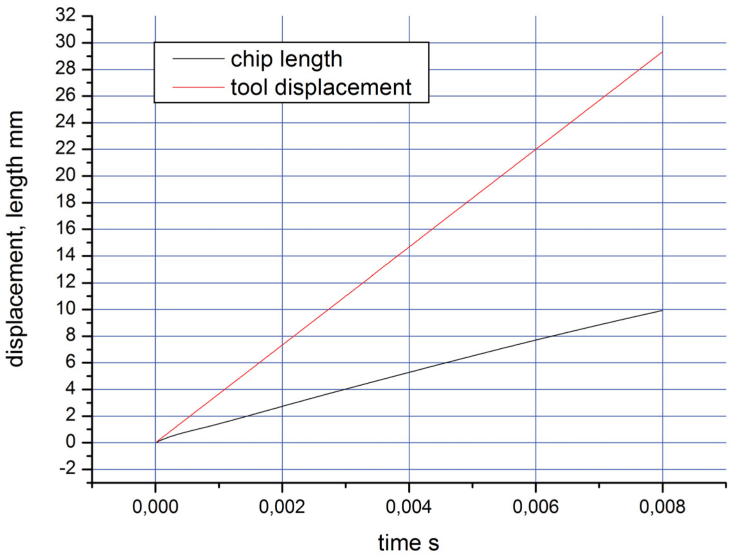
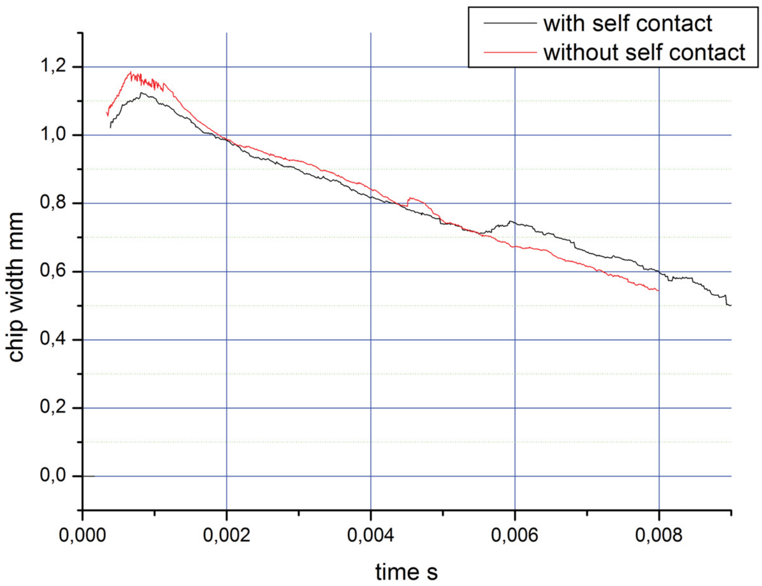
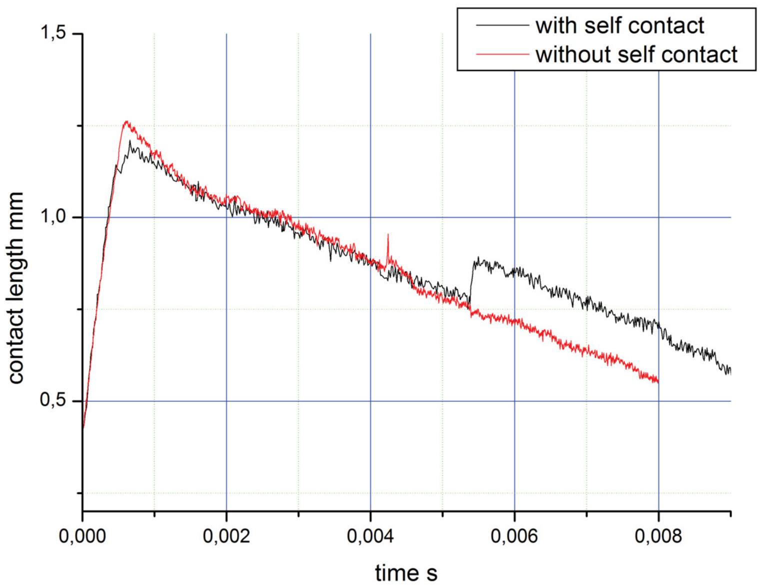


FIGURE 3.11: Tool displacement and the corresponding chip length.

Figure 3.11 shows that although the chip's width changes in the model with self contact, the chip's rate of growth does not change over time. Hence follows that the material flow of the chip is constant, even when the contact length, the temperature or the chip's width changes. The chip's formation length is about 35% of the tool's displacement.



(a)



(b)

FIGURE 3.12: a) Comparison of the chip's width with and without self contact. b) Comparison of the chip's contact length with and without self contact.

3.3 Inhomogeneous simulation results

The study of inhomogeneous chip formation can be of importance for realistically simulating the behavior when inhomogeneities such as inclusions reach a size that influence the process. Therefore, a study is conducted to investigate the influences of different inclusion sizes at different positions in relation to the formation of a chip. The parameter study features three different inclusion sizes with a diameter of 0.15 mm, 0.1 mm and 0.05 mm at three different positions. The cutting depth d is 0.4 mm. Their sizes in relation to the uncut chip thickness are about $0.37d$, $0.25d$ and $0.125d$. First, a homogeneous reference case is simulated until the steady state is reached. Then the inclusions are implemented into the workpiece at three different positions. The first position is defined at $0.25d$ from the surface of the uncut chip. The second position $0.75d$ and the third position is positioned 0.2 mm underneath the cutting path (see Figure 3.13).

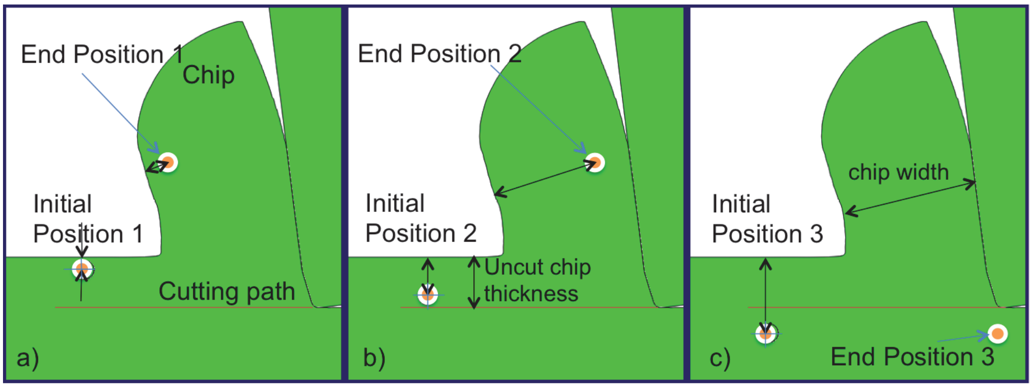


FIGURE 3.13: Positions of the inclusions: a) $0.25d$, b) $0.75d$, c) 2 mm underneath the cutting path.

3.3.1 Chip formation and plastification

Depending on the size and position of the inclusion, the chip forms differently. It turns out that the position closest to the surface has the most distinctive effect. The chip's width decreases when the inclusion is passing the primary deformation zone and a sharp notch forms perpendicular to the flow direction of the chip. After the inclusion has passed the primary deformation zone the chip width starts to increase again reaching beyond the steady state's chip width until it finally reaches its steady state again. The size of the sharp notch is related to the size of the inclusion and its position. Inclusions that are placed on position 2 shown in 3.13 (b) cause a shallower notch. The same is true

Relative inclusion size	*Initial position*	*End position*
$0.37d$	$0.25d$	$0.39d'$
$0.25d$	$0.25d$	$0.35d'$
$0.125d$	$0.25d$	$0.31d'$
$0.37d$	$0.75d$	$0.79d'$
$0.25d$	$0.75d$	$0.78d'$
$0.125d$	$0.75d$	$0.78d'$

Table 3.3: Inclusion positions

for smaller inclusions at the same position. The smallest inclusion with a size of $0.125d$ has almost no effect on the chip's formation. Figure 3.14 shows the chip for all three inclusion sizes and the sharp notch which forms differently depending on the inclusion size. The positions of the inclusions in regard to the uncut or cut chip width change slightly relative to their initial position. Table 3.3 shows that an inclusion which is positioned close to the surface is located considerably deeper inside the chip after it has crossed the primary deformation zone. d' is the chip width. Is the inclusion positioned close to the cutting path, a shift towards the surface can be seen. The influence of the size of an inclusion positioned near the cutting path is less pronounced than it is for an inclusion positioned near the surface. In this case the shift is smaller and the size of the inclusion shows less influence on the change of the relative position.

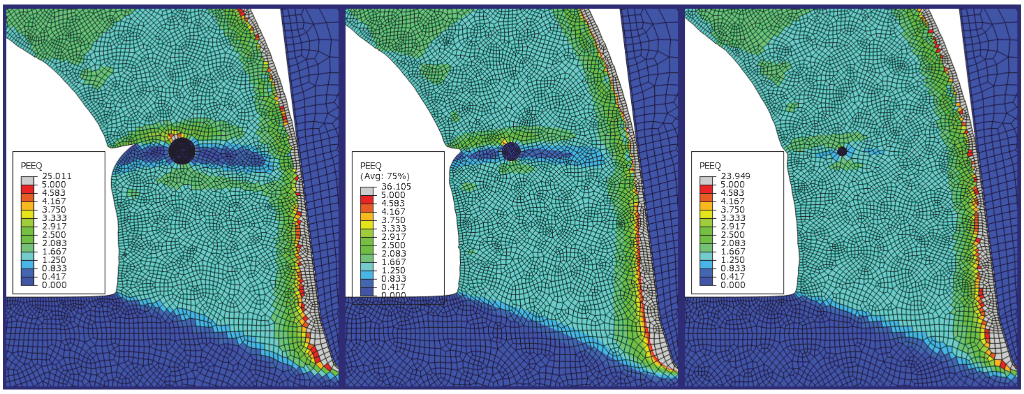


Figure 3.14: PEEQ Plot with different inclusion sizes.

3.3.2 Effects in the primary deformation zone

When the inclusion starts to cross the primary deformation zone, it hinders the chip-material from shearing. The material flow faces a smaller area to pass the primary deformation zone. This results in higher strain rates near the tool tip and at the upper side of the inclusion, which can be observed by comparing Figure 3.15 (a) and (b).

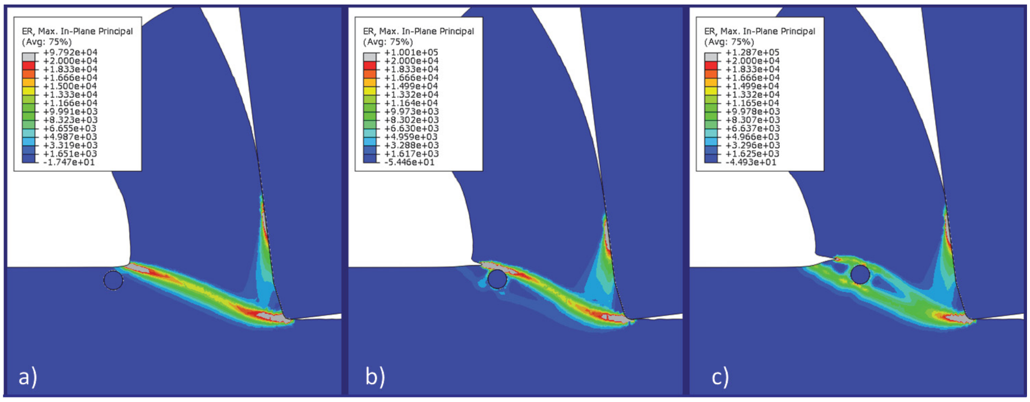


FIGURE 3.15: Change of the strain rate in the PDZ caused by the inclusion with the diameter 0.37d. a) before entering the PDZ, b) crossing the PDZ and c) leaving the PDZ.

At first the strain rate starts to increase at the top side of the inclusion and an S-form of the PDZ appears which becomes more distinctive at a later stage. The higher flow rate causes a local increase in temperature on the top of the inclusion, whereas on the bottom the opposite effect takes place. As the inclusion reaches the center of the deformation zone it hinders the shearing. This splits the material movement into two strain rate bands, which are situated at the top and the bottom of the inclusion. The split regions of the primary deformation zone show a significantly lower strain rate. Since the build up rate of the chip is constant, the expected increase of the strain rate occurs on the opposite side where the tool interacts with the chip. When the inclusion is about to leave the deformation zone, the strain rate in the PDZ starts to increase until the model has reached its steady state again. The bottom of the inclusion shows a zone with lower temperature. This lower temperature increases the chip's width, until the steady state has been reached again, resulting in a nose at the chip's outer surface.

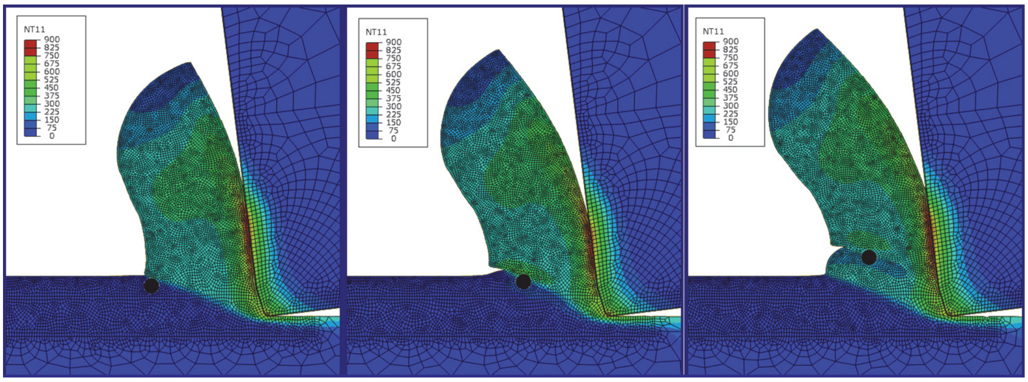


FIGURE 3.16: Change of the temperature caused by an inclusion.

3.3.3 Forces and contact length

When the inclusion is moving through the primary deformation zone the chip's width decreases and so does the contact length. Now, the chip's counterpressure against the tool is reduced and the cutting force decreases. Additionally, the crossing position of the inclusion can result in an advanced bending of the chip which results in an even more decreased contact length. When the inclusion is placed at initial position 3.15 (a), the material flow to build up the chip is hindered at the surface side, forcing it to cross the deformation zone close to the outer surface of the chip. This results in an increased material flow on the tool-side, which in turn causes an advanced bending of the chip. When the inclusion crosses at the midsection of the chip, a smaller bending can be observed. Figure 3.18 (a) shows the contact length over time with the biggest inclusion crossing the primary deformation zone at three different locations. The contact length at the top position experiences a decrease of the contact length of 22.7%, whereas the middle position's decrease of the contact length is only 9%. When the inclusion moves underneath the tool's edge, no significant change in the contact length is observed. Plot (b) shows the contact force of the same process. The top position reduces the contact force by almost 9%, the middle position by 6%. Placing the inclusion underneath the cutting path shows also no effect on the cutting force. The shift between the three plots is caused by the position of the inclusions. All inclusions start at the same point and move towards the primary deformation zone. Due to the shear band orientation, the inclusion of the top position reaches the primary deformation zone first followed by the midsection position 0.15ms later.

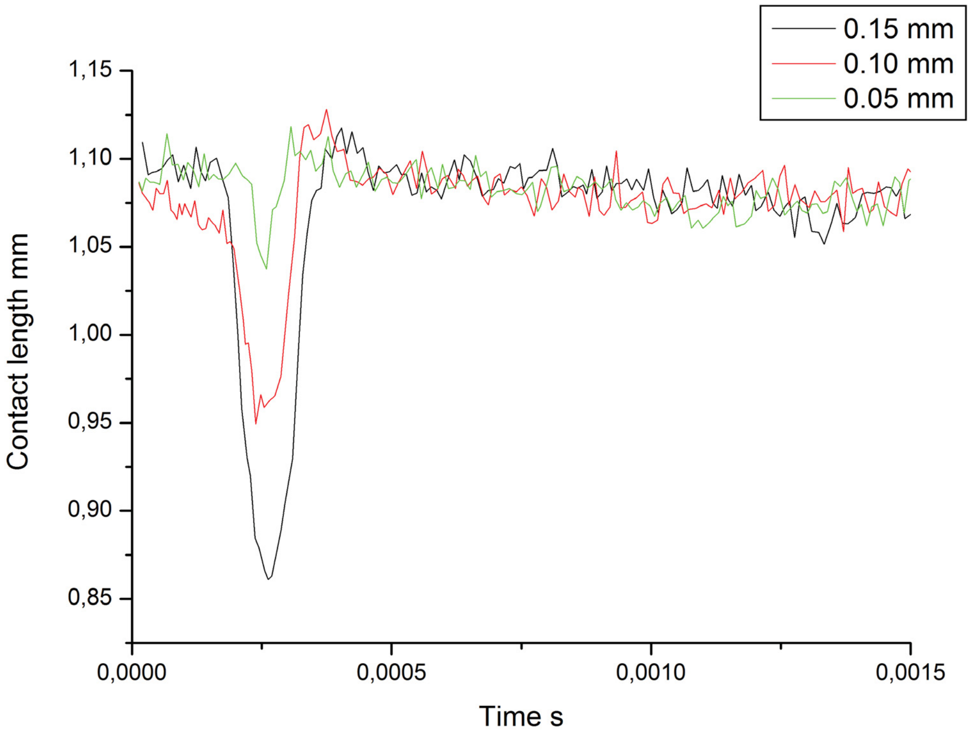
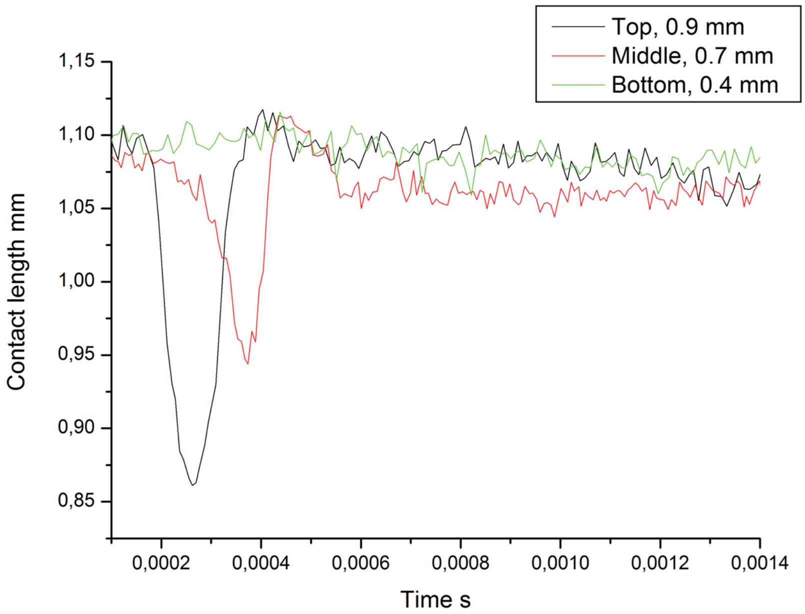
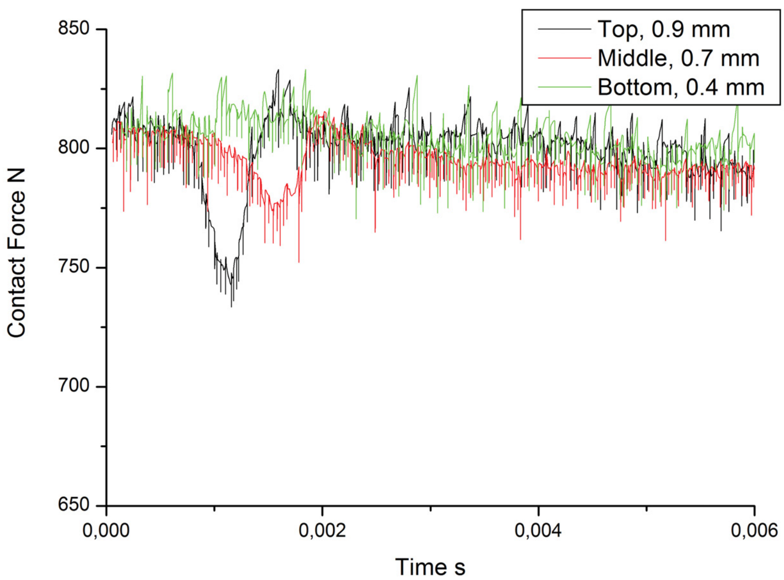


FIGURE 3.17: Contact length change depending on the size of the inclusion.

As expected, the contact length reduction is also related to the size of the inclusion. For the position close to the surface the biggest inclusion causes a decrease of the contact length of 22.7%, the medium size inclusion a decrease of 13.6% and the smallest inclusion a decrease of about 4.5% (see Figure 3.17).



(a)

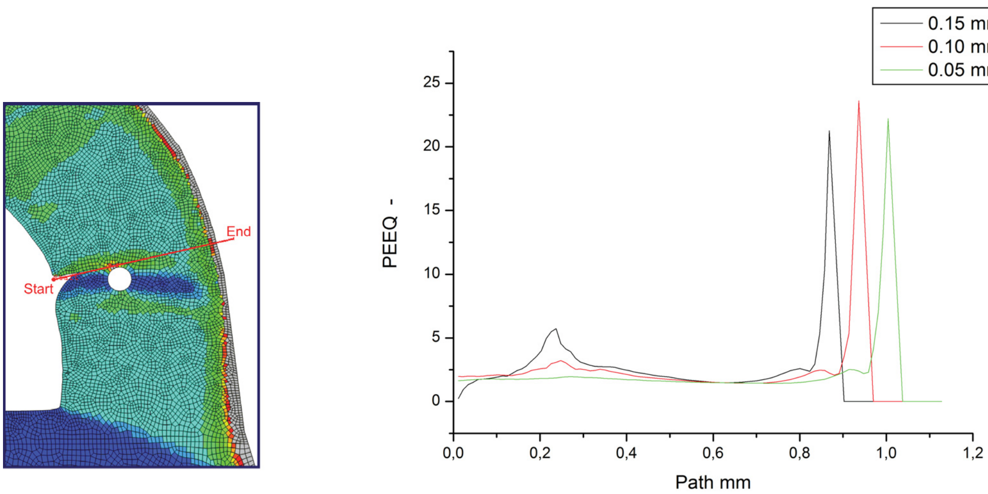


(b)

FIGURE 3.18: a) Contact length of the biggest inclusion at different positions. b) Contact force of the biggest inclusion at different positions.

3.3.4 Influence of inclusion on chip fracture

As mentioned in the previous section, the inclusion causes an increase in the strain rate while it is moving into the primary deformation zone. Therefore a PEEQ of more than 500% can be found at the top side of the inclusion. Figure 3.14 shows the PEEQ-distributions for the three inclusion sizes. In Figure 3.19 (a) an evolution is shown for the comparison of the amount of PEEQ on the upper pole of the three different inclusions. A big inclusion also results in a high amount of PEEQ at the top. Additionally, each graph shows a peak at the end of the plot, which represents the cut surface's plastic strain (see Figure 3.19 (b)). Beside the primary deformation zone, also the secondary deformation zone along the cutting edge causes high plastic strains. The different positions of the three peaks are caused by the different bending and chip thickness of the chips.



(a) (b)

FIGURE 3.19: a) Path through the chip on the top side of the inclusion. b) PEEQ distribution for different inclusion-sizes.

High values of plastic deformation can lead to ductile damage starting with the formation of pores under low negative or positive stress triaxiality. These pores can cause material separation and material failure which in this case may lead to fracture of the chip.

The position where ductile damage may cause a chip separation can be calculated for example with the damage indicator D_i by Hancock-Mackenzie. It depends on the stress triaxiality σ_H/σ_{eq} weighted by the function 3.3 and the equivalent strain ϵ_{eq} as expressed in equation 3.2 [30]. Ductile fracture most likely appears at the position with the highest

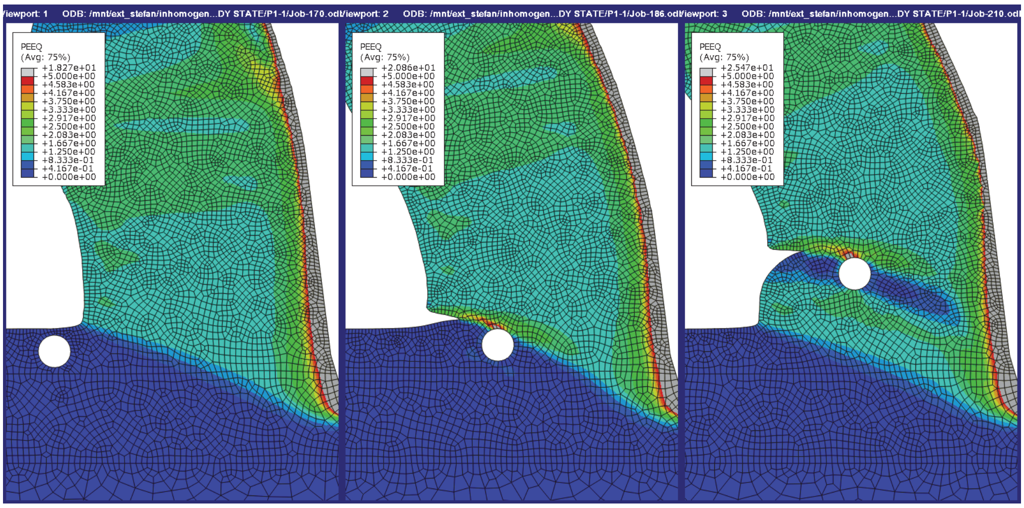


FIGURE 3.20: PEEQ around the inclusion.

damage indicator. The dependency of the damage indicator on the equivalent strain has to be calibrated experimentally and is expressed as the constant "C". To calculate the weighting function the parameter R is set to a value of 2 as in Gänser et al. [30].

$$D_i = \int_0^{\epsilon_{eq}} f_i \left(\frac{\sigma_H}{\sigma_{eq}}, \ldots \right) \mathrm{d}\epsilon_{eq} \tag{3.2}$$

$$f_1 \left(\frac{\sigma_H}{\sigma_{eq}} \right) = C \exp \left(R \frac{\sigma_H}{\sigma_{eq}} \right) \tag{3.3}$$

To estimate the impact of the stress triaxiality caused by the inclusion, three different elements are investigated with a simplified approach. The first position is located at the upper side of the inclusion, the second position at the inner side of the sharp notch witch evolves during the inclusion's pass through the PDZ, and the third position is located at the center of the PDZ (see fig. 3.21). The element with the highest value of the weighting function with respect to its change of plastic deformation indicates the position where ductile fracture will preferably occur. The most severe changes in plastic deformation at the inclusion's surface take place when the inclusion is moving through the PDZ. The resulting value of the weighting function also reaches a maximum when the inclusion is about to leave the PDZ. To calculate the damage indicator, the integral of all measured weighting values being a function of the plastic deformation increment has to be considered. In the cutting process the stress triaxiality at a material point moving through the PDZ can in a rough estimation be assumed to be constant. Hence for the sake of simplicity in this thesis the weighting function is assumed to be constant.

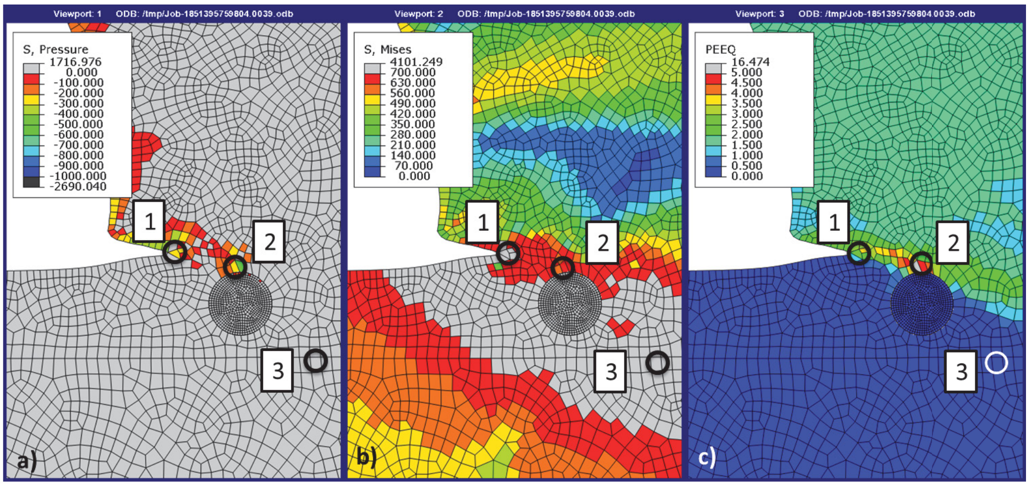


FIGURE 3.21: a) Negative hydrostatic stress distribution, b) Mises stress distribution and c) equivalent plastic strain distribution on the chip.

Nr.	σ_h[MPa]	σ_{eq}[MPa]	Triaxiality	*Average weight*	*Damage indicator*
1	324	698	0.46	2.5C	7.33C
2	479	637	0.75	4.5C	22.7C
3	-450	770	-0.58	0.5C	0.16C

TABLE 3.4: Damage indication

Far away from the inclusion (fig. 3.21, pos.3) only hydrostatic pressure acts on the investigated material point and so the damage indication is very small (see table 3.4). In the vicinity of the notch and the inclusion the material sees, in contrast to the rest of the PDZ, a positive stress triaxiality. Since the onset of damage is depending exponentially on the stress triaxiality and high plastic deformations, ductile fracture of the chip is likely to start from this region. The value of the damage indicator at the upper side of the inclusion (fig. 3.21, pos.2) is 3 times the amount of damage inside the notch (pos.1). Hence, the direct vicinity of the inclusion is most critical for damage initiation. Furthermore it indicates that the inclusion's geometry and position is indirectly responsible for the chip's separation.

3.3.5 Chip formation for a second cut

To get a general idea about the universal applicability of the common 2D remeshing tool (see chapter 2.2.1), a second cutting step is simulated. Figure 3.22 displays the first cut on the left side and the second cut on the right side. It is evident that after the first chip has formed, the hardened surface impedes the formation of the second chip, causing a change in the chip's buildup. The left surface of the second chip exhibits a highly deformed zone, leading to a decreased chip width.

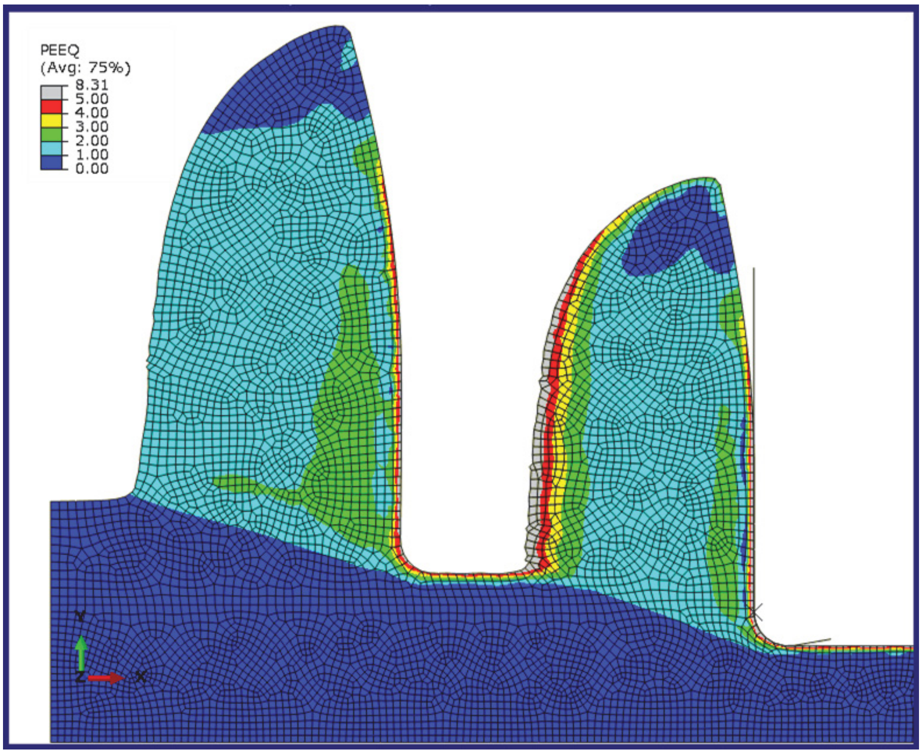


FIGURE 3.22: Comparison between the first and second cut.

Chapter 4

Discussion and Conclusion

The formation of the chip in today's machining is a complex process, which demands the development of simulation tools to predict aspects such as the formation of the chip, cutting forces, temperature distribution or tool wear. Depending on the aspects to investigate, three different methods are commonly used in the literature for chip formation simulations. The EULERIAN method, where the material flows through a stationary mesh, the LAGRANGIAN method, where the mesh moves with the material points and the ALE method, being a combination of both. The UL (Updated Lagrangian) method of the commercial finite element software ABAQUS^{TM}/Standard was used in this thesis. The large displacement during the formation of the chip causes severe distortion of the mesh. Thus, this method demands a repeated remeshing. Since a manual approach is cumbersome and time consuming an automation method was developed using Python-scripts. The scripts written for this thesis can be used to automate any 2D chip formation simulation. It is not limited to the amount of inclusions, material layers or parts. First, a parameter study regarding the mesh size dependence was conducted. It showes that with a smaller element size the chip width decreases and higher plastic deformations, higher strain rates and higher temperatures can be found. Inside the primary deformation zone, the strain rate of the mesh with an element size of 0.05 mm is about 14000 s^{-1}, whereas the observed strain rates of the mesh with the element size of 0.01 mm are between 23000 s^{-1} and 37000 s^{-1}. Thus changing the element size has a great impact on the results. The smallest element size still allowing an acceptable computational time was chosen for all subsequent simulations. A homogeneous CAE-model was built and simulated with these scripts. The chip of the first homogeneous model bends and may get in self-contact with the workpiece. Once in contact, the chip starts to rise resulting in an increase of the contact length between tool and workpiece of about 0.12 mm and an increase of the chip's width of about 0.05 mm compared to the contact length and chip width of the homogeneous model without self-contact. The contact length, force and chip width distribution remains constant after self-contact has been established. The results are compared with a model where self-contact is prevented by truncating the chip before self-contact can occur. The contact length and chip width of both models show a good agreement. Small variations between the results can be explained by slightly different element sizes used in the two simulations. The build-up velocity of the chip reaches 35% of the tool's velocity in both models. Finally, the influence of inhomogeneities in form of circular inclusions was investigated. A parameter study with three different hard inclusions with a diameter of $0.37d$, $0.25d$ and $0.125d$ was conducted, where d is the uncut chip thickness set to 0.4 mm. The study includes the effect of the three different inclusion sizes at three different positions. The first position is $0.25d$, which places the inclusion right under the workpiece's surface. The second position is at a depth of $0.75d$, which places the inclusion right above the cutting path of the cutting tool. The third position is underneath the cutting path. Placing the largest inclusion at the first position has the most significant influence on the chip's formation. When the inclusion approaches the primary deformation zone, the contact force decreases from about 810 MPa to 740 MPa and increases again to 810 MPa when the inclusion leaves the primary deformation zone. The contact length deviates between 1.1 mm and 0.86 mm. The effects are less pronounced with a decreasing inclusion size or an initial position deeper inside the material. After the inclusion has passed the primary deformation zone the position of the biggest inclusion changes from 25% of the uncut chip thickness to 39% of the resulting chip width. This change in position is also influenced by the inclusion size as well as its initial position. At the inclusion poles and at the tip of the sharp notch hydrostatic tensile stresses occur leading to positive stress triaxialities. A simplified investigation of ductile damage shows the highest damage indicator values at the inclusion poles while it is passing through the PDZ. Combined with the high damage indicator at the notch tip it will lead to ductile fracture of the chip.

Future investigations could e.g. include a damage model being capable of simulating the effects of ductile damage and chip fracture. Further objectives are the modeling of the formation of the chip with a soft inclusion such as graphite and the simulation of a second cut model to investigate the influence of ductile inclusions on chip formation in the presence of stresses at the surface. After calibration and an experimental validation the simulation tool shows the potential for realistically simulating cutting processes even taking into consideration complex material laws and sophisticated process conditions.

Bibliography

[1] B. Shi and H. Attia. Current status and future direction in the numerical modeling and simulation of machining processes: A critical literature review. *Machining Science and Technology*, 14(2):149 – 188, 2010. doi: 10.1080/10910344.2010.503455. URL http://www.tandfonline.com/doi/abs/10.1080/10910344.2010.503455.

[2] V. P. Astakhov and J. C. Outeiro. Modeling of the contact stress distribution at the tool-chip interface. *Machining Science and Technology*, 9(1):85 – 99, 2005. doi: 10.1081/MST-200051372. URL http://www.tandfonline.com/doi/abs/10.1081/MST-200051372.

[3] F. Klocke and W. König. *Fertigungsverfahren 1*. Springer-Verlag Berlin Heidelberg New York, achte auflage edition, 2008. doi: 10.1007/978-3-540-35834-3. URL http://www.springer.com/engineering/production+engineering/book/978-3-540-23458-6.

[4] Y.M Quan, Z.H Zhou, and B.Y Ye. Cutting process and chip appearance of aluminum matrix composites reinforced by sic particle. *Journal of Materials Processing Technology*, 91(1 - 3):231 – 235, 1999. ISSN 0924 - 0136. doi: http://dx.doi.org/10.1016/S0924-0136(98)00444-0. URL http://www.sciencedirect.com/science/article/pii/S0924013698004440.

[5] D. Umbrello, R. M'Saoubi, and J.C. Outeiro. The influence of johnson - cook material constants on finite element simulation of machining of aisi 316l steel. *International Journal of Machine Tools and Manufacture*, 47(3 - 4):462 – 470, 2007. ISSN 0890 - 6955. doi: http://dx.doi.org/10.1016/j.ijmachtools.2006.06.006. URL http://www.sciencedirect.com/science/article/pii/S0890695506001507.

[6] S.M. Afazov, S.M. Ratchev, and J. Segal. Modelling and simulation of micro-milling cutting forces. *Journal of Materials Processing Technology*,

210(15):2154 – 2162, 2010. ISSN 0924-0136. doi: http://dx.doi.org/10.1016/j.jmatprotec.2010.07.033. URL http://www.sciencedirect.com/science/article/pii/S0924013610002335.

[7] P.J. Arrazola and T. Özel. Investigations on the effects of friction modeling in finite element simulation of machining. *International Journal of Mechanical Sciences*, 52(1):31 – 42, 2010. ISSN 0020 - 7403. doi: http://dx.doi.org/10.1016/j.ijmecsci.2009.10.001. URL http://www.sciencedirect.com/science/article/pii/S0020740309002033.

[8] C. Duan, H. Yu, Y. Cai, and Y. Li. Finite element simulation and experiment of chip formation during high speed cutting of hardened steel. *Applied Mechanics and Materials*, 29-32:1838–1843, 2010. URL http://www.scopus.com/inward/record.url?eid=2-s2.0-78650796435&partnerID=40&md5=d14cf9bab10e59ae4dba57257b6ca642.

[9] Dassualt System. Abaqus 6.12 analysis user manual, 2012.

[10] G.R. Johnson and W.H. Cook. A constitutive model and data for metals subjected to large strains, high strain rates and high temperatures. *Proceedings of 7th International Symposium on Ballistics*, pages 12 – 21, 1983.

[11] A. Moufki, A. Molinari, and D. Dudzinski. Modelling of orthogonal cutting with a temperature dependent friction law. *Journal of the Mechanics and Physics of Solids*, 46(10):2103 – 2138, 1998. ISSN 0022 - 5096. doi: http://dx.doi.org/10.1016/S0022-5096(98)00032-5. URL http://www.sciencedirect.com/science/article/pii/S0022509698000325.

[12] M. Bäker. Finite element investigation of the flow stress dependence of chip formation. *Journal of Materials Processing Technology*, 167(1):1 – 13, 2005. URL http://www.scopus.com/inward/record.url?eid=2-s2.0-23144449134&partnerID=40&md5=a7bc912f6a8c8cd14a86da61b82cd1bb.

[13] M. Bäker. Finite element simulation of high-speed cutting forces. *Journal of Materials Processing Technology*, 176(1 - 3):117 – 126, 2006. ISSN 0924-0136. doi: http://dx.doi.org/10.1016/j.jmatprotec.2006.02.019. URL http://www.sciencedirect.com/science/article/pii/S0924013606002020.

[14] E. Ng and D.K. Aspinwall. Modelling of hard part machining. *Journal of Materials Processing Technology*, 127(2):222 – 229, 2002. ISSN 0924-0136. doi: http://dx.doi.org/10.1016/S0924-0136(02)00146-2. URL http://www.sciencedirect.com/science/article/pii/S0924013602001462.

[15] M. Bäker, J. Rösler, and C. Siemers. A finite element model of high speed metal cutting with adiabatic shearing. *Computers & Structures*, 80(5 - 6):495 – 513, 2002. ISSN 0045-7949. doi: http://dx.doi.org/10.1016/S0045-7949(02)00023-8. URL http://www.sciencedirect.com/science/article/pii/S0045794902000238.

[16] V. Kalhori. *Modelling and Simulation of Mechanical Cutting*. PhD thesis, Luleå University of Technology, 2001. URL http://pure.ltu.se/portal/files/155221/LTU-DT-0128-SE.pdf.

[17] M. Bäker. The influence of plastic properties on chip formation. *Computational Materials Science*, 28(3 - 4):556 – 562, 2003. ISSN 0927-0256. doi: http://dx.doi.org/10.1016/j.commatsci.2003.08.013. URL http://www.sciencedirect.com/science/article/pii/S092702560300140X.

[18] F. Zanger and V. Schulze. Investigations on mechanisms of tool wear in machining of ti-6al-4v using {FEM} simulation. *Procedia {CIRP}*, 8(0):157 – 162, 2013. ISSN 2212-8271. doi: http://dx.doi.org/10.1016/j.procir.2013.06.082. URL http://www.sciencedirect.com/science/article/pii/S2212827113003594.

[19] L.-J. Xie, J. Schmidt, C. Schmidt, and F. Biesinger. 2d {FEM} estimate of tool wear in turning operation. *Wear*, 258(10):1479 – 1490, 2005. ISSN 0043-1648. doi: http://dx.doi.org/10.1016/j.wear.2004.11.004. URL http://www.sciencedirect.com/science/article/pii/S0043164804004077.

[20] Y.B. Guo and D.W. Yen. A fem study on mechanisms of discontinuous chip formation in hard machining. *Journal of Materials Processing Technology*, 155 - 156(0):1350 – 1356, 2004. ISSN 0924-0136. doi: http://dx.doi.org/10.1016/j.jmatprotec.2004.04.210. URL http://www.sciencedirect.com/science/article/pii/S0924013604006119.

[21] M.S. Gadala. Recent trends in {ALE} formulation and its applications in solid mechanics. *Computer Methods in Applied Mechanics and Engineering*, 193(39 - 41):4247 – 4275, 2004. ISSN 0045-7825. doi: http://dx.doi.org/10.1016/j.

cma.2004.02.019. URL http://www.sciencedirect.com/science/article/pii/S004578250400221X.

[22] M.R. Vaziri, M. Salimi, and M. Mashayekhi. A new calibration method for ductile fracture models as chip separation criteria in machining. *Simulation Modelling Practice and Theory*, 18(9):1286 – 1296, 2010. ISSN 1569-190X. doi: http://dx.doi.org/10.1016/j.simpat.2010.05.003. URL http://www.sciencedirect.com/science/article/pii/S1569190X10000833.

[23] M.R. Vaziri, M. Salimi, and M. Mashayekhi. Evaluation of chip formation simulation models for material separation in the presence of damage models. *Simulation Modelling Practice and Theory*, 19(2):718 – 733, 2011. ISSN 1569-190X. doi: http://dx.doi.org/10.1016/j.simpat.2010.09.006. URL http://www.sciencedirect.com/science/article/pii/S1569190X10001966.

[24] F. Ducobu, E. Riviére-Lorphévre, and E. Filippi. Influence of the material behavior law and damage value on the results of an orthogonal cutting finite element model of ti6al4v. *Procedia {CIRP}*, 8(0):378 – 383, 2013. ISSN 2212-8271. doi: http://dx.doi.org/10.1016/j.procir.2013.06.120. URL http://www.sciencedirect.com/science/article/pii/S2212827113003971.

[25] W.M. Mohammed, E. Ng, and M.A. Elbestawi. Modeling the effect of the microstructure of compacted graphite iron on chip formation. *International Journal of Machine Tools and Manufacture*, 51(10 - 11):753 – 765, 2011. ISSN 0890-6955. doi: http://dx.doi.org/10.1016/j.ijmachtools.2011.06.005. URL http://www.sciencedirect.com/science/article/pii/S0890695511001180.

[26] S.L. Soo, D.K. Aspinwall, and R.C. Dewes. 3d fe modelling of the cutting of inconel 718. *Journal of Materials Processing Technology*, 150(1 - 2):116 – 123, 2004. ISSN 0924-0136. doi: http://dx.doi.org/10.1016/j.jmatprotec.2004.01.046. URL http://www.sciencedirect.com/science/article/pii/S0924013604000834.

[27] Soehner and Joerg. *Beitrag zur Simulation zerspanungstechnologischer Vorgänge mit Hilfe der Finite Element Methode*. Institut für Produktionstechnik (WBK), 2003. URL http://digbib.ubka.uni-karlsruhe.de/volltexte/2392003.

[28] H. Weseslindtner. *Unterlagen zur Vorlesung "Mechanische Technologie", Teil 1, Spanende Formgebung.*

[29] C. Hortig. *Local and Non-local Thermomechanical Modeling and Finite Element Simulation of High Speed Cutting*. Schriftenreihe des Instituts für Mechanik: Institut für Mechanik. 2010. ISBN 9783921823545. URL http://books.google.at/books?id=TuVumgEACAAJ.

[30] H.-P. Gänser, A. G. Atkins, O. Kolednik, F. D. Fischer, and O. Richard. Upsetting of cylinders: A comparison of two different damage indicators. *Journal of Engineering Materials and Technology*, 123:94 – 99, 2000. doi: 10.1115/1.1286186. URL http://dx.doi.org/10.1115/1.1286186.

Appendix A

Subroutines

A.1 URDFIL and UVARM

Fortran 77 Code for the used Subroutines UVARM and URDFILL.

```
        SUBROUTINE UVARM(UVAR,DIRECT,T,TIME,DTIME,CMNAME,ORNAME,
     1 NUVARM,NOEL,NPT,LAYER,KSPT,KSTEP,KINC,
     2 NDI,NSHR,COORD,JMAC,JMATYP,MATLAYO,LACCFLA)
C
      INCLUDE 'ABA_PARAM.INC'
C
      CHARACTER*80 CMNAME,ORNAME
      CHARACTER*3 FLGRAY(15)
C
      DIMENSION   UVAR(NUVARM),DIRECT(3,3),T(3,3),TIME(2)
      DIMENSION   ARRAY(15),JARRAY(15),JMAC(*),JMATYP(*),COORD(*)
C
      IF (KINC.EQ.1) THEN
      CALL GETVRM('PE',ARRAY,JARRAY,FLGRAY,JRCD,JMAC,JMATYP,
     1 MATLAYO,LACCFLA)
        UVAR(1) = ARRAY(7)
         UVAR(2) = 0
      END IF
      IF (KINC.GT.1) THEN
      CALL GETVRM('PE',ARRAY,JARRAY,FLGRAY,JRCD,JMAC,JMATYP,
     1 MATLAYO,LACCFLA)
          UVAR(2) = ARRAY(7)-UVAR(1)
      END IF
      RETURN
      END
      SUBROUTINE URDFIL(LSTOP,LOVRWRT,KSTEP,KINC,DTIME,TIME)
```

```
C
      INCLUDE 'ABA_PARAM.INC'
C
      DIMENSION ARRAY(513),JRRAY(NPRECD,513),TIME(2)
      EQUIVALENCE (ARRAY(1),JRRAY(1,1))
      PARAMETER(TOL=0.6)
C
C FIND CURRENT INCREMENT.
C
      CALL POSFIL(KSTEP,KINC,ARRAY,JRCD)
      DO K1=1,999999
         CALL DBFILE(0,ARRAY,JRCD)
         IF (JRCD.NE.0) GO TO 110
         KEY=JRRAY(1,2)
C
         IF (KINC.GT.2) THEN
             IF (KEY.EQ.87) THEN
               IF (ARRAY(4).GT.TOL) THEN
                  write(7,*) 'ACHTUNG TOL'
                  LSTOP=1
                  GO TO 110

                 END IF
            END IF
         END IF
      END DO
 110  CONTINUE
C
      RETURN
      END
```

Appendix B

Python Scripts

B.1 Kernel Script

```
#!/usr/bin/env python
"""
This script contains all functions for the common 2D deformation in ABAQUS/
    Standard.

Software compatibility: abaqus 6.121
required modules and/or scripts: xml.dom (build in module)


"""

from odbAccess import *
from abaqus import *
from abaqusConstants import *
import os
import sys
import traceback
from xml.dom.minidom import Document
from pprint import pprint as pp
from operator import itemgetter
import subprocess
import shlex

__author__ = "Stefan Distlberger"
__copyright__ = "Copyright 2013, Materials Center Leoben, MCL"
__credits__ = ["Stefan Distlberger", "Dr. Werner Ecker", "Martin Krobath"]
__license__ = "GPL"
__version__ = "0.309a"
```

```
__maintainer__ = "Stefan Distlberger"
__email__ = "stefan.distlberger@mcl.at"
__status__ = "Production" #"Developement", "Prototype"

global bugMeModus,bugMeModus_L1,bugMeModus_oneLoop
bugMeModus=False # modus to get debugging print outs
bugMeModus_L1=False #L1 modus to debbug
bugMeModus_oneLoop=False
stopCriteria=False #condition to stop the cycle

####################################################

CurCaeCycle=1
MODELNAME=session.sessionState[session.currentViewportName]['modelName']
M=mdb.models[MODELNAME]
A=M.rootAssembly
I=A.instances
session.journalOptions.setValues(recoverGeometry = INDEX) #shows real values instead of the hexxode in the recovery file
session.journalOptions.setValues(replayGeometry = INDEX) #shows real values instead of the hexxode in the replay file
PartList=[] #includes the C2dPart Objects
PartNames={}
C2DStep='step-1' ##USERINPUT, if it is not the only step available
CurCaeAnalysed=False
CurCaeIsInit=False
NextCaeCycle=CurCaeCycle+1
JobName=''
SetList={}
#default orphan geometry properties
importStep = 1 # Step number-1. 0<=Step<=N-1 where N is the number of available steps (Step 0 is the first Step)
deformedShape = DEFORMED # Shape - Possible values are UNDEFORMED and DEFORMED (mostly deformed)
ATV = 0.005 #adjustment Tolerance for the interaction definitions (adjustive slave nodes)
maxCycles=1000 #maximum of all cycles
globalAbstractAngle=15 #abstarction angle for the orphan geometry import
initialCaeName= 'realmatr_5celsius_biggerMesh_Rigid.cae'
VARIATION_COUNTER=0
MAX_VARIATIONS=4
variations=False


def initModel():
  global MODELNAME
  global M
  global A
```

```
  global I
  global C2DStep
  MODELNAME=session.sessionState[session.currentViewportName]['modelName']
  M=mdb.models[MODELNAME]
  A=M.rootAssembly
  I=A.instances
  if len(M.steps)==2:
    C2DStep=M.steps.keys()[-1]


##
### ANALYSE FUNCTIONS ###
##

def analyse():
  '''initial analyse of the model'''
  initModel()
  global PartList
  global PartNames
  global PartNumbers
  global CurCaeAnalysed
  PartList=[]
  PartNames, currentCycle = analyseInstances()
  if PartNames != None:
    for Part in PartNames.keys():
      PartList.append(C2dPart(Part,PartNames[Part]))
    PartNumbers={}
    for i in range(len(PartList)):
      PartNumbers[PartList[i].name]=i
    analyseSurfaces(PartNumbers)
    analyseAnalysisType(PartNumbers)
    analyseInteractions()
    analyseBC(PartNumbers)
    nodeSetToPartList()
  else:
    print 'No part instances found in the current model: %s' % MODELNAME
    return 0
  CurCaeAnalysed=True
  if analyseFiles():
    return 1
  return 1

def analyseInstances():
  '''
  Returnes a Dictionary with the PartNames and their Cycle Version {'Part-1':
    '1', 'Part-2': '101'}.
  It also returns the current cycle number.
  return PartNames, Cycle
```

```
  '''
  global CurCaeIsInit
  global CurCaeCycle
  PartNames={}
  try:
    InstanceNames=[item[0] for item in I.items()]
    if InstanceNames==[]:
      return None, None
  except:
    return None, None
  if bugMeModus_L1: print '-'*10+' INSTANCES, NUMBER '+'-'*10
  for InstanceName in InstanceNames:
    Version=InstanceName.split('-')[-1]
    #InitName=InstanceName[:-len('-'+str(Version))]
    InitName=I[InstanceName].part.name
    if InitName not in PartNames.keys():
      PartNames[InitName]=Version
    elif PartNames[InitName]<Version:
      PartNames[InitName]=Version
    Values=[int(x) for x in set(PartNames.values())]
    Cycle=Values[-1] #last value of the cycle
    if bugMeModus_L1: print InstanceName, Cycle
    if Cycle!=1:
      CurCaeIsInit=False
    if Cycle>CurCaeCycle:
      CurCaeCycle=Cycle
  return PartNames, Cycle

def analyseSurfaces(PartNumbers):
  '''Writes the surface informations into the PartList Object Array '''
  global PartList
  SurfaceNames=A.surfaces.keys()
  check=['edges', 'elements', 'faces', 'nodes']
  for SurfaceName in SurfaceNames:
    for lookType in check:
      try:
        instanceName=A.surfaces[SurfaceName].__getattribute__(lookType)[0].
    instanceName
        PartList[PartNumbers[instanceName]].SurfaceNames.append(SurfaceName)
        break
      except:
        pass

def analyseAnalysisType(PartNumbers):
  '''Writes the analysisType into the PartList Object Array '''
  global PartList
  global I
```

```
  check=['DEFORMABLE_BODY','EULERIAN','DISCRETE_RIGID_SURFACE','ANALYTIC_RIGID_SURFACE']
  checkDeform = ['DEFORMABLE_BODY','EULERIAN']
  for instanceName in I.keys():
    if A.features[instanceName].isSuppressed()==False:
      type=I[instanceName].analysisType
      if type not in check:
        print 'ERROR on analysisType. The type %s is not valid.' % type
        print 'Valid types are: %s' % check
      else:
        PartList[PartNumbers[instanceName]].analysisType = type
  # when a part is not deformable, such as a rigid its deformed state will be set to False
  for Part in PartList:
    try:
      if I[Part.name].analysisType in checkDeform:
        Part.deformed=True
      else:
        Part.deformed=False
    except:
      if bugMeModus_L1: print 'The Part is not a valid instance: %s' %Part.name

def analyseInteractions():
  '''Find interactions and reference the used parts to the interaction depending on their type: slave, master'''
  global PartList
  global M
  check=[SURFACE_TO_SURFACE]
  for interName in M.interactions.keys():
    interaction= M.interactions[interName]
    if interaction.enforcement in check and interaction.__doc__[:4]!='Self':
      master = interaction.master[0]
      slave = interaction.slave[0]
      for Instance in PartList:
        if master in Instance.SurfaceNames:
          Instance.Contact.append([interName, 'master', interaction.enforcement])
        elif slave in Instance.SurfaceNames:
          Instance.Contact.append([interName, 'slave', interaction.enforcement])
    if interaction.__doc__[:4]=='Self':
      master = interaction.surface[0]
      slave = interaction.surface[0]
      for Instance in PartList:
        if master in Instance.SurfaceNames:
          Instance.Contact.append([interName, 'master', interaction.enforcement])
        elif slave in Instance.SurfaceNames:
          Instance.Contact.append([interName, 'slave', interaction.enforcement])

def analyseBC(PartNumbers):
```

```
  '''
  Analyses the BCs of the Model and writes the found values into the PartList Object Array.
  '''
  global PartList
  global M
  global C2DStep
  BCNames =  M.boundaryConditions.keys()
  check=['nodes','cells','edges', 'elements', 'faces','referencePoints','vertices']
  for BCName in BCNames:
    for lookType in check:
      try:
        region=M.boundaryConditions[BCName].region[0]
        if lookType=='referencePoints':
          instanceName=A.sets[region].__getattribute__(lookType)[0].__repr__().split('\'')[-2]
        else:
          instanceName=A.sets[region].__getattribute__(lookType)[0].instanceName
        states=M.steps[C2DStep].boundaryConditionStates[BCName]
        u=[states.u1,states.u2,states.u3,states.ur1,states.ur2,states.ur3]
        PartList[PartNumbers[instanceName]].boundaryConditions.append([region,BCName,u,states.amplitude])
        break
      except:
        pass

def analyseMovement():
  '''
  Analyses the already moved parts boundary conditions and stores the displacement-values into the PartList array
  '''
  global I
  global A
  initValues={}
  for Part in PartList:
    for BC in range(len(Part.initialBoundaryConditions)):
      try:
        if Part.initialBoundaryConditions[BC][0]!=Part.boundaryConditions[BC][1]:
          if Part.initialBoundaryConditions[BC][1]!=Part.boundaryConditions[BC][2]:
            init=Part.initialBoundaryConditions[BC][1]
            current=Part.boundaryConditions[BC][2]
            name=Part.initialBoundaryConditions[BC][0]
            Part.BCsDiff.append([name,[current[0]-init[0],current[1]-init[1],current[2]-init[2],current[3]-init[3],current[4]-init[4],current[5]-init[5]]])
      except:
```

```
        if bugMeModus_L1: ' Analyse Movement Error:'+repr(Part.name)
        pass

def analyseFiles():
  '''
  Analyses all the files needed to start the simulation.
  '''
  global CurCaeCycle
  global CurCaeIsInit
  if analyseJob():
    curCycle=int(mdb.jobs.keys()[0].split('-')[-1])
    for JobName in mdb.jobs.keys():
      num=int(JobName.split('-')[-1])
      if curCycle<num:
        curCycle=num
    if CurCaeIsInit==False:
      if curCycle!=CurCaeCycle:
        return False
    curJOB=mdb.jobs.keys()[0].split('-')[0]+'-'+str(curCycle)
    if curJOB==1:
      curJOB=CurCaeCycle
    if not os.path.exists(curJOB+'.odb'):
      logging.error('File %s.odb not found' %curJOB)
      print 'ERROR - no initial Job file found in this directory!'
      return False
    return True
  ## here would be a good place to implement further analyse procedures, which
  ## can guaranty the integrity of the next simulation cycle, such as the appearance of odb.lck files aso...

def analyseJob():
  '''
  Returns a True if the JobNames are all the same and just the cycle numbers are different.
  '''
  global JobName
  oldJob=mdb.jobs.keys()[0].rstrip('-'+mdb.jobs.keys()[0].split('-')[-1])
  for Name in mdb.jobs.keys():
    newJob=Name.rstrip('-'+Name.split('-')[-1])
    if oldJob!=newJob:
      return False
    oldJob=newJob
  JobName=oldJob
  return True

##
### CLASSES ###
##
```

```
class C2dPart:
  '''C2dPart is a object including all informations of a part, such as BC, interactions, remeshing, aso.'''
  def __init__(self,name,cycle,remeshing=False, *p,**kw):
    self.remesh=remeshing
    self.cleanName=name
    self.name=name+'-'+str(cycle)
    self.initialName=name+'-1'
    if cycle>1:
      self.deformed=True
    else:
      self.deformed=False
    self.SurfaceNames=[]
    self.Contact=[] #2 dim array with interaction names, types and enforcements e.g. [['Contact_MP-wp', 'slave', SURFACE_TO_SURFACE],]
    self.boundaryConditions=[]
    self.initialBoundaryConditions=[]
    self.BCsDiff=[] #shows the difference of the current bcs values in dependence to its initial values
    self.abstractAngle=10
    self.nodeSets=[]

  def __name__(self):
    '''returnes the name of the instance'''
    return self.name
  def bc_info(self):
    print self.boundaryConditions

#class C2dModel:
# ''' C2d Model includes informations such as interactions, BC, aso.'''
# def __init__(self,name, *p,**kw):
#   self.interactions=[]

class MeshWindow:
  '''
  Mesh Window Object includes all the informations about the Meshwindows on an Instance
  name = name of the meshwindow object
  instanceName= Name of the instanceto be meshed
  GS = global seed size
  During the creation of mesh windows, the given Windows will be sorted by size. Therefore, a smaller window
  will be meshed after a bigger one.
  '''
  global A,I,M
  def __init__(self,name,instanceName,GS,Var,VarCount, *p,**kw):
    CR1X1=0
```

```
        CR1X2=0
        CR2Y1=0
        CR2Y2=0
        CR3X1=0
        CR3X2=0
        CR4Y1=0
        CR4Y2=0
        self.name=name
        self.instanceName=instanceName
        self.MeshWindows=[]
        self.objects=0
        self.tolerance=0.001
        self.GS=GS
        self.new=True
        self.Partitions=[]
        self.variations=Var
        self.VarCount=VarCount

    def __name__(self):
        '''returnes the name of the object'''
        return self.name

    def critArea(self,x1,y1,x2,y2,SG):
        global CR1X1
        global CR1X2
        global CR2Y2
        global CR2Y1
        global CR3X1
        global CR3X2
        global CR4Y2
        global CR4Y1
        CR4Y1=y2-SG
        CR1X1=x1-SG
        CR1X2=x1+SG
        CR2Y2=y1+SG
        CR2Y1=y1-SG
        CR3X1=x2-SG
        CR3X2=x2+SG
        CR4Y2=y2+SG

    def crit(self,instanceName):
        '''
        To ensure that during the remeshprocess no vertex is bodering the calculation
        this skript calculates the optimal position of the meshwindow in the of the
        user specified tolerance!
        looks for the vertices and meshwindowpositions. it compares the positions to
        ensure that meshing problems wont accur.
        '''
```

```
Vert=[]
Vert=I[instanceName].vertices
VertLen=len(Vert)
xVal=[]
yVal=[]
CritVal1=[]
CritVal2=[]
CritVal3=[]
CritVal4=[]
retVal={}
for i in range(VertLen):
  xVal.append(Vert[i].pointOn[0][0])
  yVal.append(Vert[i].pointOn[0][1])
print xVal
print yVal
for i in range(VertLen):
  if (CR1X1 < xVal[i]) & (CR1X2 > xVal[i]) & (CR4Y1 < yVal[i]) & (CR2Y2 > yVal[i]):
    CritVal1.append(Vert[i])
  if (CR2Y1 < yVal[i]) & (CR2Y2 > yVal[i]) & (CR1X1 < xVal[i]) & (CR3X2 > xVal[i]):
    CritVal2.append(Vert[i])
  if (CR3X1 < xVal[i]) & (CR3X2 > xVal[i]) & (CR4Y1 < yVal[i]) & (CR2Y2 > yVal[i]):
    CritVal3.append(Vert[i])
  if (CR4Y1 < yVal[i]) & (CR4Y2 > yVal[i]) & (CR1X1 < xVal[i]) & (CR3X2 > xVal[i]):
    CritVal4.append(Vert[i])
if len(CritVal1)>1:
  print "adaption of Area1"
  VAL1=self.calcCritVal(CritVal1,CR1X1,CR1X2,"x")
  retVal['x1']=VAL1
  print "adapted Value: x1= "+repr(VAL1)
if len(CritVal2)>1:
  print "adaption of Area2"
  VAL2=self.calcCritVal(CritVal2,CR2Y1,CR2Y2,"y")
  retVal['y1']=VAL2
  print "adapted Value: y1= "+repr(VAL2)
if len(CritVal3)>1:
  print "adaption of Area3"
  VAL3=self.calcCritVal(CritVal3,CR3X1,CR3X2,"x")
  retVal['x2']=VAL3
  print "adapted Value: x2= "+repr(VAL3)
if len(CritVal4)>1:
  print "adaption of Area4"
  VAL4=self.calcCritVal(CritVal4,CR4Y1,CR4Y2,"y")
  retVal['y2']=VAL4
  print "adapted Value: y2= "+repr(VAL4)
```

```
    return retVal

  def calcCritVal(self,CritVal,EdgeCoordMIN,EdgeCoordMAX,Coord):
    from operator import itemgetter

    if (Coord=="x"):
      coord=0
    else:
      coord=1
    CritVal.sort()
    CritValLen=len(CritVal)
    Solution={} #Dictionary for the combination of Value and Index
    CalcDict={}
    OutputVal=0
    if CritValLen>=2:
      print "two or more critical vertices found --- starting adaption"
      for i in range(CritValLen):
        Solution[CritVal[i].index]=CritVal[i].pointOn[0][coord]
        X=sorted(Solution.items(), key=itemgetter(1))
      print X
      for i in range(CritValLen-1):
        a=X[i+1][1]-X[i][1]
        b=X[i][1]+((X[i+1][1]-X[i][1])/2)
        CalcDict[a]=b
      X=sorted(CalcDict.items(), reverse=True)
      print X
      if X[0][0]==0: # if two points are exactly above each other they will be handled as one point.
        CritValLen=1
        print "merging points"
      else:
        return X[0][0]
    if CritValLen==1:
      print "one critical vertex found --- starting adaption"
      if (CritVal[0].pointOn[0][coord]-EdgeCoordMIN) > (EdgeCoordMAX-CritVal[0].pointOn[0][coord]):
        OutputVal=EdgeCoordMIN + ((CritVal[0].pointOn[0][coord]-EdgeCoordMIN)/2)
        return OutputVal
      else:
        OutputVal=EdgeCoordMAX - ((EdgeCoordMAX-CritVal[0].pointOn[0][coord])/2)
        return OutputVal
    if CritValLen==0:
      print "no critical vertices found --- no adaption"
      OutputVal=EdgeCoordMIN+(EdgeCoordMAX-EdgeCoordMIN)
    return OutputVal

  def calcRelCoord(self,coords,direction):
    '''
```

```
    calculates the relative offset to a given Node of another instance.
    '''
    global A,I
    relativeInstance=I[coords[0]]
    NodeIndex=coords[1]-1
    offset=coords[2]
    if direction%2==0:
      xrel=relativeInstance.nodes[NodeIndex].coordinates[0]
      newCoord=xrel+offset
    else:
      yrel=relativeInstance.nodes[NodeIndex].coordinates[1]
      newCoord=yrel+offset
    return newCoord


  def window(self,X1,Y1,X2,Y2,SEED,store={}):
    '''
    The coordinates can be set as relative coordinate or as static coordinate.
    To set a static coordinate, the input has to be a float
    To set a relative coordinate, a tuple has to be the input parameter.
    exanple:
    static: X1=2.3
    dynamic: X1=('Part-1-2',345,5.2), where 'Part-1-2' is the relative Part (not remeshable),
    345 is the Label of the Node you want refer to and 5.2 is the offset distance in x-direction from that Node.
    '''
    #this slightly variates the mesh to gain convergence in certain cases.
    if self.variations==True:
      if self.VarCount%2==0:
        SEED=SEED*0.02*(0.5*self.VarCount)+SEED
      else:
        SEED=SEED-SEED*0.02*(0.5*self.VarCount)
    coords=[X1,Y1,X2,Y2]
    for i in range(len(coords)):
      if type(coords[i]).__name__ == 'tuple':
        coords[i]=self.calcRelCoord(coords[i],i)
    if self.new:
      store['counter'] = 0
      self.new=False
    if isinstance(store, dict): #checks if the dict store is allready stored
      store['counter'] = store.get('counter',0) + 1 #increase the counter by one
    X1=coords[0]
    Y1=coords[1]
    X2=coords[2]
    Y2=coords[3]
    self.critArea(X1,Y1,X2,Y2,SEED)
    adap=self.crit(self.instanceName)
```

```
    try:
      X1=adap['x1']
      X2=adap['x2']
      Y1=adap['y1']
      Y2=adap['y2']
    except:
      print "adaption of some directions not needed"
    Id = 'id'
    WindowObject={}
    WindowObject[Id]=store.get('counter',0)
    self.objects=store.get('counter',0)
    WindowObject['pointOn2']=((X2,Y2,0.),)
    WindowObject['pointOn1']=((X1,Y1,0.),)
    xc=((X1+X2)/2)
    yc=((Y1+Y2)/2)
    deltax=X2-X1
    deltay=Y2-Y1
    Area=abs(deltax)*abs(deltay)
    WindowObject['center']=((xc,yc,0.),)
    WindowObject['size']=Area
    WindowObject['seed']=SEED
    self.MeshWindows.append(WindowObject)

  def delMW(self):
    try:
      self.Partitions.reverse()
      for partitionname in self.Partitions:
        del A.features[partitionname]
    except:
      print 'not all partitions were deleted'
    del self

  def applyMW(self):
    '''
    the following lines generate the mesh.
    If something isn't working on the meshes, mostly the positions or geometry
    are responsible
    '''
    from pprint import pprint as pp
    global A,I,M
    instance=I[self.instanceName]
    for partitionname in self.Partitions:
      try:
        del A.features[partitionname]
      except:
        pass
    self.Partitions=[]
```

```
    for i in range(self.objects):
        try:
          mySketch = M.ConstrainedSketch(name='mySketch',sheetSize=200)
          mySketch.rectangle(point1=(self.MeshWindows[i]['pointOn1'][0][0],self.MeshWindows[i]['pointOn1'][0][1]),point2=(self.MeshWindows[i]['pointOn2'][0][0],self.MeshWindows[i]['pointOn2'][0][1]))
          A.PartitionFaceBySketch(faces=(instance.faces), sketch=mySketch)
          self.Partitions.append(A.features[A.features.keys()[-1]].name)
          del M.sketches['mySketch']
        finally:
          print 'partition created: MW '+repr(i)

    A.seedPartInstance(deviationFactor=0.1, regions=(instance, ), size=self.GS)
    tempArray=[]
    for i in range(self.objects-1):
        if self.MeshWindows[i]['size']<self.MeshWindows[i+1]['size']:
            tempArray=self.MeshWindows[i]
            self.MeshWindows[i]=self.MeshWindows[i+1]
            self.MeshWindows[i+1]=tempArray

    for i in range(self.objects):
        SIZE=self.MeshWindows[i]['seed']
        for j in range(len(instance.edges)):
            if (instance.edges[j].pointOn[0][0]>=self.MeshWindows[i]['pointOn1'][0][0]-self.tolerance) and (instance.edges[j].pointOn[0][1]>=self.MeshWindows[i]['pointOn2'][0][1]-self.tolerance) and (instance.edges[j].pointOn[0][0]<=self.MeshWindows[i]['pointOn2'][0][0]+self.tolerance) and (instance.edges[j].pointOn[0][1]<=self.MeshWindows[i]['pointOn1'][0][1]+self.tolerance):
              A.seedEdgeBySize(edges=(instance.edges[j],), size=SIZE)
    #self.MeshWindows.append(self.GS)

    for i in range(len(instance.faces)):
      A.setMeshControls(regions=instance.faces.findAt(instance.faces[i].pointOn), technique=FREE, elemShape=QUAD)
    A.generateMesh(regions=(instance, ))
    #pp(self.MeshWindows)



##
### REBUILD FUNCTIONS ###
##

def edgeSet(pos, PName,BCName, TYPE='horizontal'):
  '''
  creating a set, using the geometry of the part.
  pos = position in x or y direction (float)
```

```
  Pname = name of the part instance
  BCName = name of the set
  TYPE = keyword defining the x or y coordinate of pos
    valid types: 'horizontal','vertical'
  example: >>> edgeSet(0.0,'WORKPIECE-1-102','__a','vertical')
  '''
  global I
  global A
  valids=[]
  edges=I[PName].edges
  if TYPE=='horizontal':
    for i in range(len(edges)):
      if edges[i].pointOn[0][1] == pos:
        valids.append(edges.findAt(edges[i].pointOn))
  elif TYPE=='vertical':
    for i in range(len(edges)):
      if edges[i].pointOn[0][0] == pos:
        valids.append(edges.findAt(edges[i].pointOn))
  else:
    print 'unknown orientation: %s' %TYPE
  A.Set(edges=valids, name=BCName)

def setFromMaxSideGeom(Pname,SetName,POS,posVal=None,type='node',accuracy=0):
  '''
  looks for nodes or edges and the highest value available.
  Pname = name of the part instance
  SetName = name of the set
  POS = keyword defining the position.
    possible values are: right, left, top, bottom
  posVal = position where the function should look.
  type = specifies the type of entities. node or edge
  accuracy = value wich specifies the aberration to its base value, like 10 +/-
    0.001. (accuracy = 0.001)
  '''
  global I
  global A
  typeValids= ['node','edge']
  valids = ['right','left','top','bottom']
  if POS not in valids:
    print 'position invalid: %s , possible values: right, left, top, bottom' %
    Pname
    return None
  elif Pname not in I.keys():
    print 'partname not found: %s' %Pname
    return None
  else:
    if len(I[Pname].edges)==0 or type=='node':
      print 'searching for nodes in part %s' %Pname
```

```
    n=I[Pname].nodes
    coords=[coord.coordinates for coord in n]
    xCoords=[a[0] for a in coords]
    yCoords=[a[1] for a in coords]
    labels=[]
    if POS=='right':
      maxX=max(xCoords)
      if posVal!=None:
        maxX=posVal
      for node in n:
        if node.coordinates[0]<=maxX+accuracy and node.coordinates[0]>=maxX-
  accuracy:
          labels.append(node.label)
    elif POS=='left':
      minX=min(xCoords)
      if posVal!=None:
        minX=posVal
      for node in n:
        if node.coordinates[0]<=minX+accuracy and node.coordinates[0]>=minX-
  accuracy:
          labels.append(node.label)
    elif POS=='top':
      maxY=max(yCoords)
      if posVal!=None:
        maxY=posVal
      for node in n:
        if node.coordinates[1]<=maxY+accuracy and node.coordinates[1]>=maxY-
  accuracy:
          labels.append(node.label)
    elif POS=='bottom':
      minY=min(yCoords)
      if posVal!=None:
        minY=posVal
      for node in n:
        if node.coordinates[1]<=minY+accuracy and node.coordinates[1]>=minY-
  accuracy:
          labels.append(node.label)
    nodeL=n[labels[0]-1:labels[0]]
    labels.pop(0)
    for label in labels:
      nodeL+=n[label-1:label]
    A.Set(nodes=nodeL,name=SetName)
  else:
    print 'searching for edges in part %s' %Pname
    e=I[Pname].edges
    xCoords=[x[0][0] for x in e.pointsOn]
    yCoords=[x[0][1] for x in e.pointsOn]
    indizes=[]
```

```
      if POS=='right':
        maxX=max(xCoords)
        if posVal!=None:
          maxX=posVal
        for edge in e:
          if edge.pointOn[0][0]<=maxX+accuracy and edge.pointOn[0][0]>=maxX-accuracy:
            indizes.append(edge.index)
      elif POS=='left':
        minX=min(xCoords)
        if posVal!=None:
          minX=posVal
        for edge in e:
          if edge.pointOn[0][0]<=minX+accuracy and edge.pointOn[0][0]>=minX-accuracy:
            indizes.append(edge.index)
      elif POS=='top':
        maxY=max(yCoords)
        if posVal!=None:
          maxY=posVal
        for edge in e:
          if edge.pointOn[0][1]<=maxY+accuracy and edge.pointOn[0][1]>=maxY-accuracy:
            indizes.append(edge.index)
      elif POS=='bottom':
        minY=min(yCoords)
        if posVal!=None:
          minY=posVal
        for edge in e:
          if edge.pointOn[0][1]<=minY+accuracy and edge.pointOn[0][1]>=minY-accuracy:
            indizes.append(edge.index)
      edgeL=e[indizes[0]:indizes[0]+1]
      indizes.pop(0)
      for index in indizes:
        edgeL+=e[index:index+1]
      A.Set(edges=edgeL,name=SetName)

'''def reconParts(PartList, StepName, JobName, importStep, CurCaeCycle):
  odbName = JobName+'-'+repr(CurCaeCycle)+'.odb'
  for Part in PartList:
    if Part.deformed:
      orphanInstance = Part.cleanName.upper()+'-'+str(CurCaeCycle)
      orphanName = Part.cleanName+'-'+str(CurCaeCycle+1)
      if Part.remesh:
        geoName = Part.cleanName+'-'+str(CurCaeCycle+1)
        orphanImportGeo(odbName, orphanInstance, DEFORMED, importStep, orphanName, StepName, Part.abstractAngle, geoName)
```

```
      else:
        orphanImport(odbName, orphanInstance, DEFORMED, importStep, orphanName,
    StepName)
      makeInstance(orphanName,orphanName)
      A.suppressFeatures((Part.cleanName+'-'+str(CurCaeCycle),))
    else:
      try:
        A.features.changeKey(fromName=Part.cleanName+'-'+str(CurCaeCycle),toName=
    Part.cleanName+'-'+str(CurCaeCycle+1))
      except:
        A.features.changeKey(fromName=Part.cleanName+'-1',toName=Part.cleanName
    +'-'+str(CurCaeCycle+1))
'''
def reconParts(PartList, StepName, JobName, importStep, CurCaeCycle):
  odbName = JobName+'-'+repr(CurCaeCycle)+'.odb'
  #staName = JobName+'-'+repr(CurCaeCycle)+'.sta'
  #if os.path.exists(odbName)==False or os.path.exists(staName)==False:
  # print 'ERROR: Job not found or sta file missing: %s' %odbName
  # return 0

  for Part in PartList:
    if Part.deformed:
      orphanInstance = Part.cleanName.upper()+'-'+str(CurCaeCycle)
      orphanName = Part.cleanName+'-'+str(CurCaeCycle+1)
      if Part.remesh:
        geoName = Part.cleanName+'-'+str(CurCaeCycle+1)
        orphanImportGeo(odbName, orphanInstance, DEFORMED, importStep, orphanName
    , StepName, Part.abstractAngle, geoName)
      else:
        orphanImport(odbName, orphanInstance, DEFORMED, importStep, orphanName,
    StepName)
      makeInstance(orphanName,orphanName)
      A.suppressFeatures((Part.cleanName+'-'+str(CurCaeCycle),))
    else:
      try:
        A.features.changeKey(fromName=Part.cleanName+'-'+str(CurCaeCycle),toName=
    Part.cleanName+'-'+str(CurCaeCycle+1))
        odb=openOdb(odbName,readOnly=TRUE)
        val=odb.steps[odb.steps.keys()[importStep]].frames[-1].fieldOutputs['U'].
    values

        try:
          data=[i.dataDouble for i in val if i.instance.name==Part.cleanName.
    upper()+'-'+str(CurCaeCycle)]
          print Part.cleanName.upper()+'-'+str(CurCaeCycle+1)
        except:
          data=[i.data for i in val if i.instance.name==Part.cleanName.upper()+'-
    '+str(CurCaeCycle)]
```

```
        print data
        try:
          A.translate(instanceList=(Part.cleanName+'-'+str(CurCaeCycle+1),), vector=(data[0][0],data[0][1],0.0))
        except:
          print 'ERROR: no translation data found in old odb data for instance %s' %Part.cleanName.upper()+'-'+str(CurCaeCycle+1)
        print '*'*10
      except:
        A.features.changeKey(fromName=Part.cleanName+'-1',toName=Part.cleanName+'-'+str(CurCaeCycle+1))
  return 1

def orphanImport(odbName, orphanInstance, deformedShape, importStep, orphanName, stepName):
  '''imports a parts orphan mesh'''
  for t in range(5):
    try:
      odb= openOdb(path=odbName,readOnly=True)
    except:
      time.sleep(2)
  lastFrame=odb.steps[stepName].frames[-1]
  importframe = lastFrame.frameId
  #orphan = M.PartFromOdb(fileName=odbName,name=orphanName,instance=orphanInstance,shape=deformedShape,step=importStep,frame=importframe)
  orphan = M.PartFromOdb(fileName=odbName,name=orphanName,instance=orphanInstance,shape=deformedShape,step=importStep)

def orphanImportGeo(odbName, orphanInstance, deformedShape, importStep, orphanName, stepName, angle, geoName):
  '''imports the geometry of a orphan mesh'''
  for t in range(5):
    try:
      odb= openOdb(path=odbName,readOnly=True)
    except:
      time.sleep(2)
  lastFrame=odb.steps[stepName].frames[-1]
  importframe = lastFrame.frameId
  #orphan = M.PartFromOdb(fileName=odbName,name=orphanName,instance=orphanInstance,shape=deformedShape,step=importStep,frame=importframe)
  orphan = M.PartFromOdb(fileName=odbName,name=orphanName,instance=orphanInstance,shape=deformedShape,step=importStep)
  geo = M.Part2DGeomFrom2DMesh(name=geoName,part=orphan,featureAngle=angle)

def makeInstance(instName, partName):
  '''makes an instance out of a existing part'''
  A.Instance(dependent=OFF, name=instName, part=M.parts[partName])

```

```
def checkSuppressState():
  for iPart in PartList:
    try:
      if A.features[iPart.initialName].isSuppressed() == False:
        A.features[iPart.initialName].suppress()
    except:
      pass

def getSideEdge(Instance,SurfaceName):
  '''creates a surrounding surface on an orphan mesh'''
  global A,I
  edges=I[Instance].elementEdges
  cedges=[x for x in edges if not len(x.getElements())>1]
  side1=[]
  side2=[]
  side3=[]
  side4=[]
  for i in range(len(cedges)):
    n1=cedges[i].getNodes()[0].label-1
    n2=cedges[i].getNodes()[1].label-1
    con=cedges[i].getElements()[0].connectivity
    if (n1==con[0] or n2==con[0]) and (n1==con[1] or n2==con[1]):
      side1.append(cedges[i].getElements()[0].label)
    elif (n1==con[1] or n2==con[1]) and (n1==con[2] or n2==con[2]):
      side2.append(cedges[i].getElements()[0].label)
    elif (n1==con[2] or n2==con[2]) and (n1==con[3] or n2==con[3]):
      side3.append(cedges[i].getElements()[0].label)
    elif (n1==con[3] or n2==con[3]) and (n1==con[0] or n2==con[0]):
      side4.append(cedges[i].getElements()[0].label)
  el=I[Instance].elements
  s1=el.sequenceFromLabels(side1)
  s2=el.sequenceFromLabels(side2)
  s3=el.sequenceFromLabels(side3)
  s4=el.sequenceFromLabels(side4)
  A.Surface(face1Elements=s1, face2Elements=s2, face3Elements=s3, face4Elements=
    s4, name=SurfaceName)

def assignSections():
  global M,I,A
  for part in PartList:
    name=part.cleanName+'-'+str(CurCaeCycle+1)
    if part.sectionName != 'None':
      if part.remesh == False:
        M.parts[name].SectionAssignment(offset=0.0,offsetField='',offsetType=
    MIDDLE_SURFACE, region=Region(elements=M.parts[name].elements),sectionName=
    part.sectionName)
      else:
```

```
        M.parts[name].SectionAssignment(thicknessAssignment=FROM_GEOMETRY, region=Region(faces=M.parts[name].faces),sectionName=part.sectionName)

def createSetbySetName(PartList):
  '''
  Creates new Sets which consists out of the same names.
  The important part is the Keyword Part at the end of the set name.
  E.g. If the Setname ends with __L, an edge will be searched on the lower side of the instance. (meshed)
  If there is a keyword like _PRB, the right bottom point will be set as a new set.
  '''
  global A,I

  validEdge=['__L','__R','__B','__T']
  validPoint=['_PRB','_PRT','_PLB','_PLT']

  for Part in PartList:
    print 'Instance: '+Part.name
    PName=Part.cleanName+'-'+str(NextCaeCycle)
    #Instance=I[Part.cleanName+'-'+repr(CurCaeCycle)]
    for nSet in Part.nodeSets:
      if nSet[0][-3:].upper() in validEdge:
        POS=''
        if nSet[0][-3:].upper() == '__L':
          POS='left'
        if nSet[0][-3:].upper() == '__R':
          POS='right'
        if nSet[0][-3:].upper() == '__B':
          POS='bottom'
        if nSet[0][-3:].upper() == '__T':
          POS='top'
        print 'position: '+POS
        setFromMaxSideGeom(PName,nSet[0],POS,accuracy=0.001)
      elif nSet[0][-4:].upper() in validPoint:
        if nSet[0][-2:].upper() == 'RB':
          n=findNodeMin('RB',PName)
          A.SetFromNodeLabels(name=nSet[0],nodeLabels=((PName,(n,)),))
        elif nSet[0][-2:].upper() == 'RT':
          n=findNodeMin('RT',PName)
          A.SetFromNodeLabels(name=nSet[0],nodeLabels=((PName,(n,)),))
        elif nSet[0][-2:].upper() == 'LB':
          n=findNodeMin('LB',PName)
          A.SetFromNodeLabels(name=nSet[0],nodeLabels=((PName,(n,)),))
        elif nSet[0][-2:].upper() == 'LT':
          n=findNodeMin('LT',PName)
          A.SetFromNodeLabels(name=nSet[0],nodeLabels=((PName,(n,)),))
        print 'Set: '+nSet[0]
```

```
      elif Part.remesh==False and Part.deformed==True:
        A.SetFromNodeLabels(nodeLabels=((PName,tuple(nSet[1])),),name=nSet[0])
        print 'Set: '+nSet[0]

def findNodeMin(POS,Instance):
  '''
  Finds the node at a given position and returnes a label.
  valid POS values are: 'RB', 'RT', 'LB', 'LT'
  '''
  global A,I
  n=I[Instance].nodes
  tempNode=n[0]
  for node in n:
    if POS=='RB':
      if tempNode.coordinates[0]<=node.coordinates[0] and tempNode.coordinates[1]>=node.coordinates[1]: tempNode=node
    if POS=='RT':
      if tempNode.coordinates[0]<=node.coordinates[0] and tempNode.coordinates[1]<=node.coordinates[1]: tempNode=node
    if POS=='LB':
      if tempNode.coordinates[0]>=node.coordinates[0] and tempNode.coordinates[1]>=node.coordinates[1]: tempNode=node
    if POS=='LT':
      if tempNode.coordinates[0]>=node.coordinates[0] and tempNode.coordinates[1]<=node.coordinates[1]: tempNode=node
  return tempNode.label



##
### INITIAL MODEL ANALYSE
##

def analyseInitCAE(initialName, overwrite=True, noPrompt=False):
  import xml.dom
  from abaqus import mdb
  global CurDir
  global M,I,A,P
  global PartList
  global PartNumbers
  global SetList
  global CurCaeAnalysed, CurCaeIsInit

  CurPath = mdb.pathName
  CurDir=os.getcwd()
  if CurPath=='<unnamed>':
    print 'WARNING: no cae file found in current session'
  elif CurCaeIsInit==True:
```

```
    print 'WARNING: the current cae file is the initial cae file'
  elif CurCaeAnalysed==False:
    print 'WARNING: the current cae file has to be analysed before the initial cae file (Analyse())'
  else:
    filepath=initialName+'.xml'
    if not os.path.isfile(filepath):
      getInitValues(initialName, overwrite=overwrite, noPrompt=noPrompt)
    elif overwrite==True:
      getInitValues(initialName, overwrite=overwrite, noPrompt=noPrompt)
    try:
      data=readXMLDoc(filepath)
    except:
      print 'ERROR: The file %s.xml was not found.' %initialName
      return None

    if bugMeModus: print CurPath
    mdb = openMdb(CurPath)

    M=mdb.models[session.sessionState[session.currentViewportName]['modelName']]
    A=M.rootAssembly
    I=A.instances
    P=M.parts
    #reading the current BCs and look for the values of the initial BCS and save them to
    #the initialBoundaryConditions Array of the PartList bbject
    for instance in I.keys():
      if not A.features[instance].isSuppressed():
        if instance.split('-')[-1] != '1':
          try:
            Object=PartList[int(PartNumbers[instance.rstrip(instance.split('-')[-1])+'1'])]
          except:
            Object=PartList[int(PartNumbers[instance])]
        else:
          Object=PartList[int(PartNumbers[instance])]
        Object.initialBoundaryConditions=[]
        try:
          if bugMeModus: print Object.name
          attr=data.getElementsByTagName(Object.initialName)[0]
          for i in range(len(attr._get_attributes().keys())):
            if str(attr._get_attributes().keys()[i]) != 'id' and str(attr._get_attributes().keys()[i]) != "SECTIONNAME":
              u=eval(attr.getAttribute(str(attr._get_attributes().keys()[i])))
              if bugMeModus: print str(attr._get_attributes().keys()[i]),u
              Object.initialBoundaryConditions.append([str(attr._get_attributes().keys()[i]),u])
            if str(attr._get_attributes().keys()[i]) == "SECTIONNAME":
```

```
              Object.sectionName=str(attr._get_attributes()[str(attr._get_attributes().keys()[i])].value)
        except:
          pass
def analyseSetFromInitOdb(odbName):
  '''
  analyses the first odb, to recieve information about the sets and their parts.
  This is also only possible, if the CAE file is analysed and the odb to be analysed is the initial odb.
  The initial odb has to end with the litteral 1!
  VALID positions for a point are: 'Setname'+'_P'+'R , L'+'T, B' ...
  VALID positions for an edge are: 'Setname+'__L , R , B, T' ...
  P...Point
  R...Right
  L...Left
  T...Top
  B...Bottom
  The double underlining for edgesets is the indicator for an edge set.
  The _P indicates a Point.
  This must be defined in the initial CAE File
  '''
  if CurCaeAnalysed==True and odbName[-1]=='1':
    try:
      odb=openOdb(path=odbName+'.odb',readOnly=TRUE)
    except:
      return 'path %s not found' % odbName
    odbA = odb.rootAssembly
    odbNS = odbA.nodeSets
    for SET in odbNS.keys():
      if len(odbNS[SET].instances)==1: #only Sets refering to just one instance will be analysed
        PNAME=odbNS[SET].instances[0].__repr__().split('\'')[-2] #name of the part
        #SNAME=odbNS[SET].__repr__().split('\'')[-2] #name of the set
        SNAME=SET
        ListPos=[PartNumbers[i] for i in PartNumbers.keys() if i.upper()==PNAME or i==PNAME][0]
        print PNAME, ListPos
        if PartList[ListPos].deformed==True and PartList[ListPos].remesh==False:
          SetList[SNAME]=['None',PNAME,odbNS[SET].nodes] #['orientation,partname,nodes]
        if PartList[ListPos].deformed==True and PartList[ListPos].remesh==True:
          if SNAME[-3:-1]=='__':
            SetList[SNAME]=[SNAME[-1],PNAME,odbNS[SET].nodes]
          elif (SNAME[-4]=='_') and (SNAME[-3]=='P'):
            SetList[SNAME]=[SNAME[-3:],PNAME,odbNS[SET].nodes]
          else: print 'INVALID SET NAME for remeshable part %s' % PNAME
  else: 'The CAE has to be analysed first in order to get its Set information'
```

```

def readXMLDoc(filepath):
  '''
  reads the XML file given in the form "filepath=C://temp//xml.xml"
  and returns it as a minidom object
  '''
  import xml.dom.minidom as mini
  if bugMeModus: print 'reading xml file: '+str(traceback.extract_stack()[-1][1])
  return mini.parse(filepath)

def writeXML(doc, filename):
  '''
  writes the XML file with toprettyxml function
  xml.dom.ext.PrettyPrint(doc, open(filename, "w"))
  '''
  print doc.toprettyxml(indent="")
  text = doc.toprettyxml(indent="", encoding='UTF-8')
  try:
    print '%s' %filename
    projectFile = file(filename, 'w')
  except:
    projectFile = open(filename, 'w')
  projectFile.write(text)
  projectFile.close()

def user_isSuppressed(noPrompt=False):
  '''
  looks for the instances in the current cae file and if there are some suppressesd features,
  it will raise a prompt. The user is than able to either resume all features or cancle the procedure and resume
  just the neccessary features.
  '''
  from abaqus import getWarningReply, YES, NO
  MODELNAME=session.sessionState[session.currentViewportName]['modelName']
  M=mdb.models[MODELNAME]
  A=M.rootAssembly
  I=A.instances

  if bugMeModus: A.featurelistInfo()
  for InstanceName in I.keys():
    reply=NO
    if A.features[InstanceName].isSuppressed():
      if not noPrompt:
        reply = getWarningReply(message=str(InstanceName)+' is suppressed. Do you want to resume it?', buttons=(YES,NO))
      else:
```

```
      reply=YES
    if reply==YES:
      A.features[InstanceName].resume()
  for InstanceName in I.keys():
    if A.features[InstanceName].isSuppressed():
      return 0
  return 1

def getInitValues(initName, overwrite=True, noPrompt=False):
  '''
  initName is the name of the initial cae file.
  overwrite = True,  replaces the already existing xml file.
  '''
  DIR=os.getcwd()
  FileName = DIR+"/"+initName+".xml"
  if bugMeModus: print '-'*60

  if os.path.exists(FileName) == True and overwrite==True:
    if bugMeModus: print 'start analysing CAE data'
  elif os.path.exists(FileName) == True and overwrite==False:
    return None #if there is already a file and the user decided not to overwrite it


  doc = Document()
  mdb = openMdb(initName)
  MODELNAME=session.sessionState[session.currentViewportName]['modelName']
  M=mdb.models[MODELNAME]
  A=M.rootAssembly
  I=A.instances

  print '#'*25+' opened initial CAE '+'#'*25

  #if some features are suppressed this command will rrsume all of them
  reply=user_isSuppressed(noPrompt)
  if reply==0:
    print 'WARNING: Some features are suppressed.\nThis may cause an ERROR, wich is based on insufficient informations about the initial cae model.'

  #create the <rootNode> base element
  rootNode = doc.createElement("Instance_Settings")
  doc.appendChild(rootNode)

  for instances in I.keys():
    if instances[-1]!='1':
      print 'ERROR on instance name. The instance name has to end with the number 1'
      return None
```

```
BCs={}
for instances in I.keys():
  BCs[instances]=[]

BCNames =  M.boundaryConditions.keys()
check=['nodes','cells','edges', 'elements', 'faces','referencePoints','vertices']
for BCName in BCNames:
  for lookType in check:
    try:
      region=M.boundaryConditions[BCName].region[0]
      if lookType=='referencePoints':
        instanceName=A.sets[region].__getattribute__(lookType)[0].__repr__().split('\'')[-2]
      else:
        instanceName=A.sets[region].__getattribute__(lookType)[0].instanceName
      states=M.steps[C2DStep].boundaryConditionStates[BCName]
      u=[states.u1,states.u2,states.u3,states.ur1,states.ur2,states.ur3]
      BCs[instanceName].append([region,BCName,u,states.amplitude])
      break
    except:
      pass
for i in range(1,len(BCs.keys())+1):
  pname=BCs.keys()[i-1]
  if bugMeModus: print pname+'+'*10
  if BCs[pname]!=[]:
    # Create the main <gr> element
    gr = doc.createElement(str(pname))
    gr.setAttribute("id",str(i))
    try:
      gr.setAttribute("SECTIONNAME",str(M.parts[I[pname].part.name].sectionAssignments[0].sectionName))
    except:
      gr.setAttribute("SECTIONNAME","None")
    for value in range(len(BCs[pname])):
      gr.setAttribute(str(BCs[pname][value][1]),str(BCs[pname][value][2]))
    rootNode.appendChild(gr)
    if bugMeModus: print 'complete: ' + str(traceback.extract_stack()[-1][1])
    ##
    ##ELSE PART IS JUST TEMPORARY. caused by an empty BCs array, the if statement didn't assign the section.
    ##
  else:
    # Create the main <gr> element
    gr = doc.createElement(str(pname))
    gr.setAttribute("id",str(i))
    try:
```

```
        gr.setAttribute("SECTIONNAME",str(M.parts[I[pname].part.name].sectionAssignments[0].sectionName))
      except:
        gr.setAttribute("SECTIONNAME","None")
      for value in range(len(BCs[pname])):
        gr.setAttribute(str(BCs[pname][value][1]),str(BCs[pname][value][2]))
      rootNode.appendChild(gr)
      if bugMeModus: print 'complete: ' + str(traceback.extract_stack()[-1][1])
  #print doc.toprettyxml(indent="")
  writeXML(doc, FileName)
  if bugMeModus: print 'XML file saved: '+str(traceback.extract_stack()[-1][1])
  print '#'*25+' closing initial CAE '+'#'*25
  mdb.close()

def nodeSetToPartList():
  global PartList
  for Part in PartList:
    print Part.name
    if Part.remesh==False and Part.deformed==True:
      if not 'nodeSets' in dir(Part):
        Part.nodeSets=[]
      for bc in Part.boundaryConditions:
        try:
          Part.nodeSets.append((bc[0],[x.label for x in A.sets[bc[0]].nodes]))
        except:
          print 'Error while appending NodeSet to PartList'


""" DEPRECATED, DUE TO OTHER SOLUTIONS TO PREVENT STRESS FLICKERING
def reinstateFrictionForces(StepName,PartName,contactSpecification='',CType='CSHEARF'):
  '''
  Looks for the CNORMF Parameters in the odb file and applies them onto the new nodes.
  contactSpecification: String wich indicates the Name of the Contact e.g.:'ASSEMBLY_KONTAKT_WERKSTOFF_S/ASSEMBLY_KONTAKT_WERKZEUG_S'
  if only one contact is specified contactSpecification=''
  StepName: Name of the odb Step
  PartName= Name of the instance to be restored
  '''

  if (PartList[PartNumbers[PartName+'-1']].remesh != True) and (PartList[PartNumbers[PartName+'-1']].deformed==True):
    instanceName=PartName.upper()+'-'+str(CurCaeCycle)
    delLoads=[i for i in M.loads.keys() if i.startswith('FrN-'+PartName+CType)]
    for i in delLoads:
      try: del M.loads[i]
      except: pass
```

```
try:
  try: del odb
  except: pass
  odb=openOdb(path=JobName+'-'+str(CurCaeCycle)+'.odb',readOnly=TRUE)
except:
  print 'ERROR: Odb File %s could not be opened' %(JobName+'-'+str(CurCaeCycle))
  return 0
#look, if CSHEARF is accessable
fieldOP=odb.steps[StepName].frames[-1].fieldOutputs
CSHEARFlen=len([i for i in fieldOP.keys() if i.startswith(CType)])
if CSHEARFlen > 1 and contactSpecification=='':
  print 'ERROR: please specify which contact has to be analysed'
  odb.close()
  return 0
elif CSHEARFlen == 1:
  ContCSHEAR=[i for i in fieldOP.keys() if i.startswith(CType)][0]
  CSHEAR=fieldOP[ContCSHEAR]
  CSHEARv=CSHEAR.values
  PartV=[i for i in CSHEARv if i.instance.name==instanceName]
  PartData=[(i.nodeLabel,i.data) for i in PartV if i.data[0]!=0]
  if PartData.__len__()==0:
    'No friction values found in part %s' %instanceName
    odb.close()
    return 0
  else:
    print 'Reinstate friction forces'
    for force in PartData:
       addForceToNode(force,PartName+'-'+str(CurCaeCycle+1),CType=CType,PN=PartName)
    odb.close()
else:
  #if more than one contact is available and reinstateFrictionForces is set
  try:
    CSHEAR=fieldOP['CSHEARF    '+contactSpecification]
  except:
    'ERROR: Input Error on contactSpecification in reinstateFrictionForces'
    odb.close()
    return 0
  CSHEARv=CSHEAR.values
  PartV=[i for i in CSHEARv if i.instance.name==instanceName]
  PartData=[(i.nodeLabel,i.data) for i in PartV if i.data[0]!=0]
  if PartData.__len__()==0:
    'No friction values found in part %s' %instanceName
    odb.close()
    return 0
  else:
    for force in PartData:
```

```
          addForceToNode(force,PartName+'-'+str(CurCaeCycle+1),CType=CType,PN=PartName)
        odb.close()
  else:
    print 'ERROR: Part is not deformable or a remeshpart'
    return 0

def addForceToNode(force,instanceName,CType,PN):
  print force,instanceName
  nodes1 = I[instanceName].nodes[force[0]-1:force[0]]
  region = Region(nodes=nodes1,)
  M.ConcentratedForce(name='FrN-'+PN+CType+str(force[0]), createStepName='step-1',
      region=region, cf1=force[1][0], cf2=force[1][1], distributionType=UNIFORM, field='', localCsys=None)
  #if two steps are present
  try:
    M.loads['FrN-'+PN+CType+str(force[0])].deactivate('step-2')
  except: pass

def interpolateSurfaceForces(StepName,PartNameData,PartName,SurfaceName,contactSpecification='',CType='CSHEARF',TOL=0.1):
  if (PartList[PartNumbers[PartName+'-1']].remesh == True) and (PartList[PartNumbers[PartName+'-1']].deformed==True):
    instanceName=PartNameData.upper()+'-'+str(CurCaeCycle)
    delLoads=[i for i in M.loads.keys() if i.startswith('FrN-'+PartName+CType)]
    for i in delLoads:
      try: del M.loads[i]
      except: pass
    try:
      try: del odb
      except: pass
      odb=openOdb(path=JobName+'-'+str(CurCaeCycle)+'.odb',readOnly=TRUE)
    except:
      print 'ERROR: Odb File %s could not be opened' %(JobName+'-'+str(CurCaeCycle))
      return 0
    #look, if CSHEARF is accessable
    fieldOP=odb.steps[StepName].frames[-1].fieldOutputs
    CSHEARFlen=len([i for i in fieldOP.keys() if i.startswith(CType)])
    if CSHEARFlen > 1 and contactSpecification=='':
      print 'ERROR: please specify which contact has to be analysed'
      odb.close()
      return 0
    elif CSHEARFlen == 1:
      ContCSHEAR=[i for i in fieldOP.keys() if i.startswith(CType)][0]
      CSHEAR=fieldOP[ContCSHEAR]
      CSHEARv=CSHEAR.values
```

```
  PartV=[i for i in CSHEARv if i.instance.name==instanceName]
  PartData=[(i.nodeLabel,i.data,odb.rootAssembly.instances[instanceName].nodes[i.nodeLabel-1].coordinates) for i in PartV if i.data[0]!=0]
  if PartData.__len__()==0:
    'No friction values found in part %s' %instanceName
    odb.close()
    return 0
  else:
    PartData=[[i[0],i[1][0],i[1][1],i[2][0],i[2][1]] for i in PartData]

    ##Y-DIRECTION -> interpolating forces in the x direction
    PartData=sorted(PartData,key=itemgetter(4),reverse=True) #sorted coordinates in y direction from top to bottom
    WPNODES=[[i.label,i.coordinates[0],i.coordinates[1]] for i in A.surfaces[SurfaceName].nodes
      if (i.coordinates[1]>PartData[-1][4]-TOL) and
      (i.coordinates[1]<PartData[0][4]+TOL) and
      (i.coordinates[0]<=PartData[-1][3]) and
      (i.coordinates[0]>=PartData[0][3])] ##[nodelabel,coordinates] of the remeshed part.
    WPNODES=sorted(WPNODES,key=itemgetter(2),reverse=True)
    SUM_fX=0 #sum of the input data forces
    SUM_fY=0
    for force in PartData:
      SUM_fX+=force[1]
      SUM_fY+=force[2]

    SUM_fxNEW=0 #SUM of the workpiece forces
    SUM_fyNEW=0
    ForceDataWP=[] #New force array

    for n in range(1,PartData.__len__()):
      ##force = X . direction = Y
      Y1=PartData[n][4]
      Y0=PartData[n-1][4]
      deltaY=Y1-Y0
      fX0=PartData[n-1][1]/2
      fX1=PartData[n][1]/2
      deltafX=fX1-fX0
      k_factor=deltafX/deltaY

      ##force = y . direction = X
      X0=PartData[n-1][3]
      X1=PartData[n][3]
      deltaX=X1-X0
      fY0=PartData[n-1][2]/2
      fY1=PartData[n][2]/2
      deltafY=fY1-fY0
```

```
          k_factor2=deltafY/deltaX

          CURWPNODES=[i for i in WPNODES if (i[2]>=Y1) and (i[2]<=Y0)]

          if len(CURWPNODES)!=0:
            for m in CURWPNODES:
              fxNEW=fX0+k_factor*(m[2]-Y0)
              fyNEW=fY0+k_factor2*(m[1]-X0)
              ForceDataWP.append([m[0],[fxNEW,fyNEW]])

        for i in ForceDataWP:
          SUM_fxNEW+=i[1][0]
          SUM_fyNEW+=i[1][1]

        ScaleFaktorX=SUM_fX/SUM_fxNEW
        ScaleFaktorY=SUM_fY/SUM_fyNEW

        if bugMeModus: pp(WPNODES)
        print 'Scale factor X:'+repr(ScaleFaktorX)
        print 'Scale factor Y:'+repr(ScaleFaktorY)

        NewForces=[[i[0],[i[1]*ScaleFaktorX,i[2]*ScaleFaktorY]] for i in WPNODES]
        print '*'*10
        if bugMeModus: pp(NewForces)
        print 'Reinstate friction forces'
        #pp(ForceDataWP)
        for force in NewForces:
           AddForceToNode(force,PartName+'-'+str(CurCaeCycle+1),CType=CType,PN=
    PartName)
        odb.close()
  else:
    print 'ERROR on Part remesh or deformed'
"""

def createJob():
  mdb.Job(atTime=None, contactPrint=OFF, description='', echoPrint=OFF,
      explicitPrecision=DOUBLE, getMemoryFromAnalysis=True, historyPrint=OFF,
      memory=50, memoryUnits=PERCENTAGE, model=MODELNAME , modelPrint=OFF,
      multiprocessingMode=DEFAULT, name=JobName+'-'+str(NextCaeCycle),
    nodalOutputPrecision=SINGLE,
      numCpus=1, numDomains=1, parallelizationMethodExplicit=DOMAIN, queue=None,
      scratch='', type=ANALYSIS, waitHours=0, waitMinutes=0)
##
### ADDITIONAL FUNCTIONS
##

def printPngToFile(FileName):
  from abaqus import session
```

```
  i=1
  while os.path.isfile(str(FileName)+repr(i)+'.png') == True:
    i+=1
  session.pngOptions.setValues(imageSize=(400, 400))
  session.printToFile(fileName=str(FileName)+repr(i), format=PNG, canvasObjects=(session.viewports[session.currentViewportName], ))

def continueProg(arg):
  if arg == True:
    return None
  else:
    sys.exit()

def addMapSolutions():
  global M
  odb= openOdb(path=JobName+'-'+repr(CurCaeCycle)+'.odb',readOnly=True)
  ostep=odb.steps.keys()[-1]
  firstStep=odb.steps.keys()[0]
  try:
    lastFrame=odb.steps[ostep].frames[-1]
  except:
    print 'ERROR while trying to read the last odb file!'
    return False
  M.keywordBlock.synchVersions(storeNodesAndElements=False)
  for mem in M.keywordBlock.sieBlocks:
    if mem.startswith('*Step, name='+firstStep):
      memID_Initial = M.keywordBlock.sieBlocks.index(mem)
  M.keywordBlock.insert(memID_Initial-2, '\n*MAP SOLUTION, STEP='+str(len(odb.steps))+',INC='+repr(lastFrame.incrementNumber)+', UNBALANCED STRESS=RAMP')
  odb.close()
  return True

def startJob(version='abq6121',Del=False,ChkStep=1):
  '''
  start Job fpr Linux platform
  standard version is abq6121
  '''
  import sys
  if sys.platform == 'win32':
    print 'ERROR: function startJob is only for linux based platforms'
  else:
    mdb.jobs[JobName+'-'+str(NextCaeCycle)].writeInput(consistencyChecking=OFF)
    print 'Input-File successfully generated!'
    #jobmaker = file('jobmaker.sh','w+')
    #jobmaker.write(version+' job='+JobName+'-'+str(NextCaeCycle)+' oldjob='+JobName+'-'+str(CurCaeCycle)+' user=urdfiluvarm.f cpus=1')
    #jobmaker.close()
```

```
    EXECSTRING='xterm -e '+version+' job='+JobName+'-'+str(NextCaeCycle)+' oldjob='+JobName+'-'+str(CurCaeCycle)+' user=urdfiluvarm.f cpus=1 -interactive'
    sub=subprocess.Popen(shlex.split(EXECSTRING))
    sub.wait()
    #os.system('chmod a=rwx jobmaker.sh')
    #os.system('./jobmaker.sh')
    Logpfad=JobName+'-'+str(NextCaeCycle)+'.sta'
    try:
      FILE=open(Logpfad,'r')
    except:
      print 'ERROR: No sta file found: %s' %Logpfad

    if os.path.exists(Logpfad):
      Liste=FILE.readlines()
      print Logpfad+'***'+Liste[-1]
    else:
      return False

    if Del:
      deleteOldData(CurCaeCycle)

    if Liste[-3][3]!=ChkStep and Liste[-1].startswith(' THE ANALYSIS HAS NOT BEEN COMPLETED'):
      return True
    else:
      return False
    #waitForCompletion(1,1000)

def deleteOldData(JOB):
  '''
  deletes old job-data
  '''
  import os
  killJob=JOB-9 #deletes the data of the 9th job before the current job
  if JOB>10:
    os.system('rm -r '+JobName+'-'+repr(killJob)+'.msg')
    os.system('rm -r '+JobName+'-'+repr(killJob)+'.stt')
    os.system('rm -r '+JobName+'-'+repr(killJob)+'.mdl')
    os.system('rm -r '+JobName+'-'+repr(killJob)+'.res')
    os.system('rm -r '+JobName+'-'+repr(killJob)+'_extrapolated.mdl')
    os.system('rm -r '+JobName+'-'+repr(killJob)+'_extrapolated.stt')
    os.system('rm -r '+JobName+'-'+repr(killJob)+'.fil')
    os.system('rm -r '+JobName+'-'+repr(killJob)+'.ipm')
    os.system('rm -r '+JobName+'-'+repr(killJob)+'.log')
    os.system('rm -r '+JobName+'-'+repr(killJob)+'.prt')
    os.system('rm -r '+JobName+'-'+repr(killJob)+'.sim')
    os.system('rm -r '+JobName+'-'+repr(killJob)+'.sta')
    os.system('rm -r '+JobName+'-'+repr(killJob)+'.dat')
```

```
    os.system('rm -r '+JobName+'-'+repr(killJob)+'.com')
    os.system('rm -r '+JobName+'-'+repr(killJob)+'.res')
    os.system('rm -r '+JobName+'-'+repr(killJob)+'.lck')


#Wait for COMPETION


def waitForCompletion(TIME,ZYKLUS):
  '''
  Parameters:
  1. Logpath to the sta file
  2. TIME     -> periode time for the observation
  3. ZYKLUS   -> defines how much periodes occur

  #--------------------------------------------------------------------------------
  #this method looks for the sta-file of the current simulation and looks for the Completed, notCompleted or Abbruch statement in order
  #to continue the script. This is just a possibility to ensure the completion of the simulation.
  '''
  Logpfad=JobName+'-'+str(NextCaeCycle)+'.sta'
  Completed=False
  NotCompleted=False
  Abbruch=False
  STA=False
  Zaehler=0
  while STA==False and Zaehler<=ZYKLUS:
    time.sleep(TIME)
    try:
      Logfile=file(Logpfad, 'r')
    except IOError:
      Zaehler=Zaehler+1
      continue
    if os.path.exists(Logpfad):
      STA=True
  while Completed==False and NotCompleted==False and Abbruch==False:
    time.sleep(TIME)
    try:
      Logfile=file(Logpfad, 'r')
    except IOError:
      Zaehler=Zaehler+1
      continue
    if STA == True:
      Liste=list()
      Liste=Logfile.readlines()
      Logfile.close()
      Zeilen=list()
```

```
      for zeile in Liste:
        Zeilen.append(zeile)
      Length=len(Zeilen)
      for j in range(Length):
        if 'COMPLETED' in Zeilen[j]:
          Completed=True
      for i in range(Length):
        if 'NOT BEEN COMPLETED' in Zeilen[i]:
          NotCompleted=True

def pauseScript(ti):
  import time
  numObjects = 100
  for i in range(numObjects+1):
    milestone('Computing', 'Percentage', i, numObjects)
    time.sleep(float(ti)/numObjects)

def checkOldJobAccess(JobNumber,checker=10):
  check=0
  while check<checker:
    try:
      odb=openOdb(path=JobName+'-'+str(JobNumber)+'.odb',readOnly=TRUE)
      odb.close()
      return
    except:
      PauseScript(5)
      check+=1
  print 'ERROR on odb Access after %s attempts' %checker

def startAutomationProcedure():
  """
  This function is the startfunction for the automation script.
  """
  import logging

  logging.basicConfig(filename='Automator.log',level=logging.DEBUG)
  logging.info('Logfile created: %s' %time.asctime( time.localtime(time.time()) )
    )

  session.journalOptions.setValues(recoverGeometry = INDEX)
  session.journalOptions.setValues(replayGeometry = INDEX)

  AnalyseResult=analyse()
  if AnalyseResult==1:
    ##has to be opened by the UI if possible
    if os.path.exists('USER.txt') and CurCaeAnalysed==True:
      useUserFile=True
      ##maybe as UI checkbox
```

```
  userFile=open('USER.txt','r')
  userFileLines=userFile.readlines()
  inputUSER={}
  if bugMeModus_L1: print '-'*10+'USER FILE'+'-'*10
  for line in userFileLines:
    try:
      inputUSER[line.split('=')[0]]=eval(line.split('=')[1])
      if bugMeModus_L1: print line.split('=')[0],line.split('=')[1]
    except:
      pass
else:
  useUserFile=False #to disable userscripting capability


if useUserFile:

  try: C2DStep=inputUSER['StepName']
  except: C2DStep = M.steps.keys()[-1]


  try:  initialCaeName=inputUSER['initialCaeName']
  except: logging.error('initial Cae File name not found in User.txt')
  if CurCaeIsInit == False:
    analyseInitCAE(initialCaeName, overwrite=True)


  try: remesh=inputUSER['remesh']
  except:
    logging.warning('No part was specified for remeshing in the User.txt')
    remesh=[]

  if remesh !=[]:
    for Part in PartList:
      if (Part.initialName in remesh) or (Part.initialName.rstrip('-1') in
remesh):
        Part.remesh=True
      else:
        Part.remesh=False

  try: globalAbstractAngle=inputUSER['globalAbstractAngle']
  except: logging.info('No changes in global abstract angle by the User.txt')

  try: abstractAngle=inputUSER['abstractAngle']
  except:
    logging.warning('No abstract angles specified in the User.txt')
    abstractAngle={}

  try: deformed=inputUSER['deformed']
```

```
  except:
    logging.warning('No deformed parts')
    deformed=[]
  for Part in PartList:
      if (Part.initialName.rstrip('-1').upper() in abstractAngle):
        Part.abstractAngle=abstractAngle[Part.initialName.rstrip('-1')]
      else:
        Part.abstractAngle=globalAbstractAngle

        print Part.initialName.rstrip('-1').upper()
        print deformed
      if (Part.initialName.rstrip('-1').upper() in deformed):
        Part.deformed=True
      else:
        Part.deformed=False
        logging.info('Part is set to not deformable: %s' %Part.name)

  global newBCVal
  newBCVal={}
  try: newBcVal=inputUSER['newBcVal']
  except: logging.info('No changes in the Boundary Conditions by the User.txt')

  try: importStep=inputUSER['importStep']
  except: importStep=0

  try: ATV=inputUSER['ATV']
  except: logging.info('No global adjustments at contact nodes by the User.txt')

  try: maxCycles=inputUSER['maxCycles']
  except: logging.info('No changes in maxCycles by the User.txt')

else:
  C2DStep = M.steps.keys()[-1]
  ##or from GUI
  ##if CurCaeIsInit == False:
  ## initial Name from GUI
  ##newBcVal = from GUI
  ##importStep = from GUI
  ##ATV = from GUI
  ##maxCycles = from GUI
  ##abstractAngle = from GUI

M.steps[C2DStep].Restart(frequency=1, numberIntervals=0, overlay=OFF, timeMarks=OFF)
M.steps[C2DStep].setValues(maxNumInc=10000)
```

```
      analyseInitCAE(initialCaeName,overwrite=True,noPrompt=True)

      NextCaeCycle = CurCaeCycle+1
      #start of the cycle
    else: stopCriteria=True

def startVariation():
    """
    This function initializes the current cycle and restarts with variations
    """
    mdb = openMdb(CurPath)
    M=mdb.models[session.sessionState[session.currentViewportName]['modelName']]
    A=M.rootAssembly
    I=A.instances
    P=M.parts
    os.system('rm '+JobName+'-'+str(NextCaeCycle)+'*') #delete the abborted
      simulation
```

./Appendices/KernelFunctions.py

B.2 Automation Script

```
#!/usr/bin/env python
"""
This Script is the control script to automatise the deformation process in ABAQUS
    /Standard.

Software compatibility: abaqus 6.121
required modules and/or scripts: KernelFunctions (v0.202)

"""

execfile('KernelFunctions.py')
from odbAccess import *
from abaqus import *
from abaqusConstants import *
import time
import logging
#maybe suppressable imports
from material import *
from section import *
from assembly import *
from step import *
from interaction import *
from load import *
import mesh
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *
from abaqus import backwardCompatibility

__author__ = "Stefan Distlberger"
__copyright__ = "Copyright 2013, Materials Center Leoben, MCL"
__credits__ = ["Stefan Distlberger", "Martin Krobath"]
__license__ = "GPL"
__version__ = "0.200"
__maintainer__ = "Stefan Distlberger"
__email__ = "stefan.distlberger@mcl.at"
__status__ = "Production" #"Developement", "Prototype"

StartAutomationProcedure()

while stopCriteria==False and NextCaeCycle<=maxCycles:

  ReconParts(PartList, C2DStep, JobName, importStep, CurCaeCycle)
  logging.info('Reconstruction of the parts in Cycle %s' %CurCaeCycle)

```

```
if CurCaeIsInit:
  mdb.saveAs(pathName='COMMON2D.cae') ##the name has to be an input parameter from GUI
  CurCaeIsInit=False
  AnalyseInitCAE(initialCaeName,overwrite=False,noPrompt=True)

# Changes in BC thru user input
if newBCVal!={}:
  for Part in PartList:
    for BC in Part.boundaryConditions:
      if BC[1] in newBcVal.keys():
        BC[2]=newBcVal[BC[1]]

# surfaces
##NOT YET AUTOMATED
#A.Surface(side2Edges=A.instances[PartList[0].cleanName+'-'+repr(NextCaeCycle)].edges[0:1],name='kontakt_rigid_s')
getSideEdge(PartList[0].cleanName+'-'+repr(NextCaeCycle),SurfaceName='kontakt_werkzeug_s')
A.Surface(side1Edges=A.instances[PartList[1].cleanName+'-'+repr(NextCaeCycle)].edges,name='kontakt_werkstoff_s')

# interactions
#when the sets and surfaces kept their initial name, the interactions wil be updated automatically

# meshwindows
##usually from GUI
#temporary values
#P1=PartList[1].cleanName+'-'+repr(NextCaeCycle) #referenz
mw=MeshWindow('mw1',PartList[1].cleanName+'-'+repr(NextCaeCycle),0.4)
mw.Window(1.8,10,3.5,-3.9,0.02)
mw.Window(1.2,10,1.8,-3.9,0.03)
mw.Window(3.5,10,100,-3.9,0.02)
mw.Window(1.2,-3.9,100,-4,0.02)
mw.Window(1.2,-4,100,-6,0.05)
mw.Window(-1,-3.2,1.2,-6,0.1)
mw.ApplyMW()
if A.getUnmeshedRegions() != None:
  #if the mesh wasn't created
  #approach for meshwindow displacement
  ## half automated! the coordinate to displace has to be scripted manually
  deltaU=0.1
  counts=0
  while A.getUnmeshedRegions() != None:
    counts+=1
    mw.delMW()
    mw=MeshWindow('mw1',PartList[1].cleanName+'-'+repr(NextCaeCycle),0.4)
```

```
    mw.Window(2+(deltaU*counts),10,3.5,-3.9,0.03)
    mw.Window(1.1+(deltaU*counts),10,2,-3.9,0.03)
    mw.Window(3.5,-3.0,100,-3.9,0.02)
    mw.Window(1.1+(deltaU*counts),-3.9,100,-4,0.03)
    mw.Window(1.1+(deltaU*counts),-4,100,-6,0.05)
    mw.Window(-1,-3.2,1.1+(deltaU*counts),-6,0.1)
    mw.ApplyMW()
    if counts==5:
      print 'ERROR: Not all Faces could be meshed'
      break

elemType1 = mesh.ElemType(elemCode=CPE4T, elemLibrary=STANDARD)
elemType2 = mesh.ElemType(elemCode=CPE3T, elemLibrary=STANDARD)
A.setElementType(regions=(A.instances[PartList[1].cleanName+'-'+repr(NextCaeCycle)].faces,), elemTypes=(elemType1, elemType2))

# sets
createSetbySetName(PartList)
# beispiel A.SetByBoolean(name='_MP_1',sets=(A.sets['_MP_1'],A.sets['MP_TOP']), operation=DIFFERENCE)
A.Set(nodes=I[PartList[1].cleanName+'-'+repr(NextCaeCycle)].nodes,name='werkzeug_all')

# Section Assignment
assignSections()

AddMapSolutions()

# Job
createJob()
PauseScript(2)
checkSuppressState() #checks if initial Parts are still active
checkOldJobAccess(CurCaeCycle)
startJob(Del=TRUE)

CurCaeCycle+=1
NextCaeCycle+=1

if bugMeModus_oneLoop: stopCriteria = True
```

./Appendices/Automator.py

B.3 Rendering Script

```
##################################################
##################################################
'''
scriptname: makeShots
copyright: MCL (c) 2013
author: Stefan Distlberger
version: 1.0
compatibility: abaqus 6.12

------------------------------------------------------------------------------

description:
CL Script, which provides functionallities to render Pictures or video sequences
    out of abaqus and
transcode them into an output video.

------------------------------------------------------------------------------

required modules and/or scripts:
Scripts:
-videoRender.py ab 1.0
-pictureRender.py ab 1.0
-makeShots.bat

PlugIns:
CamPosToXML

Modules:
mplayer-svn-35935
ffmpeg-20130428


------------------------------------------------------------------------------
'''
__Author__='MCL: Stefan Distlberger'
__Review__=''
__Version__='0527,1.0'

#############################################
######## make Shots  ###########################
#############################################

import os, sys, traceback, shutil, time, platform, math
from odbAccess import *
from abaqusConstants import *
from numpy import array, dot, reshape
```

```
from pprint import pprint as pp
import time


BUGGY=0 #debugging level
if BUGGY!=0:
  BUGME=open('BUGMEALL.txt','w')
  BUGME.write('DEBUG LOG VERSION '+__Version__+'\n')
  BUGME.write('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n')
  BUGME.write(time.asctime()+'\n\n')

#~~~~~~~~~~~~~~~~~~~~~~
def usage():
  print '\n'
  print '====================================================='
  print '=============== ODB - Frames to Video ==============='
  print '====================================================='
  print 'Version: '+__Version__
  print 'Author:\t'+ __Author__
  print 'Review:\t'+__Review__
  print '\n'+'-'*60
  print '\nArguments:'

usage()
cwd=os.getcwd()
sys.path.insert(0,cwd)
odb=None
ODB_LIST=[]

##
#### temporary timer for performance tests
##

#TIMER
if sys.platform == "win32":
    timer = time.clock
else:
    timer = time.time
t0 = t1 = 0
def timerstart():
    global t0
    t0 = timer()
def timerfinish():
    global t1
    t1 = timer()
def timerseconds():
    return int(t1 - t0)
def timermilli():
```

```
    return int((t1 - t0) * 1000)
def timermicro():
    return int((t1 - t0) * 1000000)

##
#### functions
##

# End program
def endProg():
  if odb:
    odb.close()
  if BUGGY!=0:
    if BUGME: BUGME.close()
  timerfinish()
  print str(timerseconds())+' seconds'
  print '\n'
  print '=============== program finished ==============='
  sys.exit()
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Continue Program
def contProg():
  dec=raw_input('Do you wish to continue [y/n(default)]:')
  if (dec!='y' and dec!='Y'):
    endProg()
  return 1
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# get odbPath
def getodbPath():
  '''
  returns an array with the paths to all of the odb files
  '''
  retPathArray=[]
  check=False

  while check==False:
    odbPreName=raw_input('\nEnter the name of the odb File to process (w/o .odb):
    ')
    odbRange=raw_input('\nEnter the range of the odbs.\nIf there is only one odb
    to process press ENTER.\nIf there is a range type as following [2,24]:')
    #check single ODB file
    if odbRange=='' or odbRange=="''":
      odbName=odbPreName
      if odbName=='' or odbName=="''":
        print '\nERROR: Entered ODB name not found...'
        contProg()
        odbPath, odbName = '',''
```

```
      continue
    if odbName.endswith('.odb'):
      odbName=odbName.rstrip('.odb')
    odbPath=odbName+'.odb'
    if not(os.path.exists(odbPath)):
      print '\n'+'-'*60
      err='\nERROR: The odb file - %s - does not exist.' % (odbPath,)
      print err
      os.system('dir *.odb')
      print '\n'+'-'*60
      contProg()
      continue
    print '\nodb path: '+odbPath
    retPathArray.append(odbPath)
    check=True
    print '\n=============== DONE ==============='
    return retPathArray

  #check multiple ODB file
  else:
    try:
      odbR=eval(odbRange)
    except:
      print '\nERROR: The form of the range has to be like the example: [2,24]'
      contProg()
      continue
    if type(odbR).__name__=='str':
      print '\nERROR: The form of the range has to be like the example: [2,24]'
      contProg()
      continue
    if odbR[1]<=odbR[0]:
      print '\nERROR: The second number has to be higher as the first number...
  '
      contProg()
      odbPath, odbName = '',''
      continue
    else:
      for odbNumber in range(odbR[0],odbR[1]+1):
        if odbPreName=='' or odbPreName=="''":
          print 'ERROR: Entered ODB name not found...'
          contProg()
          odbPath, odbName = '',''
          continue
        if odbPreName.endswith('.odb'):
          odbPreName=odbPreName.rstrip('.odb')
        if odbPreName.endswith('-'):
          odbPreName=odbPreName.rstrip('-')
        odbName=odbPreName+'-'+str(odbNumber)
```

```
          odbPath=odbName+'.odb'
          if not(os.path.exists(odbPath)):
            print '\n'+'-'*60
            err='\nERROR: The odb file - %s - does not exist.' % (odbPath,)
            print err
            os.system('dir *.odb')
            print '\n'+'-'*60
            contProg()
            continue
          retPathArray.append(odbPath)
        print '\nodb path: '+odbPath+'\nthe numbers range is: '+odbRange
        check=True
        print '\n=============== DONE ==============='
        return retPathArray
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

def setCamera(ODB_LIST):
  if len(ODB_LIST)==0:
    print 'No ODB path found'
    endProg()
  if query_yes_no('Do you want to set the Camera position?'):
    print '1) Adjust the camera position.\n2) Set the primary field variable and limits.\n3) Go to Plug-ins / MCL / SaveCam... and update and export the current state.'
    contProg()
    os.system('abaqus viewer database='+ODB_LIST[0])
    print '\n=============== closing Abaqus ==============='
  else:
    pass
    # get the path to the XML file
    # read XML file


#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def query_yes_no(question, default="yes"):
  '''Ask a yes/no question via raw_input() and return their answer.

  "question" is a string that is presented to the user.
  "default" is the presumed answer if the user just hits <Enter>.
    It must be "yes" (the default), "no" or None (meaning
    an answer is required of the user).

  The "answer" return value is one of "yes" or "no".
  '''
  valid = {"yes":True, "y":True, "ye":True, "no":False, "n":False}
  if default == None:
    prompt = " [y/n] "
  elif default == "yes":
```

```
    prompt = " [Y (default) /n] "
  elif default == "no":
    prompt = " [y/N (default)] "
  else:
    raise ValueError("invalid default answer: '%s'" % default)

  while True:
    sys.stdout.write(question + prompt)
    choice = raw_input().lower()
    if default is not None and choice == '':
      return valid[default]
    elif choice in valid:
      return valid[choice]
    else:
      sys.stdout.write("Please respond with 'yes' or 'no' (or 'y' or 'n').\n")

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def makePics():
  print '\n=============== start rendering pictures with abaqus ==============='
  os.system('abaqus viewer script=pictureRender.py')
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

def makeVids():
  print '\n=============== start rendering videos with abaqus ==============='
  os.system('abaqus cae script=videoRender.py')

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def picsToAvi(version='mplayer-svn-35935',FileName="output.avi",ListName="list.txt",fps=25):
    if os.path.exists(os.getcwd()+'\\'+version):
      os.system(version+"\mencoder mf://@"+ListName+" -mf type=png:fps="+str(fps)+" -ovc x264 -x264encopts preset=slow:tune=film:crf=5 -oac copy -o "+FileName)
    else:
      print "Error while reading version, filename or/and listname"

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def AviToAvi(version='mplayer-svn-35935',FileName="output.avi",ListName="list.txt"):
    VidNames=""
    list=open(ListName,'r')
    lines=list.readlines()
    for line in lines:
      if line.endswith('\n'):
        VidNames+=" "+line.rstrip('\n')
      else:
        VidNames+=" "+line
    if os.path.exists(os.getcwd()+'\\'+version):
```

```
      os.system(version+"\mencoder -ovc x264 -forceidx -o "+FileName+" "+VidNames)
      print version+"\mencoder -ovc x264 -forceidx -o "+FileName+" "+VidNames
    else:
      print "Error while reading version, filename or/and listname"
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

def timeOrframeBased():
  print "\nThe Video can be set as frame-based or time-based"
  answer=query_yes_no("Do you want a time-based video")
  fileName=raw_input("Please enter the name of the output file [default='output']:")
  if fileName=="" or fileName=='':
    fileName='output'
  else:
    contProg()
  if answer:
    makeVids()
    try:
      os.system('del odb_list.txt')
    except:
      os.system('rm -r odb_list.txt')
    AviToAvi(FileName=fileName+'.avi')
    try:
      list=open('list.txt')
      lines=list.readlines()
      for i in lines:
        os.system('del '+i[:-1])
    except:
      print "Error while trying to delete the old avi files"
  else:
    makePics()
    try:
      os.system('del odb_list.txt')
    except:
      os.system('rm -r odb_list.txt')
    picsToAvi(FileName=fileName+'.avi',fps=25)
    try:
      os.system('del *.png')
    except:
      os.system('rm -r *.png')

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~



##Program Start##
```

```

ODB_LIST=getodbPath()
setCamera(ODB_LIST)
#os.system('echo '+str(ODB_LIST)+'> odb_list.txt')
thefile=file('odb_list.txt','w')
thefile.write('[')
for item in ODB_LIST:
  if not item==ODB_LIST[0]: thefile.write(',')
  thefile.write("'%s'" % item)
thefile.write(']')
thefile.close()
time.sleep(1)
timeOrframeBased()


print '=============== program finished ==============='

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

./Appendices/makeShots.py

```
###################################################
###################################################
'''
scriptname: pictureRender
copyright: MCL (c) 2013
author: Stefan Distlberger
version: 1.0
compatibility: abaqus 6.12

---------------------------------------------------------------------------

description:
Script to Render Pictures in corporation with makeShots.py

---------------------------------------------------------------------------


required modules and/or scripts:
required for: makeShots.py  ab 0404,0.1

---------------------------------------------------------------------------
'''
from odbAccess import *
from abaqus import session
from abaqusConstants import *
import sys, os
from xml.dom.minidom import Document
import xml.dom
import xml.dom.minidom

legend=True
state=True
title=True

RESOLUTIONS=[(640,480),(1280,720),(1600,900),(1920,1080),(2048,1152)]
QUALITY=RESOLUTIONS[1] #here you can choose between the resolutions mentioned above
#==================================================================
if os.path.exists('odb_list.txt'):
  file = open('odb_list.txt','r')
else:
  sys.exit(0)
lines=file.readlines()
odbList=eval(lines[0])

###################################################
def _updateVP_():
  global sv
```

```
  global svV1oD
  global svV1
  global svV1view
  sv=session.viewports
  svV1=sv[session.currentViewportName]
  svV1view=svV1.view
  svV1oD=svV1.odbDisplay.primaryVariable
  svV1.makeCurrent()
  svV1.odbDisplay.display.setValues(plotState=(CONTOURS_ON_DEF, ))

##################################################
#sets the saved variables of the vieport in viewport-1.
def setSavedViewportZoom():
  _updateVP_()
  readDoc = xml.dom.minidom.parse('VideoSettings1.xml')
  theNode = readDoc.getElementsByTagName("ViewportData_2")
  for e in theNode[0].childNodes:
    if e.nodeType == e.ELEMENT_NODE:
      nP=float(e.getAttribute("nearPlane"))
      fP=float(e.getAttribute("farPlane"))
      w=float(e.getAttribute("width"))
      h=float(e.getAttribute("heigth"))
      vOX=float(e.getAttribute("viewOffsetX"))
      vOY=float(e.getAttribute("viewOffsetY"))
      cPu=e.getAttribute("cameraPosition")
      cP=eval(cPu)
      cUVu=e.getAttribute("cameraUpVector")
      cUV=eval(cUVu)
      cTu=e.getAttribute("cameraTarget")
      cT=eval(cTu)
  svV1view.setValues(nearPlane=nP, farPlane=fP, width=w, height=h, viewOffsetX=
    vOX, viewOffsetY=vOY, cameraTarget=cT,cameraUpVector=cUV, cameraPosition=cP)



##################################################

def setSavedPVarSettings():
  _updateVP_()
  OUTPUTPOSITIONS={'U':NODAL,'S':INTEGRATION_POINT,'NT11':NODAL,'PEEQ':
    INTEGRATION_POINT,'CF':NODAL,'CSHEAR1':ELEMENT_NODAL,'CSLIP1':ELEMENT_NODAL,'
    PE':INTEGRATION_POINT,'LE':INTEGRATION_POINT}
  readDoc = xml.dom.minidom.parse('VideoSettings1.xml')
  try:
    theNode = readDoc.getElementsByTagName("ViewportData_3")
    '''
    for nex in theNode[0].childNodes:
```

```
      if str(nex.nodeName) == getCurrentPrimaryVariable()[0] and str(nex.getAttribute("type"))==getCurrentPrimaryVariable()[1]:
        svV1.odbDisplay.contourOptions.setValues(maxAutoCompute=OFF, maxValue=float(nex.getAttribute('max')), minAutoCompute=OFF, minValue=float(nex.getAttribute('min')))
        print '#'*26+' DONE '+'#'*26
    '''
    nex=theNode[0].childNodes[1]
    if str(nex.getAttribute("type"))!="" and str(nex.getAttribute("type"))!='""':
      session.viewports[session.currentViewportName].odbDisplay.setPrimaryVariable(variableLabel=str(nex.nodeName), outputPosition=OUTPUTPOSITIONS[str(nex.nodeName)], refinement=(INVARIANT, str(nex.getAttribute("type"))), )
    else:
      session.viewports[session.currentViewportName].odbDisplay.setPrimaryVariable(variableLabel=str(nex.nodeName), outputPosition=OUTPUTPOSITIONS[str(nex.nodeName)],)
  except:
    print 'ERROR reading or setting the attributes for the primary Variable'

###################################################

def PrintPngToFile(FileName):
  i=1
  while os.path.isfile(str(FileName)+repr(i)+'.png') == True:
    i+=1
  session.pngOptions.setValues(imageSize=QUALITY)
  session.printToFile(fileName=str(FileName)+repr(i), format=PNG, canvasObjects=(session.viewports['Viewport: 1'], ))
  return i

###################################################

def setPrintNotationPositions(legend, state, title):
  session.graphicsOptions.setValues(backgroundOverride=OFF, translucencyMode=2, backgroundStyle=SOLID,backgroundColor='#FFFFFF')
  svV1.viewportAnnotationOptions.setValues(statePosition=(1, 22),titlePosition=(1, 9),triad=OFF,stateBox=ON, stateBackgroundStyle=OTHER, stateBackgroundColor='#FFFFFF',stateFont='-*-arial-medium-r-normal-*-*-80-*-*-p-*-*-*',
  titleBox=ON, titleBackgroundStyle=OTHER, titleBackgroundColor='#FFFFFF', titleFont='-*-arial-medium-r-normal-*-*-80-*-*-p-*-*-*',legendBackgroundStyle=OTHER, legendBackgroundColor='#FFFFFF',legendPosition=(1, 98))
  if legend == True:
    session.viewports[session.currentViewportName].viewportAnnotationOptions.setValues(legend=ON)
  else:
```

```
    session.viewports[session.currentViewportName].viewportAnnotationOptions.setValues(legend=OFF)

  if state == True:
    session.viewports[session.currentViewportName].viewportAnnotationOptions.setValues(state=ON)
  else:
    session.viewports[session.currentViewportName].viewportAnnotationOptions.setValues(state=OFF)

  if title == True:
    session.viewports[session.currentViewportName].viewportAnnotationOptions.setValues(title=ON)
  else:
    session.viewports[session.currentViewportName].viewportAnnotationOptions.setValues(title=OFF)

##################################################
##################################################
#cleaning old pictures
try:
  os.system('del *.png')
except:
  os.system('rm -r *.png')
counter = 0
for odbName in odbList:
  os.system('echo '+odbName)
  odb=session.openOdb(path=odbName,readOnly=TRUE,name=odbName)
  session.viewports[session.currentViewportName].setValues(displayedObject=None)
  session.viewports[session.currentViewportName].setValues(displayedObject=odb)
  session.viewports[session.currentViewportName].odbDisplay.display.setValues(
    plotState=(CONTOURS_ON_DEF, ))
  setSavedViewportZoom()
  setSavedPVarSettings()
  setPrintNotationPositions(legend, state, title)


  #for oSTEP in odb.steps.keys():
  # for oFRAME in range(len(odb.steps[oSTEP].frames)):
  oSTEP=odb.steps.keys()[-1]
  for oFRAME in range(len(odb.steps[oSTEP].frames)):
    session.viewports[session.currentViewportName].odbDisplay.setFrame(step=oSTEP, frame=oFRAME)
    counter=PrintPngToFile('Picture')
  odb.close()
file = open('list.txt','w')
for i in range(0,counter):
  file.write('Picture'+str(i+1)+'.png\n')
```

```
file.close()
sys.exit(0)
```

./Appendices/pictureRender.py

```
###################################################
###################################################
'''
scriptname: videoRender
copyright: MCL (c) 2013
author: Stefan Distlberger
version: 1.0
compatibility: abaqus 6.12

-------------------------------------------------------------------------

description:
Script to Render Videos in corporation with makeShots.py

-------------------------------------------------------------------------

required modules and/or scripts:
only working with: makeShots.py 0404,0.1

-------------------------------------------------------------------------

'''
from odbAccess import *
from abaqus import session
from abaqusConstants import *
import sys, os
from xml.dom.minidom import Document
import xml.dom
import xml.dom.minidom

legend=True
state=True
title=True

RESOLUTIONS=[(640,480),(1280,720),(1600,900),(1920,1080),(2048,1152)]
QUALITY=RESOLUTIONS[1] #here you can choose between the resolutions mentioned above
#===================================================================
if os.path.exists('odb_list.txt'):
  file = open('odb_list.txt','r')
else:
  sys.exit(0)
lines=file.readlines()
odbList=eval(lines[0])

###################################################
def _updateVP_():
```

```
  global sv
  global svV1oD
  global svV1
  global svV1view
  sv=session.viewports
  svV1=sv[session.currentViewportName]
  svV1view=svV1.view
  svV1oD=svV1.odbDisplay.primaryVariable
  svV1.makeCurrent()
  svV1.odbDisplay.display.setValues(plotState=(CONTOURS_ON_DEF, ))

###################################################
#sets the saved variables of the vieport in viewport-1.
def setSavedViewportZoom():
  _updateVP_()
  readDoc = xml.dom.minidom.parse('VideoSettings1.xml')
  theNode = readDoc.getElementsByTagName("ViewportData_2")
  for e in theNode[0].childNodes:
    if e.nodeType == e.ELEMENT_NODE:
      nP=float(e.getAttribute("nearPlane"))
      fP=float(e.getAttribute("farPlane"))
      w=float(e.getAttribute("width"))
      h=float(e.getAttribute("heigth"))
      vOX=float(e.getAttribute("viewOffsetX"))
      vOY=float(e.getAttribute("viewOffsetY"))
      cPu=e.getAttribute("cameraPosition")
      cP=eval(cPu)
      cUVu=e.getAttribute("cameraUpVector")
      cUV=eval(cUVu)
      cTu=e.getAttribute("cameraTarget")
      cT=eval(cTu)
  svV1view.setValues(nearPlane=nP, farPlane=fP, width=w, height=h, viewOffsetX=
    vOX, viewOffsetY=vOY, cameraTarget=cT,cameraUpVector=cUV, cameraPosition=cP)



###################################################

def setSavedPVarSettings():
  _updateVP_()
  OUTPUTPOSITIONS={'U':NODAL,'S':INTEGRATION_POINT,'NT11':NODAL,'PEEQ':
    INTEGRATION_POINT,'CF':NODAL,'CSHEAR1':ELEMENT_NODAL,'CSLIP1':ELEMENT_NODAL,'
    PE':INTEGRATION_POINT,'LE':INTEGRATION_POINT}
  readDoc = xml.dom.minidom.parse('VideoSettings1.xml')
  try:
    theNode = readDoc.getElementsByTagName("ViewportData_3")
    '''
    for nex in theNode[0].childNodes:
```

```
      if str(nex.nodeName) == getCurrentPrimaryVariable()[0] and str(nex.getAttribute("type"))==getCurrentPrimaryVariable()[1]:
        svV1.odbDisplay.contourOptions.setValues(maxAutoCompute=OFF, maxValue=float(nex.getAttribute('max')), minAutoCompute=OFF, minValue=float(nex.getAttribute('min')))
        print '#'*26+' DONE '+'#'*26
    '''
    nex=theNode[0].childNodes[1]
    svV1.odbDisplay.contourOptions.setValues(maxAutoCompute=OFF, maxValue=float(nex.getAttribute('max')), minAutoCompute=OFF, minValue=float(nex.getAttribute('min')))

    if str(nex.getAttribute("type"))!="" and str(nex.getAttribute("type"))!='""':
      try:
        session.viewports[session.currentViewportName].odbDisplay.setPrimaryVariable(variableLabel=str(nex.nodeName), outputPosition=OUTPUTPOSITIONS[str(nex.nodeName)], refinement=(INVARIANT, str(nex.getAttribute("type"))), )
      except:
        session.viewports[session.currentViewportName].odbDisplay.setPrimaryVariable(variableLabel=str(nex.nodeName), outputPosition=OUTPUTPOSITIONS[str(nex.nodeName)], refinement=(COMPONENT , str(nex.getAttribute("type"))), )
    else:
      session.viewports[session.currentViewportName].odbDisplay.setPrimaryVariable(variableLabel=str(nex.nodeName), outputPosition=OUTPUTPOSITIONS[str(nex.nodeName)],)
  except:
    print 'ERROR reading or setting the attributes for the primary Variable'

##################################################

def PrintPngToFile(FileName):
  i=1
  while os.path.isfile(str(FileName)+repr(i)+'.png') == True:
    i+=1
  session.pngOptions.setValues(imageSize=QUALITY)
  session.printToFile(fileName=str(FileName)+repr(i), format=PNG, canvasObjects=(session.viewports['Viewport: 1'], ))
  return i

##################################################

def setPrintNotationPositions(legend, state, title):
  session.graphicsOptions.setValues(backgroundOverride=OFF, translucencyMode=2, backgroundStyle=SOLID,backgroundColor='#FFFFFF')
```

```
  svV1.viewportAnnotationOptions.setValues(statePosition=(1, 22),titlePosition
    =(1, 9),triad=OFF,stateBox=ON, stateBackgroundStyle=OTHER,
    stateBackgroundColor='#FFFFFF',stateFont='-*-arial-medium-r-normal
    -*-*-80-*-*-p-*-*-*',
  titleBox=ON, titleBackgroundStyle=OTHER, titleBackgroundColor='#FFFFFF',
    titleFont='-*-arial-medium-r-normal-*-*-80-*-*-p-*-*-*',legendBackgroundStyle
    =OTHER, legendBackgroundColor='#FFFFFF',legendPosition=(1, 98))
  if legend == True:
    session.viewports[session.currentViewportName].viewportAnnotationOptions.
    setValues(legend=ON)
  else:
    session.viewports[session.currentViewportName].viewportAnnotationOptions.
    setValues(legend=OFF)

  if state == True:
    session.viewports[session.currentViewportName].viewportAnnotationOptions.
    setValues(state=ON)
  else:
    session.viewports[session.currentViewportName].viewportAnnotationOptions.
    setValues(state=OFF)

  if title == True:
    session.viewports[session.currentViewportName].viewportAnnotationOptions.
    setValues(title=ON)
  else:
    session.viewports[session.currentViewportName].viewportAnnotationOptions.
    setValues(title=OFF)

####################################################
####################################################
#cleaning old pictures
try:
  os.system('del Vid*.avi')
except:
  os.system('rm -r Vid*.avi')
counter = 0

import animation
from abaqusConstants import *
import time

minFrames=5
minStepTime=1e-2
TI=1E-6
#scaleFactor=200

for odbName in odbList:
  totalFrames=0
```

```
totalTime=0

os.system('echo '+odbName)
odb=session.openOdb(path=odbName,readOnly=TRUE,name=odbName)

session.viewports[session.currentViewportName].setValues(displayedObject=None)
session.viewports[session.currentViewportName].setValues(displayedObject=odb)
session.viewports[session.currentViewportName].odbDisplay.display.setValues(plotState=(CONTOURS_ON_DEF, ))
setSavedViewportZoom()
setSavedPVarSettings()
#TEST
#session.viewports[session.currentViewportName].odbDisplay.contourOptions.setValues(contourType=LINE, tickmarkPlots=ON)
session.viewports[session.currentViewportName].odbDisplay.commonOptions.setValues(visibleEdges=FEATURE ) #Possible values are ALL, EXTERIOR, FEATURE, FREE, and NONE
#
setPrintNotationPositions(legend, state, title)

for s in odb.steps.keys():
  totalFrames+=odb.steps[s].frames.__len__()
  totalTime+=odb.steps[s].timePeriod

#if odbName == odbList[0]:
maxTime=session.animationController.animationOptions.maxTime
# TI = float(getInput('Enter the timeIncrement (current maxTime for '+str(totalFrames)+' Frames is '+str(maxTime)+'):'))

if totalFrames >= minFrames:
  counter+=1
  session.odbData[odbName].steps['step-1'].setValues(activateFrames=OFF, )
  session.aviOptions.setValues(sizeDefinition=USER_DEFINED, imageSize=QUALITY)
  session.animationController.animationOptions.setValues(timeIncrement=TI, timeHistoryMode=TIME_BASED)
  session.imageAnimationOptions.setValues(vpDecorations=ON, vpBackground=OFF, compass=OFF, frameRate=50)
  session.animationController.setValues(animationType=TIME_HISTORY, viewports=('Viewport: 1', ) )
  session.animationController.play(duration=UNLIMITED)
  time.sleep(2)
  session.aviOptions.setValues(codecOptions='[16]:enfdfgedbiaaaaaaaeaaaaaaelaaaaaa', compressionQuality=100)
  session.writeImageAnimation(fileName=str(counter), format=AVI, canvasObjects=(session.viewports['Viewport: 1'], ))
else:
  print 'To less Frames in this odb file: %s' %odbName
  print 'odb %s skipped' %odbName
```

```
  odb.close()

file = open('list.txt','w')
for i in range(0,counter):
  file.write(str(i+1)+'.avi\n')
file.close()
sys.exit(0)
```

./Appendices/videoRender.py

Appendix C

Material data

C.1 Workpiece material data

Input file for the workpiece's material data.

```
** Stahl Werkstoff JCDN
*Material, name=Werkstoff
*Conductivity
46.,20.
*Density
 7.8e-09,
*Elastic
210000., 0.3
*Expansion
 1.2e-05,
*Inelastic Heat Fraction
         0.9,
*Plastic, hardening=JOHNSON COOK
400., 340., 0.15,   1.,1520.,  20.
*Rate Dependent, type=JOHNSON COOK
 0.01, 0.001
*Specific Heat
 4.76e+08,
*User Output Variables
      2,
```

./Appendices/WorkpieceData.inp

C.2 Tool material data

Input file for the tool's material data.

```
** Werkzeug
**
*Material, name=Werkzeug
*Conductivity
 95.1,   25.2
 85.2,  250.9
 77.8,   504.
 74.6,  603.4
 71.4,  702.4
 68.2,  801.8
 66.3,  901.5
 65.4, 1001.4
*Density
 1.498e-08,
*Elastic
599000., 0.227,   25.
596000., 0.234,  100.
591000., 0.238,  200.
586000., 0.231,  300.
579000., 0.226,  400.
573000., 0.239,  500.
567000., 0.231,  600.
560000., 0.234,  700.
552000.,  0.23,  800.
536000.,  0.23,  900.
514000.,  0.23, 1000.
485000.,  0.23, 1100.
*Expansion, zero=291.
 4.17e-06,  80.
  4.3e-06, 100.
 4.42e-06, 150.
 4.55e-06, 200.
 4.67e-06, 250.
 4.77e-06, 300.
 4.85e-06, 350.
 4.95e-06, 400.
 5.03e-06, 450.
  5.1e-06, 500.
 5.16e-06, 550.
 5.21e-06, 600.
 5.27e-06, 650.
 5.32e-06, 700.
 5.38e-06, 750.
 5.43e-06, 800.
```

```
5.48e-06, 850.
5.55e-06, 900.
5.64e-06, 950.
5.71e-06,1000.
5.78e-06,1050.
5.85e-06,1100.
5.94e-06,1150.
6.02e-06,1200.
6.09e-06,1250.
6.14e-06,1300.
*Inelastic Heat Fraction
        0.9,
*Plastic
 62.876,        0.
125.752, 0.0002333
188.628, 0.0003499
251.504, 0.0004665
 314.38, 0.0005831
377.256, 0.0006997
440.132, 0.0008163
503.008, 0.0009328
565.884, 0.0010494
 628.76, 0.0011659
691.636, 0.0012824
754.512, 0.0013989
817.388, 0.0015155
880.264, 0.0016319
 943.14, 0.0017484
1006.02, 0.0018649
1068.89, 0.0019813
1131.77, 0.0020978
1194.64, 0.0022146
1257.52, 0.0023575
 1320.4, 0.0025177
1383.27, 0.0026804
1446.15, 0.0028447
1509.02, 0.0030107
 1571.9, 0.0031785
1634.78, 0.0033484
1697.65, 0.0035205
1760.53,  0.003695
 1823.4, 0.0038721
1886.28, 0.0040521
1949.16, 0.0042353
2012.03, 0.0044219
2074.91, 0.0046124
2137.78,  0.004807
2200.66, 0.0050064
```

```
2263.54, 0.0052111
2326.41, 0.0054217
2389.29,  0.005639
2452.17, 0.0058641
2515.04,  0.006098
2577.92, 0.0063421
2640.79, 0.0065982
2703.67, 0.0068685
2766.55, 0.0071556
2829.42, 0.0074631
 2892.3, 0.0077956
2955.17, 0.0081588
3018.05, 0.0085605
3080.93, 0.0090106
 3143.8,  0.009522
3206.68, 0.0101103
3269.55,  0.010793
3332.43, 0.0115879
3395.31, 0.0125099
3458.18, 0.0135712
3521.06, 0.0147843
3583.93, 0.0161681
3646.81, 0.0177551
3709.69, 0.0195992
3772.56, 0.0217889
3835.44, 0.0244712
3898.31, 0.0279002
3961.19, 0.0325359
4024.07, 0.0391978
4086.94, 0.0489592
4149.82, 0.0619803
4212.69, 0.0769483
4275.57, 0.0926595
4338.45,  0.108551
4401.32,  0.124388
 4464.2,   0.14006
4527.08,  0.155507
4589.97,  0.170695
4652.87,  0.185599
4715.75,  0.200207
4778.63,  0.214511
4841.51,   0.22851
4904.39,  0.242208
4967.26,  0.255615
5030.14,  0.268741
5093.01,  0.281604
5155.88,  0.294218
5218.76,  0.306601
```

```
 5281.63,   0.31877
  5344.5,  0.330742
 5407.37,  0.342533
 5470.25,  0.354157
 5533.12,  0.365628
 5595.99,  0.376959
 5658.87,  0.388162
 5721.74,  0.399245
 5784.61,   0.41022
 5847.49,  0.421094
 5910.36,  0.431875
 5973.24,   0.44257
 6036.11,  0.453187
 6098.99,  0.463729
 6161.86,  0.474203
 6224.74,  0.484614
 6287.61,  0.494966
   9500.,        1.
*Specific Heat
   2e+08,   25.2
 2.4e+08,  250.9
 2.7e+08,   504.
 2.7e+08,  603.4
 2.8e+08,  702.4
 2.8e+08,  801.8
 2.9e+08,  901.5
   3e+08, 1001.4
**
```

./Appendices/ToolData.inp

C.3 Hard inclusion material data

Input file for the hard inclusion's material data.

```
** Hard inclusion
*Material, name=Inclusion
*Conductivity
 95.1,   25.2
 85.2,  250.9
 77.8,   504.
 74.6,  603.4
 71.4,  702.4
 68.2,  801.8
 66.3,  901.5
```

```
 65.4, 1001.4
*Density
 1.8e-08,
*Elastic
400000., 0.2
*Expansion, zero=291.
 4.17e-06,  80.
  4.3e-06, 100.
 4.42e-06, 150.
 4.55e-06, 200.
 4.67e-06, 250.
 4.77e-06, 300.
 4.85e-06, 350.
 4.95e-06, 400.
 5.03e-06, 450.
  5.1e-06, 500.
 5.16e-06, 550.
 5.21e-06, 600.
 5.27e-06, 650.
 5.32e-06, 700.
 5.38e-06, 750.
 5.43e-06, 800.
 5.48e-06, 850.
 5.55e-06, 900.
 5.64e-06, 950.
 5.71e-06,1000.
 5.78e-06,1050.
 5.85e-06,1100.
 5.94e-06,1150.
 6.02e-06,1200.
 6.09e-06,1250.
 6.14e-06,1300.
*Inelastic Heat Fraction
        0.9,
*Plastic
 2e+12,0.
 1e+13,1.
*Specific Heat
   2e+08,   25.2
 2.4e+08,  250.9
 2.7e+08,   504.
 2.7e+08,  603.4
 2.8e+08,  702.4
 2.8e+08,  801.8
 2.9e+08,  901.5
   3e+08, 1001.4
```

./Appendices/HardIncData.inp

C.4 Graphite material data

Input file for the graphite's material data.

```
** Graphite inclusion
*Material, name=Inclusion
*Conductivity
 80.,  0.
240.,100.
*Density
 2.2e-08,
*Elastic
150000., 0.3
*Expansion, zero=291.
 4.17e-06,  80.
  4.3e-06, 100.
 4.42e-06, 150.
 4.55e-06, 200.
 4.67e-06, 250.
 4.77e-06, 300.
 4.85e-06, 350.
 4.95e-06, 400.
 5.03e-06, 450.
  5.1e-06, 500.
 5.16e-06, 550.
 5.21e-06, 600.
 5.27e-06, 650.
 5.32e-06, 700.
 5.38e-06, 750.
 5.43e-06, 800.
 5.48e-06, 850.
 5.55e-06, 900.
 5.64e-06, 950.
 5.71e-06,1000.
 5.78e-06,1050.
 5.85e-06,1100.
 5.94e-06,1150.
 6.02e-06,1200.
 6.09e-06,1250.
 6.14e-06,1300.
*Inelastic Heat Fraction
        0.9,
*Plastic
103.,0.
*Specific Heat
 7.12e+08,
```

./Appendices/GraphiteData.inp