

# Bachelorarbeit

Arbeitstitel:

**Präventive messtechnische Abbildung typischer Fertigungsabweichungen**

Betreuer: M. Sc. Robert Hofmann

Prüfer: Univ.-Prof. Dr.-Ing. Sophie Gröger

Name: Jan Doant  
E-Mail: jan.doant@s2017.tu-chemnitz.de  
Studiengang: Bachelor Maschinenbau (7. Semester)  
Matrikelnummer: 461311

Eingereicht am: 06.06.2018

# Inhaltsverzeichnis

<b>Aufgabenstellung</b>	<b>3</b>
<b>1 Einführung</b>	<b>4</b>
<b>2 Grundlagen</b>	<b>5</b>
2.1 Oberflächenabweichungen und -unvollkommenheiten . . . . .	5
2.1.1 Wirkliche Oberfläche . . . . .	5
2.1.2 Istoberfläche . . . . .	5
2.1.3 Geometrische Oberfläche . . . . .	5
2.1.4 Gestaltabweichungen . . . . .	6
2.1.5 Oberflächenunvollkommenheiten . . . . .	7
2.2 Typische Abweichungen bei verschiedenen spanenden Fertigungsverfahren . . . . .	8
2.2.1 Ursachen für Fertigungsabweichungen . . . . .	8
2.2.2 Fertigungsabweichungen beim Drehen . . . . .	10
2.2.3 Fertigungsabweichungen beim Bohren . . . . .	12
2.2.4 Fertigungsabweichungen beim Fräsen . . . . .	13
2.2.5 Fertigungsabweichungen beim Räumen . . . . .	14
2.2.6 Fertigungsabweichungen beim Schleifen . . . . .	15
<b>3 Lösungsansatz</b>	<b>16</b>
3.1 Lösungsidee . . . . .	16
3.2 Beschreibung der Lösungsschritte . . . . .	16
3.3 Eingabegeometrie . . . . .	17
3.3.1 Verwendung von STEP-Dateien . . . . .	17
3.3.2 Aufbau von STEP-Dateien . . . . .	17
3.3.3 Beschreibung von STEP-Entitäten . . . . .	18
3.4 Programmiersprache . . . . .	19
<b>4 Umsetzung des Lösungsweges</b>	<b>21</b>
4.1 Decodierung der STEP-Datei . . . . .	21
4.1.1 Implementierte STEP-Entitäten . . . . .	21
4.1.2 Erläuterung der implementierten STEP-Entitäten . . . . .	23
4.1.3 Bildung einzelner STEP-Entitäten . . . . .	26
4.1.4 Abbildung des gesamten STEP-Modells in Java . . . . .	27
4.2 Diskretisierung der Körperflächen . . . . .	28
4.2.1 Basistransformation . . . . .	29
4.2.2 Diskretisierung zylindrischer Flächen . . . . .	32
4.2.3 Diskretisierung planarer Flächen . . . . .	34

4.3	Deformation der Körperflächen . . . . .	37
4.4	Ausgabe der verformten Geometrie . . . . .	39
4.5	Erläuterung eines beispielhaften Programmablaufs . . . . .	39
<b>5</b>	<b>Fazit und Ausblick</b>	<b>42</b>
<b>A</b>	<b>Anhang</b>	<b>44</b>
A.1	STEP-Beschreibung eines Quaders . . . . .	44
	<b>Abbildungsverzeichnis</b>	<b>52</b>
	<b>Literatur</b>	<b>53</b>
	<b>Selbstständigkeitserklärung</b>	<b>54</b>

## Aufgabenstellung

Alle realen Fertigungsverfahren erzeugen gewisse charakteristische geometrische Abweichungen von der Nenngestalt, die zum Beispiel einem zeitlichen Trend oder einer zufälligen Auftretenswahrscheinlichkeit folgen. Ursache für diese Abweichungen sind dem jeweiligen Fertigungsverfahren innewohnende, teils unvermeidliche Erscheinungen. Zu diesen gehören zum Beispiel der Verschleiß des Werkzeugs oder äußere Schwingungen.

Die beherrschte Vorhersage solcher Abweichungen ist bedeutend zur Absicherung der Produktqualität über die gesamte Produktionsstückzahl. Die Kenntnis über die sich wahrscheinlich einstellenden Ist-Abweichungen der Nenngestalt ist wichtig für verschiedene Prozesse der Fertigungsmesstechnik, wie zum Beispiel zur Definition der Messstrategie (Messpunkt muster, Messpunktanzahl, etc.).

Ziel der Abschlussarbeit ist einerseits die umfassende Zusammenstellung und Kategorisierung charakteristischer geometrischer Fertigungsabweichungen. Darauf aufbauend ist ein geeignetes Werkzeug zu generieren, um die zu erwartenden Abweichungen durch eine erzeugte Punktwolke darzustellen. Das geschieht, indem die vorhandene Soll-Geometrie durch einen geeigneten Algorithmus mit den erarbeiteten typischen Fertigungsfehlern beaufschlagt werden können. Dafür wird die Umsetzung mittels MATLAB oder einer geeigneten CAD-Software vorgeschlagen.

### Teilaufgaben

- Recherche und Katalogisierung von charakteristischen geometrischen Abweichungen, die aus realen Fertigungsverfahren hervorgehen können
- Erarbeitung eines Werkzeugs zur Erzeugung von erwartbaren Messpunktfolgen durch die Beaufschlagung der Soll-Geometrie mit geometrischen Fertigungsabweichungen, zum Beispiel mittels MATLAB oder CAD
- Zusammenstellung der Erkenntnisse und Erarbeitung eines Ausblicks

## 1 Einführung

## 2 Grundlagen

### 2.1 Oberflächenabweichungen und -unvollkommenheiten

Um die Qualität einer technischen Oberfläche beurteilen zu können, ist es zunächst unerlässlich, verschiedene standardisierte Begriffe zu definieren. Dazu geben sowohl DIN 4760 [1] als auch DIN EN ISO 8785 [2] diverse Bezeichnungen vor, welche die möglichen auftretenden Oberflächenunvollkommenheiten voneinander abgrenzen und ordnen sollen.

#### 2.1.1 Wirkliche Oberfläche

DIN 4760 definiert den Begriff der Wirklichen Oberfläche als die tatsächliche, den betrachteten Gegenstand von seinem Umgebungsmedium trennende Oberfläche. Für die Messtechnik ist diese Gestalt allerdings nicht in voller Gänze zu erfassen. Aus diesem Grund werden weitere Begriffe benötigt, um die Wirkliche Oberfläche zu abstrahieren und für die Messtechnik verfügbar zu machen.

#### 2.1.2 Istoberfläche

Als Istoberfläche bezeichnet die oben beschriebene Norm das messtechnisch erfasste Abbild der Wirklichen Oberfläche eines Formelementes. Dabei ist festzuhalten, dass hier von einer bereits vereinfachten Oberfläche zu sprechen ist. Abhängig von Messverfahren, Messparametern, Messplan und Messabfolge ergeben sich für verschiedene Messungen auch unterschiedliche Istoberflächen des Objektes. Dies ist bei der Betrachtung und Analyse einer Istoberfläche stets zu beachten. Natürlich wirken sich auch systematische und zufällige Messfehler auf die Gestalt der Istoberfläche aus. Um von der messtechnisch erfassten Gestalt des Bauteils auf dessen Gestaltabweichungen schließen zu können, ist der nachfolgende Begriff der Geometrischen Oberfläche unerlässlich.

#### 2.1.3 Geometrische Oberfläche

Als Geometrische Oberfläche wird in DIN 4760 die ideale Form des betrachteten Objektes definiert. Sie wird auch als Nennform bezeichnet und ist durch die jeweiligen technischen Zeichnungen oder andere technische Unterlagen, wie zum Beispiel 3D-CAD-Informationen, festgeschrieben. Da kein Fertigungsprozess fehlerfrei abläuft, ist es unmöglich, diese Idealgeometrie in der Realität zu fertigen. Allerdings ist es mit Hilfe der technischen Dokumente möglich, zulässige

Abweichungen von der Idealgeometrie festzulegen. Vielmehr dient diese Beschreibung in der Messtechnik als Referenz, um die im folgenden Abschnitt behandelten Gestaltabweichungen überhaupt erst erfassbar zu machen.

#### 2.1.4 Gestaltabweichungen

Als Gestaltabweichungen werden in der beschreibenden Norm die Gesamtheit aller Abweichungen der messtechnisch erfassten Istoberfläche von der idealen Geometrischen Oberfläche bezeichnet. Grundlage für das Vorhandensein von Gestaltabweichungen sind mannigfaltig und werden in den folgenden Abschnitten weiter ausgeführt. Grundsätzlich ist festzustellen, dass unterschieden wird zwischen solchen Abweichungen, die nur durch Betrachtung des gesamten Objektes feststellbar sind, und Abweichungen, die nur durch Analyse eines Ausschnittes der Oberfläche erkennbar werden (siehe Abbildung 1). Dazu werden die Gestaltabweichungen in 6 verschiedenen Ordnungen klassifiziert.

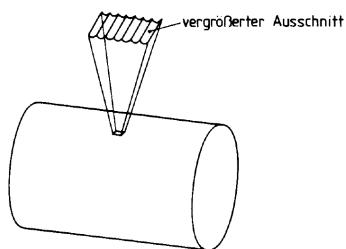


Abbildung 1: DIN 4760 - Ausschnitt aus der Istoberfläche

Als Gestaltabweichungen 1. Ordnung werden in DIN 4760 jene Gestaltabweichungen beschrieben, die, wie weiter oben bereits angeschnitten, bei der Beurteilung der gesamten Istoberfläche ersichtlich werden. Abweichungen dieser Kategorie werden als Formabweichungen bezeichnet. Darunter zählen beispielsweise Geradheits-, Ebenheits- und Rundheitsabweichungen. Diese Art der Abweichungen begründet die Norm in fehlerhaften Führungen der bearbeitenden Maschine, Durchbiegung von Werkzeug oder Werkzeugmaschine, falscher Einspannung des Werkstückes, Verschleiß oder Härteverzug.

Die 2. Ordnung der Gestaltabweichungen wird unter dem Begriff Welligkeit geführt. Es handelt sich dabei um periodisch wiederholt auftretende Gestaltabweichungen der gemessenen Oberfläche eines untersuchten Formelements. Als wellig definiert die Norm DIN 4760 Abweichungen, bei denen das Verhältnis von Wellenabständen zur Wellentiefe generell zwischen 1000 : 1 und 100 : 1 liegt. Es handelt sich also um relativ niederfrequente Wellen mit einer verhältnismäßig niedrigen Amplitude. Als grundlegende Ursachen für die Entstehung von welligen Formen nennt die Norm sowohl außermittige Einspannung des Werkstücks als auch Form- und Laufabweichungen des Werkzeugs und Schwingungen der Werkzeugmaschine und des Werkzeugs.

Die Gestaltabweichungen 3. bis 5. Ordnung werden als Rauheit bezeichnet. Sie sind nur durch Betrachtung eines Oberflächenausschnitts des Objektes zu erkennen. Rauheitsabweichungen kehren entweder regelmäßig oder unregelmäßig wieder. Sie sind gekennzeichnet durch ein Verhältnis der Abstände zur Tiefe zwischen 100 : 1 und 5 : 1. Es handelt sich also im Vergleich zum im vorherigen Abschnitt beschriebenen Welligkeitsphänomen um hochfrequenter Abweichungen der Oberfläche. Innerhalb der Klasse der Rauheitsabweichungen lassen sich verschiedene Ausprägungen beschreiben. So unterscheidet die Norm Rillen (4. Ordnung), Riefen, Schuppen, Kuppen (5. Ordnung) und Oberflächenrauheit, welche durch die Gefügestruktur hervorgerufen wird (6. Ordnung). Diese Kategorien unterscheiden sich in Frequenz und Amplitude ihrer Form. Außerdem lassen sich verschiedene Ursachen für ihr Auftreten finden.

Rillen werden durch die Form der Werkzeugschneide sowie aufgrund des eingestellten Vorschubes und der Schnittbewegung des Werkzeuges hervorgerufen. Riefen, Schuppen und Kuppen sind Oberflächenunvollkommenheiten, deren Gründe im Prozess der Spanbildung zu finden sind. Des Weiteren lassen sie sich beispielsweise durch Werkstoffverformung beim Strahlen oder Knospenbildung bei galvanischer Behandlung erklären.

Rauheiten, welche durch die Gefügestruktur des Werkstoffes verursacht werden, sind durch Kristallisationsvorgänge oder Veränderungen der Oberfläche beispielsweise durch chemische oder korrosive Vorgänge entstanden. Sie bilden zusammen mit der 6. Ordnung der Gestaltabweichungen, welche durch den Gitteraufbau des Materials zu erklären sind, die nicht mehr durch Messverfahren ermittelbaren Gestaltabweichungen.

Das bedeutet, dass in der Istoberfläche für gewöhnlich eine Struktur zu erkennen ist, die durch Überlagerung der Gestaltabweichungen 1. bis 4. Ordnung verursacht wird.

### 2.1.5 Oberflächenunvollkommenheiten

Zusätzlich zu den in DIN 4760 aufgeführten Gestaltabweichungen werden in DIN EN ISO 8785 Oberflächenunvollkommenheiten erläutert. In der Norm werden diese als „Element, Unregelmäßigkeit oder Gruppe von Elementen und Unregelmäßigkeiten der wirklichen Oberfläche, die unbeabsichtigt oder zufällig durch die Bearbeitung, Lagerung oder Funktion der Oberfläche entstanden sind“ definiert. Es wird dabei unterschieden zwischen nach innen orientierten (Vertiefung), nach außen gerichteten (Buckel), kombinierten und stellenweisen Oberflächenunvollkommenheiten. Kombinierte Oberflächenunvollkommenheiten enthalten sowohl Anteile von Vertiefungen als auch Buckeln. Stellenweise Unvollkommenheiten besitzen eine kaum messbare Erhöhung bzw. Vertiefung.

In Abbildung 2 auf der nächsten Seite sind einige Beispiele für jede genannte Kategorie der

Oberflächenunvollkommenheiten zu finden.

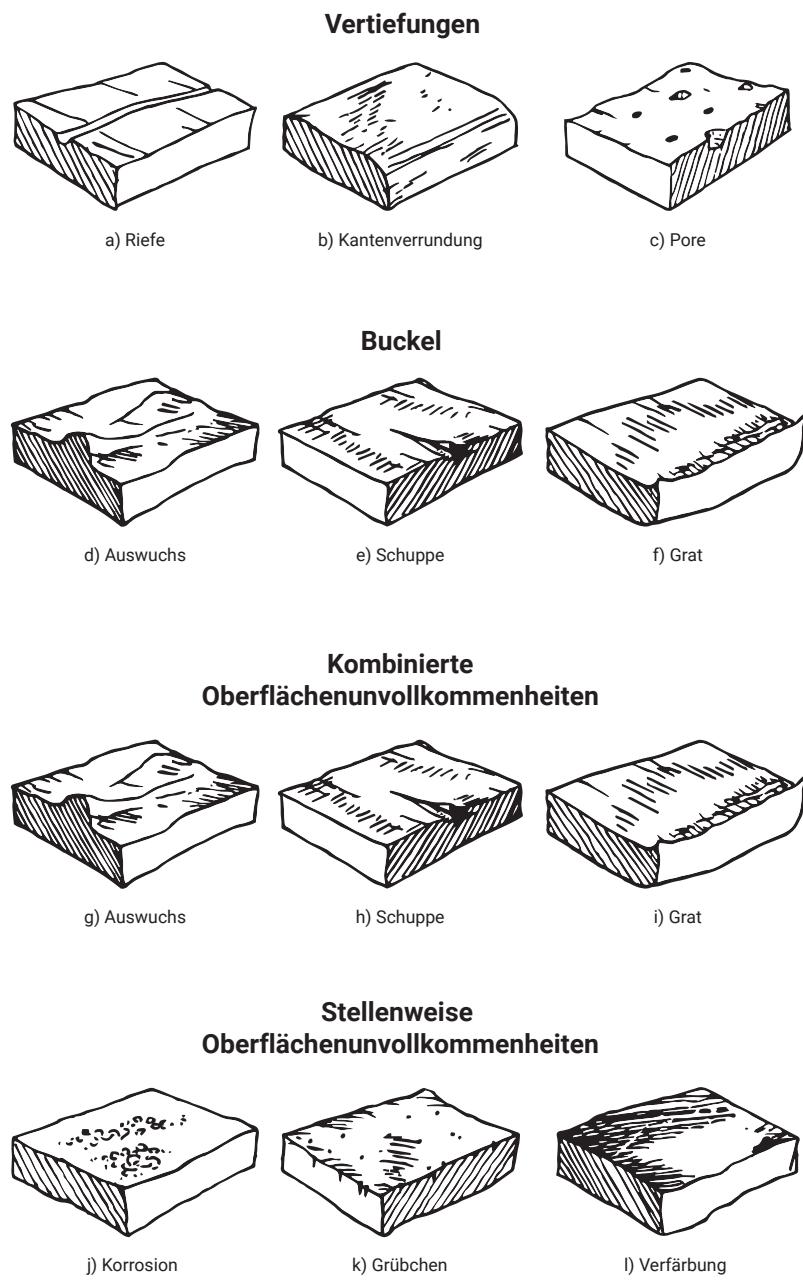


Abbildung 2: DIN EN ISO 8785 - Oberflächenunvollkommenheiten

## 2.2 Typische Abweichungen bei verschiedenen spanenden Fertigungsverfahren

### 2.2.1 Ursachen für Fertigungsabweichungen

Jedes Bauteil, welches in einem Fertigungsverfahren hergestellt wird, entspricht in seiner Realgestalt nicht der in den Fertigungsunterlagen (Technische Zeichnung oder CAD-Modell) festgehaltenen Idealgeometrie. Es sind durch verschiedene Einflüsse stets Abweichungen von der

Geometrischen Oberfläche des Objektes festzustellen. Dies hat nach [3] verschiedene Ursachen, die im Werkstück selbst, dem bearbeitenden Werkzeug, der Werkzeugmaschine oder der Fertigungsumgebung liegen.

So werden als Ursachen für Fertigungsabweichungen, die durch das Werkzeug bestimmt werden, bereits vorhandene Form- und Lagetoleranzen des Rohteils, Festigkeitsunterschiede der abzuspanenden Teile, Auslösung bzw. Einbringung von Eigenspannungen in das Werkstück und örtlich und zeitlich veränderliche Temperaturfelder im Bauteil genannt. Es ist festzuhalten, dass Wärmeeinbringung am Werkstück vor allem Formfehler zur Folge hat. Bei zylindrischen Bauteilen zeigen sich diese hauptsächlich in axialer Richtung wegen inhomogener elastischer Erwärmung.

Weiterhin hat auch das bearbeitende Werkzeug einen sehr großen Einfluss auf die Wirkliche Geometrie des Bauteils. Hier sind besonders die Nachgiebigkeit des Werkzeugs bzw. Werkzeughalters, die Lageabweichung des Werkzeugs beim Werkzeugwechsel sowie der Verschleiß des Werkzeugs zu nennen. Dabei hat der Werkzeugverschleiß einen maßgeblichen Einfluss auf die Maß- und Formgenauigkeit. Werkzeugverschleiß führt zu Schneidenversatz, dies wiederum begründet die Fehler, die die Maßhaltigkeit betreffen. Die Formabweichungen sind in den durch den Werkzeugverschleiß auftretenden höheren Schneidkräften bedingt. Diese führen zusammen mit der Nachgiebigkeit der Werkzeugmaschine zum Auftreten dieser Abweichungen.

Der Einfluss der bearbeitenden Maschine liegt in ihrer Nachgiebigkeit im Kraftfluss und ihrer thermischen Wirkung. So kommt es zu geometrischen Abweichungen und thermisch bedingten Verformungen während des Bearbeitungsprozesses. Ungenauigkeiten in den Führungen der Werkzeugmaschine führen über die kinematische Kette der Maschine zu sich fortpflanzenden geometrischen Fehlern im Bauteil [4].

Auch die Umgebung der Maschine wirkt natürlich auf die Werkstückqualität ein. So führen externe Wärmequellen, Änderungen der Umgebungstemperatur oder eine Veränderung der Kühlshmierung zu Abweichungen am Werkstück. Zählt man den Bearbeiter des Bauteils zu den Umgebungsfaktoren hinzu, ergeben sich auch durch systematische oder zufällige menschliche Fehler in der Einrichtung und Überwachung des Bearbeitungsprozesses Einflüsse auf die Formgestalt des Bauteils.

Es ist festzuhalten, dass ein komplexes System an Einflussfaktoren auf das Werkstück einwirkt, die sich gegenseitig beeinflussen können. Es ist nicht möglich, ohne detaillierte Kenntnis aller Faktoren bereits im Vorfeld konkret vorherzusagen, welche Abweichungen bei einem Fertigungsverfahren zu erwarten sind. Durch Erfahrungswerte lassen sich allerdings typische Abweichungen für die einzelnen Fertigungsverfahren beschreiben.

### 2.2.2 Fertigungsabweichungen beim Drehen

Beim Fertigungsverfahren Drehen sind, je nach konkretem Drehverfahren, unterschiedliche Maßtoleranzen zu erreichen. So sind laut [5] beim Schlichtdrehen Toleranzen von IT7 bis IT8 realistisch, beim Feinschlichten können unter optimalen Drehbedingungen sogar Genauigkeiten von IT6 umgesetzt werden. Das deckt sich mit den Angaben nach [3], wonach mit konventionellen Drehmaschinen Maßabweichungen von IT6 bis IT7 realisierbar sind. Ergänzend dazu werden mögliche Rundheitsabweichungen auf konventionellen Drehmaschinen von weniger als 2,5 µm, Zylindrizitätsabweichungen von weniger als 2 µm und gemittelte Rautiefe  $R_z$  von 2 bis 6 µm beschrieben.

Außerdem wird die mögliche Fertigungsgenauigkeit von Präzisionsdrehmaschinen genannt. Dabei werden Maßabweichungen im Bereich von IT5 bis IT6, Rundheitsabweichungen von 0,2 µm, Zylindrizitätsabweichungen von weniger als 0,1 µm sowie eine gemittelte Rautiefe  $R_z$  von 0,5 µm erreicht. Dies ist also ein deutlich genaueres Verfahren, was bei der Betrachtung der zu erwartenden Abweichungen unbedingt in Erwägung gezogen werden muss.

Die theoretische Oberflächenrauheit einer drehend hergestellten Oberfläche lässt sich laut [5] und [6] mit der Formel

$$R_t = \frac{f^2}{r_\epsilon}$$

berechnen. In dieser Formel beschreibt  $r_\epsilon$  den Eckenradius und  $f$  den Vorschub des Werkzeugs in axialer Richtung. Die Vorschubgeschwindigkeit hat demnach einen quadratischen Einfluss auf die theoretische Oberflächenrauheit und gibt diese maßgeblich vor. Mit zunehmender Nutzungsdauer des Werkzeuges steigt dessen Verschleiß, wobei sich somit der Betrag des Eckenradius stets ändert. Dies resultiert laut oben genannter Formel aus einer Veränderung der Oberflächengüte des bearbeiteten Geometrieelements.

Drehende Verfahren weisen eine stark gerichtete Bearbeitungsrichtung auf, weshalb durchweg gerichtete, rillige Oberflächen entstehen. Diese gerichteten Oberflächen weisen in der Regel quer zur Schnitttrichtung größere Rauheiten als in Schnitttrichtung auf [3]. Diese Rillen sind stark geprägt von der Form und dem Verschleißzustand des Drehmeißels sowie der Vorschubbewegung [6]. Weiterhin ist die Oberflächenrauheit geprägt von der Zerspanbarkeit des Werkstoffes. Eine gute Zerspanbarkeit ermöglicht geringe Fehler in der Form und Oberflächengüte des Bauteils. Werkstoffe mit einer homogenen Gefügeausbildung erreichen eine höhere Arbeitsgüte einfacher als Werkstücke mit Fehlstellen (z. B. Lunker oder Porositäten).

Wie alle Verfahren erzeugt auch das Drehen erfahrungsgemäß typische Fehler am Werkstück,

die konkreten Ursachen zugeordnet werden können. Dies ermöglicht den Rückschluss vom Fehlerbild auf die Ursache und erlaubt eine Anpassung der Prozessparameter, um eine höhere Arbeitsgenauigkeit zu steuern.

Ungenügende Maßgenauigkeit lässt sich zum Beispiel über einen zu hohen Verschleiß des Werkzeuges erklären. Mit übermäßigem Verschleiß wird der Schneidkantenversatz des Drehmeißels zu groß, was eine kontrollierte Einstellung der Oberflächenrauheit erschwert. Eine weitere Ursache für zu hohe Maßabweichungen kann in einer Durchbiegung des eingespannten Werkstückes liegen. Dies resultiert aus zu hohen Passivkräften bei der Zerspanung (zu hohe Schnitttiefe oder Werkzeugvorschub), fehlender Abstützung schlanker Werkstücke oder einem zu hohen Einstellwinkel  $\kappa$  des Werkzeuges [7].

Ein weiterer Fehler, der sich bei drehend hergestellten Bauteilen finden lässt, ist die Unrundheit des Profils außerhalb der vorgegebenen Toleranzen. Ein Grund dafür liegt in der übermäßigen Durchbiegung des Bauteils aus den oben genannten Gründen. Außerdem wirken sich eine ungenaue, außermittige Zentrierung, Längsführungen oder Hauptspindellagerungen mit zu viel Spiel und elastische Verformungen des Werkstückes durch zu hohe Spannkräfte negativ auf die Rundheit des Bauteils aus [5], [7].

Bei Drehteilen lassen sich außerdem wellige Oberflächen feststellen. Dies ergibt sich aus Schwingungen der Drehmaschine und damit auch des Werkzeugs im Eingriff durch zu hohes Spiel in den Führungen der Maschine. Außerdem führen eine falsche Werkzeugeinspannung und zu hohe Schnittleistungen zum Einbringen von zusätzlichen Schwingungen in die Werkzeugmaschine, was die Bildung welliger Oberflächen begünstigt [5].

Rattermarken am Werkstück entstehen aufgrund eines instabilen Werkzeugs (Auskraglänge zu groß, Schaftquerschnitt zu gering oder instabile Einspannung), zu hohen Passivkräften bei der Bearbeitung, einem zu hohen Freiflächenverschleiß oder einer ungenügenden Schneidkantenschärfe des Werkzeugs, beispielsweise wegen einer Beschichtung dessen [7].

Eine konische Form des angestrebten zylindrischen Bauteils entsteht laut [5] aufgrund einer nicht fluchtenden Anordnung von Drehachse der Maschine und Achse der Reitstockspitze.

Kratzer auf der Oberfläche eines Drehteils lassen sich auf Oxidationsverschleiß der Nebenschneide des Werkzeugs oder auf den Kontakt mit Spänen zurückführen, welche die fertig gedrehte Oberfläche beschädigen [7].

### 2.2.3 Fertigungsabweichungen beim Bohren

Beim Fertigungsverfahren Bohren sind verschiedene Maßtoleranzen und Rautiefe zu erreichen. Auch beim Bohren sind die Werte stark von dem Bohrverfahren und den Prozessparametern abhängig. Tabelle 1 stellt den Sachverhalt anschaulich dar:

Verfahren	Maßtoleranz	Rautiefe $R_t$ in $\mu\text{m}$	Oberflächenqualität
Bohren ins Volle	IT12	80	Schruppen
Aufbohren mit Wendelsenkern	IT11	20	Schlichten
Senken mit Flach- und Formsenkern	IT9	12	Schlichten
Reiben	IT7	8	Feinschlitten
Ausdrehen mit Ausdrehmeißel oder mehrschneidigem Bohrkopf	IT7	8	Feinschlitten
Ausdrehen mit Hartmetallschneiden und sehr kleinem Spanungsquerschnitt	IT7	4	Feinschlitten

Tabelle 1: Maßtoleranzen und Rautiefe verschiedener Bohrverfahren (Quelle: [5])

Des Weiteren beschreibt [7] die Abhängigkeit der erreichbaren Genauigkeiten vom Material des Bohrers. So sind mit HSS-Spiralbohrern Toleranzen von IT12, mit Vollhartmetall-Spiralbohrern Toleranzen von IT9 und mit geradgenutetem Vollhartmetallbohrer Toleranzen von IT7 herstellbar.

Generell ist zu sagen, dass ein Bohren ins Volle mit einem Wendelbohrer stets eine Schrubbearbeitung darstellt. Das heißt, es ist keine hochgenaue Fertigung möglich. Viel bessere Genauigkeiten lassen sich mit Reiben, Senken, Feinbohren und Ausdrehen erzielen [7].

Die Formgenauigkeit einer Bohrung wird hauptsächlich durch deren Rundheit und Geradheit beschrieben. Mangelnde Formgenauigkeit kann verschiedene Ursachen haben. Dazu zählen eine nicht ausreichend hohe Steifigkeit von Werkzeugmaschine, Werkzeug und der Werkzeugaufnahme. Weiterhin wirken sich eine schlechte Zentrierung beim Anschnitt, ungleichmäßiges Schneiden aller Hauptschneiden des Bohrers und zu hohe Belastungen (Vorschub- und Schnittkräfte) negativ auf die Formhaltigkeit der Bohrung aus. Positiv können sich hingegen die Verwendung geradgenuteter oder rechts gedrallter Bohrer und eine gute Spanbildung (Spanformung, Spanbruch, Spanabfuhr) niederschlagen.

Von Werkzeugen verursachte Asymmetrien unterscheiden sich deutlich von werkstückbedingten Asymmetrien. Dadurch entstehen typische Bohrfehler.

Durch ungünstige Werkzeuge zu begründende Fehler bohrend hergestellter Flächen sind bei-

spielsweise Überweite und Konizität des Geometrieelements. Dabei sind nach [Paucksch] vor allem durch Anschlifffehler erzeugte asymmetrische Werkzeuge ursächlich. Diese sind charakterisiert durch Hauptschneidenlängenunterschiede, Einstellwinkeldifferenzen, Spitzenlängenabweichungen und Außermittigkeit der Werkzeugquerschneide. Überweite kommt dabei am häufigsten vor. [8] nennt neben dem ungünstigen Anschliff des Bohrers außerdem den Einsatz falscher Bohrerdurchmesser sowie den übermäßig langen und damit verschleißreichen Einsatz von Bohrern als weitere Gründe für eine Überweite der Bohrung. Weiterhin kann es bei stumpfen Bohrern zur Ausbildung einer Wulst an der Oberseite bei gleichzeitiger Gratbildung an der Unterseite kommen. Weiterhin werden durch einen stumpfen Bohrer sehr raue Bohrungswandungen hervorgerufen. Hat das Werkzeug eine schlechte Führung, keine Zentrierspitze oder einen zu geringen Spitzenwinkel, so können in dünnwandigen Werkstücken unrunde Bohrungen entstehen [5].

Auch beim Bohren treten Abweichungen auf, die im Werkstück begründet sind. Ursachen dafür liegen nach [6] unter anderem in schrägen oder unebenen Flächen des Rohteils, auf denen die Bohrung platziert werden soll. Zusätzlich dazu sind schräge Vorbohrungen, dünne Restwandstärken, Hohlräume, Querbohrungen sowie Lunker und Einschlüsse ursächlich dafür, dass es zu einer asymmetrischen Führung des Bohrers kommt. Daraus resultieren einseitige Kräfte auf das Werkzeug, was zu einem Ausweichen des Bohrers und dessen elastischer Verbiegung führt. Im Ergebnis verläuft der Bohrer und es kommt zur Ausbildung von Mittenabweichungen, schrägen Bohrungssachsen oder einer Unrundheit des Geometrieelements. Weiterhin sorgen falsch platzierte Zentrierungen oder Vorbohrungen zu einer inkorrektten Lage der Bohrung.

#### 2.2.4 Fertigungsabweichungen beim Fräsen

Auch beim Fertigungsverfahren Fräsen hängt die erreichbare Maßgenauigkeit und Oberflächenqualität, wie bei allen anderen Fertigungsprozessen auch, stark vom konkret verwendeten Verfahren ab. Tabelle 2 zeigt diese Abhängigkeit.

Verfahren	Maßtoleranz	Rautiefe $R_t$ in µm
Walzenfräsen	IT8	30
Stirnfräsen	IT6	10
Formfräsen	IT7	20-30

Tabelle 2: Maßtoleranzen und Rautiefen verschiedener Bohrverfahren (Quelle: [5])

Es ist, wie oben schon erwähnt, zu erkennen, dass deutlich maßhaltigere Werkstücke als beim klassischen Bohren mit Wendelbohrern herstellbar sind.

Eine ungenügende Oberflächengüte beim Fräsen kann auf verschiedene Prozessparameter zu-

rückzuführen sein. So deutet eine zu hohe Rautiefe auf zu geringe Schnittgeschwindigkeiten, einen zu großen Vorschub je Schneide, eine ungenügende Werkstückspannung, zu große Schnittkräfte oder ein ratterndes Fräserwerkzeug infolge der Einbringung von Schwingungen aus der Maschine hin [5]. [8] ergänzt, dass eine zu große Schneidkantenfase oder eine ungenügende Maschinenstabilität zu nicht ausreichender Oberflächenqualität führen können.

Ein weiteres typisches Fehlerbild, was beim Fräsen auftreten kann, ist das Bilden von Kantenausbrüchen. Dabei wurde der Vorschub pro Zahn zu groß gewählt oder eine zu große Schnitttiefe eingestellt. Weiterhin ist ein zu großer Einstellwinkel und eine zu große Schneidkantenfase maßgebend für diesen Fehler [8].

Zeigt die gefräste Oberfläche Vertiefungen in gleichen Abständen, dann ist nach [5] ein schlagernder Fräser die Ursache.

### 2.2.5 Fertigungsabweichungen beim Räumen

Beim Räumen sind nach [5] und [3] mit Sicherheit Maßgenauigkeiten von IT7 bis IT8 erreichbar. Mit einem erhöhten Aufwand lassen sich sogar Toleranzen von IT6 erzielen.

Die erreichbare Oberflächengüte beim Räumen wird maßgeblich vom letzten Schlichtzahn bestimmt. Normalerweise erreicht man Oberflächentoleranzen  $R_z$  von 6,3 bis 25  $\mu\text{m}$ . Mit besonders hohem Aufwand sind auch Werte von  $R_z = 1 \text{ mm}$  realisierbar. Die Erzielung präziser Oberflächen ist sowohl bei Baustählen als auch bei gut räumbaren Automatenstählen und Gusswerkstoffen möglich. Auch bei normalgeglühten Einsatz- und Vergütungsstählen mit einer gleichmäßigen Ferrit-Perlit-Verteilung sind diese Werte umsetzbar [5], [9].

Ist die Vorbohrung, durch welche die Räumnadel gezogen wird, zu groß, kann es sowohl zum Verlaufen des Werkzeugs als auch zu großen Lageabweichungen des Geometrieelements kommen [3]. Übermäßige Lageabweichungen sind weiterhin typisch für während des Vorganges schwimmend gelagerte Werkstücke und sehr schlanke Teile, die eine geringe Quersteifigkeit aufweisen.

Ist die Auflage des Werkstücks nicht rechtwinklig zur Bohrung, kann es zu Rattermarken mit großen Abständen kommen. Ursache für Quetscherscheinungen an der Auslaufseite des Werkstücks liegen in weichen Stellen innerhalb des Werkstoffes begründet [5].

### 2.2.6 Fertigungsabweichungen beim Schleifen

Kommt es beim Schleifen zu Maßungenauigkeiten, liegt dies an einer zu großen Schleifzugeabe, ungenügender Kühlung, einer nicht ausgewuchteten oder nicht ausreichend abgerichteten Schleifscheibe.

Formfehler treten dann auf, wenn das Werkstück vor dem Schleifen nicht optimal ausgerichtet oder das Werkzeug ungünstig gewählt wurde. Dazu zählen ein zu grobes Korn, eine zu niedrige Härte und ein unzureichend dichtes Gefüge der Schleifscheibe. Außerdem kann auch die Abrichteinheit fehlerhaft sein, was ebenfalls zu Formfehlern am Bauteil führt [8].

Weist das Bauteil eine zu große Rautiefe auf, dann liegt das nach [8] in einer der folgenden Ursachen begründet. Sowohl eine zu niedrige Schnittgeschwindigkeit, eine zu hohe Einstechgeschwindigkeit der Scheibe, eine zu geringe Ausfeuerzeit der Scheibe, eine zu hohe Korngröße, ein zu geringer Schmierölanteil, eine zu hohe Abrichtgeschwindigkeit als auch eine zu hohe Breite des Abrichtwerkzeugs wirken sich negativ auf die erreichbare Oberflächentoleranz aus.

Ein typischer Fehler, der beim schleifenden Bearbeiten auftritt, sind die sogenannten Schleifriefen. Ursächlich dafür sind ein zu grober Schleifkörper und eine zu geringe Ausfeuerzeit der Schleifscheibe [5]. Ergänzend dazu nennt [8] ein zu hohes Aufmaß des Werkstücks, einen zu niedrigen Fettanteil des Schmiermittels, unzureichende Kühlung und eine zu harte Schleifscheibe. Weiterhin sind eine zu hohe Einstechgeschwindigkeit, eine zu hohe Schnittgeschwindigkeit, ein zu großer Abrichtbetrag, eine zu niedrige Abrichtgeschwindigkeit und ein zu breites Abrichtwerkzeug für dieses Fehlerbild verantwortlich.

Weiterhin sind Rattermarken typisch für geschliffene Oberflächen. Eine zu harte Schleifscheibe und in den Prozess eingebrachte Schwingungen verursachen diese. Diese Schwingungen können zum Beispiel durch Führungsbahnen der Maschine mit zu viel Spiel oder eine nicht ausreichend ausgewuchtete Schleifscheibe entstehen und so ungedämpft in den Zerspanungsvorgang eingebracht werden. Eine zu hohe Einstechgeschwindigkeit, ein zu großer Vorschub und eine zu hohe Schnittgeschwindigkeit begünstigen ebenso das Auftreten von Rattermarken auf der geschliffenen Oberfläche.

### 3 Lösungsansatz

#### 3.1 Lösungsidee

Um die gestellte Aufgabe lösen zu können, soll eine Software erstellt werden, die in der Lage ist, eine gegebene Idealgeometrie mit ausgewählten Verformungsfunktionen zu beaufschlagen. Als Ergebnis soll die entsprechend deformierte Geometrie ausgegeben werden.

Dafür sind vier Hauptschritte notwendig:

1. Zerlegung der Eingabegeometrie in einzelne Geometrieelemente
2. Erstellung einer idealen Punktewolke für jedes Geometrieelement
3. Verschiebung dieser Punkte entsprechend der Deformationsfunktionen
4. Ausgabe der veränderten Punktewolken jedes Geometrieelements

#### 3.2 Beschreibung der Lösungsschritte

Die Eingabegeometrie soll vom Nutzer in Form eines CAD-Modells bereitgestellt werden. Dabei handelt es sich um eine Datei im STEP-Format. Die Eingabegeometrie beschreibt die Idealgeometrie des zu betrachtenden Bauteils.

Diese Idealgeometrie soll im Anschluss durch das entwickelte Programm in Form einer Punktewolke diskretisiert werden. Das ist erforderlich, weil die Beschreibung der Geometrie durch das Programm in einer kontinuierlichen Form erfolgt. Messtechnisch sind allerdings die Raumkoordinaten einzelner Messpunkte von Bedeutung. Eine Diskretisierung kann verschieden detailliert vonstatten gehen. Um eine variable Anpassung des Programms an die Erfordernisse des Nutzers zu ermöglichen, soll die Anzahl der ausgegebenen Ergebnispunkte steuerbar sein. Der Benutzer wird die Möglichkeit erhalten, die dafür notwendigen Diskretisierungsparameter innerhalb des Programms für jedes Geometrieelement selbst zu bestimmen. Dadurch kann die für den jeweiligen Anwendungsfall erforderliche Dichte der Punktewolke angepasst werden.

Es notwendig, dass jeder einzelne Punkt der entstandenen Punktewolke dem jeweiligen Geometrieelement, zu dem er gehört, zugeordnet werden kann. Diese Anforderung ergibt sich, da nach erfolgter Diskretisierung der Geometrieelemente der Nutzer für jede separate Fläche des Körpers die gewünschten Deformationen in Form von Funktionen beaufschlagen können soll. Dadurch werden die einzelnen Elemente der Punktewolke so verändert, dass sie als Ergebnis die verformte Geometrie repräsentieren. Das Programm soll die Punktewolke des verformten

Bauteils ausgeben. Dies kann als Ausgangspunkt für weitere Untersuchungen im Rahmen einer Toleranzsimulation verwendet werden.

### 3.3 Eingabegeometrie

#### 3.3.1 Verwendung von STEP-Dateien

Die Eingabegeometrie soll durch den Nutzer in Form einer STEP-Datei erfolgen. STEP ist ein nach ISO 10303-21 standardisiertes Austauschformat für 3D-Objektdaten. Die STEP-Datei enthält verschiedene Informationen, zu denen vor allem der geometrische Aufbau des beschriebenen Objektes zählt. Aber auch weitere Informationen, wie Beschreibungen zur grafischen Repräsentation, physikalische Daten des Materials oder Angaben zum Produktlebenszyklus sind in einer Datei hinterlegt.

Die Generierung einer STEP-Datei ist mit praktisch jedem CAD-System möglich. Dazu muss die native CAD-Datei im .stp-Format gespeichert werden.

Eine STEP-Datei enthält die Informationen im ASCII-Format. Das macht die Datei auch für den Menschen leicht lesbar und verständlich. Es besteht die Möglichkeit, eine STEP-Datei in einem beliebigen Texteditor zu öffnen und zu betrachten.

Der hauptsächliche Vorteil von STEP-Dateien für das in dieser Arbeit zu entwickelnde Programm liegt in seinem objektorientierten Aufbau. Dadurch können die Elemente, aus denen jedes Geometrieelement des Bauteils aufgebaut ist, aus der Datei extrahiert werden. Dies ist mit den anderen Austauschformaten, wie zum Beispiel IGES oder STL nicht möglich.

#### 3.3.2 Aufbau von STEP-Dateien

Anhang A.1 zeigt den beispielhaften Aufbau einer STEP-Datei. Diese Datei beschreibt einen einfachen Quader mit den Ausmaßen 10 mm in x-Richtung, 15 mm in y-Richtung und 30 mm in z-Richtung.

Aus diesem Codebeispiel ist gut der zeilenweise Aufbau einer STEP-Datei ersichtlich.

Eine STEP-Datei ist grundsätzlich aus zwei Sektionen aufgebaut. In der HEADER-Sektion, die am Anfang jeder Datei mit dem Schlüsselwort „HEADER“ eingeleitet wird, stehen Meta-Informationen beispielsweise zum Namen, Ersteller oder Zeitstempel der Datei. Beendet wird diese Sektion mit dem Schlüsselwort „ENDSECTION“.

Der zweite Abschnitt besteht aus der sogenannten *DATA*-Sektion. Beginnend mit dem Schlüsselwort „*DATA*“, sind darin die eigentlichen Informationen zur Beschreibung des Objektes zu finden. Für diese Arbeit sind grundsätzlich die Informationen, welche den hierarchisch gegliederten geometrischen Aufbau des beschriebenen Objektes aufzeigen, von Interesse. Auch diese Sektion der Datei wird mit dem Schlüsselwort „*ENDSECTION*“ abgeschlossen. Im *DATA*-Abschnitt sind die Informationen als sogenannte *STEP-Entities* angegeben. Diese Entities (engl. für Entität, Einheit, Informationsobjekt) sind in der EXPRESS-Auszeichnungssprache verfasst. Jede Einheit wird mit einem „#“-Symbol eingeleitet und einem Semikolon abgeschlossen. Die Zahl nach dem „#“-Symbol stellt einen eindeutigen Bezeichner für diese Entität dar. Dieser tritt nur einmal pro Datei auf und dient der Identifizierung einer bestimmten Instanz der Entität. Es ist zu vermerken, dass sich dieser aber, bei verschiedenen Durchgängen der Erzeugung der *STEP*-Datei aus dem nativen Format, ändern kann. Des Weiteren ist die Klasse, zu der die Entität gehört, direkt nach dem Gleichheitszeichen zu finden. In den folgenden Klammern sind die zur Instanz gehörenden Argumente aufgeführt. Dabei ist das erste Argument stets der Name der Entität.

### 3.3.3 Beschreibung von *STEP-Entitäten*

Um die grundsätzliche Beschreibung von geometrischen Entitäten in *STEP* Dateien aufzuzeigen, sollen zwei simple Beispiele herangezogen werden.

Als Minimalbeispiel lässt sich das Codebeispiel [4 auf Seite 44](#) herbeiziehen. Es beschreibt einen Punkt in Kartesischen Koordinaten (Entität *CARTESIAN\_POINT*) mit einer ID von 173 sowie den Koordinaten x = 10,0 mm, y = 15,0 mm und z = 30,0 mm. Weiterhin ist zu erkennen, dass hierbei das Attribut *name* leer ist. Dies ist bei den meisten geometrischen Entitäten der Fall. Das ist allerdings nicht von großer Bedeutung, da, wie oben beschrieben, die ID als Identifikator genutzt wird.

```
#173=CARTESIAN_POINT( ' ', (10 . ,15 . ,30 . )) ;
```

Codebeispiel 1: Beschreibung eines Punktes in *STEP*

Ein etwas komplexeres Beispiel, was die in einer *STEP*-Datei festgehaltene Geometrie-Struktur aufzeigen soll, lässt sich in Codebeispiel [2](#) beobachten. Dieser Ausschnitt aus einer *STEP*-Datei zeigt die Beziehung zwischen den Entitäten.

```
#38=LINE( ' ', #183 ,#50 ) ;  
...  
#50=VECTOR( ' ', #154 ,10 . ) ;  
...
```

```
#154=DIRECTION('',(1.,0.,0.));
...
#183=CARTESIAN_POINT('',(0.,0.,30.));
```

Codebeispiel 2: Beschreibung einer Linie in STEP

Hierbei wird eine Linie im Raum entsprechend beschrieben, deren Anfangspunkt sich bei  $x = 0$  mm,  $y = 0$  mm und  $z = 0$  mm befindet. Der Richtungsvektor dieser Linie zeigt in  $x = 1$  mm,  $y = 0$  mm und  $z = 0$  mm und hat eine Länge von 10 mm.

Es ist zu erkennen, dass sich die Entität der Klasse „LINE“ (ID = 38) zusammensetzt aus zwei weiteren Entitäten. Die Beschreibung der Linie referenziert IDs der Instanzen der Klasse *VECTOR* (ID = 50) sowie *CARTESIAN\_POINT* (ID = 183). Der referenzierte Vektor bezieht sich wiederum auf eine Instanz der Klasse *DIRECTION* (ID = 154), welche die Richtung des Vektors ( $x = 1$  mm,  $y = 0$  mm,  $z = 0$  mm) beschreibt. Der Vektor hat, gegeben durch sein zweites Argument, eine Länge von 10,0 mm.

Die Geometrie, die in einer STEP-DATEN-DATEI beschrieben wird, setzt sich also, wie oben gezeigt, aus einzelnen Elementen zusammen. Es ist eine eindeutige und definierte Struktur zu erkennen, die ineinander verschachtelt ist. Somit ist gewährleistet, dass diese Struktur im Programm abgebildet werden kann, damit später diskretisierte Punkte den jeweiligen Flächen, die ebenfalls aus anderen Objekten bestehen, eindeutig zugeordnet werden können. In Abschnitt 4 auf Seite 26 wird detailliert darauf eingegangen, wie aus den zeilenweisen Beschreibungen der einzelnen Entitäten der STEP-Datei Java-Objekte erzeugt werden.

### 3.4 Programmiersprache

Das Programm wird in der Programmiersprache Java<sup>1</sup> umgesetzt. Das hat verschiedene Vorteile. Zum Einen sind kompilierte Programme dieser Sprache auf allen Systemen lauffähig. Das macht die Anwendung für eine Vielzahl von Anwendern nutzbar. Perspektivisch können damit sowohl stationäre Computer als auch mobile Endgeräte bedient werden.

Des Weiteren basiert die Verwendung der Programmiersprache Java auf einer kostenlosen Distribution. Das bedeutet, dass keine kostenpflichtigen Programme, wie beispielsweise CADSysteme oder MATLAB, für die Entwicklung und Ausführung der Software benötigt werden. Dadurch können auch Anwender, die nicht über Lizenzen für entsprechende Programme verfügen, das entwickelte Programm nutzen. Auch dieser Aspekt trägt dazu bei, notwendige Voraussetzungen für den Anwender zu minimieren.

---

<sup>1</sup>Webseite: <https://www.java.com/de/>

Auf technischer Seite besteht der hauptsächliche Vorteil von Java im durchgängig objektorientierten Paradigma der Programmiersprache. Da eine STEP-Datei ebenfalls aus einzelnen Objekten zusammengesetzt ist, lässt sich diese Struktur sehr gut in Java-Klassen abbilden. So bleibt der Aufbau der STEP-Geometrie im Programmcode erhalten und nachvollziehbar. Weiterhin ist es mit Java sehr gut und relativ einfach umsetzbar, den erstellten Quellcode zu testen. Dies ermöglicht eine automatisierte Kontrolle der einzelnen Einheiten des Programms. Dadurch soll die Qualität und Stabilität der Implementierung garantiert werden. Die Entwicklung des Programms kann so schneller voranschreiten, da händische Tests durch den Entwickler sehr zeitraubend und unsicher sind. Gerade bei Veränderungen oder Erweiterungen des Programms bieten Tests eine Grundlage für die Verifizierung der Softwaregüte und helfen bei der Erkennung von Fehlern.

## 4 Umsetzung des Lösungsweges

### 4.1 Decodierung der STEP-Datei

Aus dem in Abschnitt 3.3 beschriebenen zeilenweisen Aufbau einer STEP-Datei lassen sich die STEP-Entitäten in Java abbilden. Jede Zeile beschreibt genau eine Entität. Diese sind stets nach dem folgenden Muster aufgebaut:

```
#ID=KLASSE(ATTRIBUTSLISTE);
```

Codebeispiel 3: Beschreibung eines Punktes in STEP

Aufgrund dieses einheitlichen Musters lassen sie sich relativ einfach decodieren und als Java-Klassen in Programmcodes umwandeln, um in einem weiteren Schritt die Struktur des Bauteils abzubilden. Das Ziel des ersten Schrittes in der Lösungsumsetzung besteht nun darin, aus einer einzelnen Zeile der STEP-Datei Java-Objekte zu bilden und mit den Daten aus der jeweiligen Zeile zu belegen.

#### 4.1.1 Implementierte STEP-Entitäten

Die verschiedenen Entitätstypen, die in einer STEP-Datei vorhanden sein können, sind in der STEP-Dokumentation [10] aufgeführt. Die Attribute, welche zu jeder einzelnen Klasse gehören, werden darin beschrieben. Da die Implementierung aller vorhandenen STEP-Klassen den Rahmen dieser Arbeit überschreiten würde, sind vorerst folgende Klassen im Programm umgesetzt. Es ist anzumerken, dass nicht jede Klasse eine Instanz bilden kann. Diese sogenannten abstrakten Klassen dienen der Gruppierung von ähnlichen nicht abstrakten Klassen. Diese sind Bestandteil des Vererbungsprinzips der objektorientierten Programmierung in Java. Sie sind in der Liste als solche gekennzeichnet.

- REPRESENTATION\_ITEM (abstrakt)
- GEOMETRIC\_REPRESENTATION\_ITEM (abstrakt)
- TOPOLOGICAL\_REPRESENTATION\_ITEM (abstrakt)
- POINT (abstrakt)
- CARTESIAN\_POINT
- DIRECTION

- VECTOR
- PLACEMENT (abstrakt)
- AXIS2\_PLACEMENT\_3D
- SURFACE (abstrakt)
- ELEMENTARY\_SURFACE (abstrakt)
- PLANE
- CYLINDRICAL\_SURFACE
- CURVE (abstrakt)
- LINE
- CONIC (abstrakt)
- CIRCLE
- VERTEX (abstrakt)
- VERTEX\_POINT
- EDGE (abstrakt)
- EDGE\_CURVE
- ORIENTED\_EDGE
- LOOP (abstrakt)
- EDGE\_LOOP
- FACE\_BOUND
- OUTER\_FACE\_BOUND
- FACE (abstrakt)
- FACE\_SURFACE
- ADVANCED\_FACE

#### 4.1.2 Erläuterung der implementierten STEP-Entitäten

**CARTESIAN\_POINT** Beschreibt einen Punkt im dreidimensionalen Raum mit den Koordinaten x, y und z (in mm), bezogen auf das Weltkoordinatensystem des in der STEP-Datei beschriebenen Bauteils. Diese Punkte dienen als Basis für viele weitere STEP-Entitäten, beziehen sich selbst aber nicht auf andere Objekte.

**DIRECTION** Ebenso wie die *CARTESIAN\_POINTS* beziehen sich diese Entitäten nicht auf andere. Sie repräsentieren einen normalisierten Richtungsvektor (Länge = 1 mm) mit den Richtungskoordinaten x, y, z (in mm). Diese Richtungsvektoren werden von vielen STEP-Entitäten referenziert.

**VECTOR** Stellt einen räumlichen Vektor dar. Dieser hat als Grundgeometrie einen normalisierten Richtungsvektor (*DIRECTION*) und eine gegebene Länge. Ein *VECTOR* bezieht sich demnach, anders als die vorher beschriebenen Klassen, auf ein Objekt einer anderen Klasse.

**AXIS2\_PLACEMENT\_3D** Beschreibt ein lokales Koordinatensystem im Raum. Es besteht aus einem *CARTESIAN\_POINT*, welcher die Position des lokalen Koordinatenursprungs darstellt. Des Weiteren definieren zwei Richtungsvektoren die Ausrichtung dieses lokalen Koordinatensystems.

**LINE** Beschreibt eine gerade Strecke im Raum. Eine *LINE* wird gebildet aus einem *CARTESIAN\_POINT*, welcher den Startpunkt der Linie darstellt, sowie einem *VECTOR*, welcher Richtung und Länge der Strecke bestimmt.

**CIRCLE** Definiert einen Kreis im dreidimensionalen Raum. Die Position (Mittelpunkt) und Ausrichtung des Kreises wird durch ein lokales Koordinatensystem in Form eines *AXIS2\_PLACEMENT\_3D* beschrieben. Dabei steht der erste Vektor des lokalen Koordinatensystems senkrecht auf der Ebene, in welcher der Kreis liegt. Der zweite Vektor liegt in der Kreisebene. Abbildung 3 auf der nächsten Seite verdeutlicht den Aufbau.

**PLANE** Beschreibt eine Ebene im Raum. Diese Ebene hat keine räumliche Begrenzung und ist durch einen *AXIS2\_PLACEMENT\_3D* eindeutig definiert. Ähnlich wie beim *CIRCLE* liegt der erste

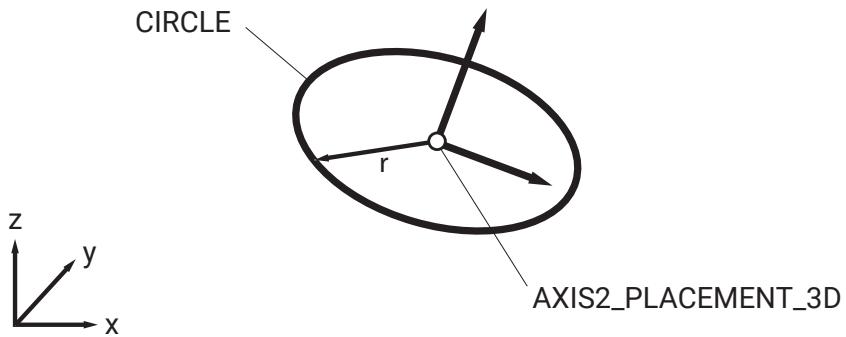


Abbildung 3: Visualisierung *CIRCLE*

Vektor des lokalen Koordinatensystems den Normalenvektor der Ebene fest. Der zweite Vektor beschreibt einen Richtungsvektor der Ebene. Durch den Koordinatenursprung des lokalen Koordinatensystems wird der Ortsvektor der Ebene definiert.

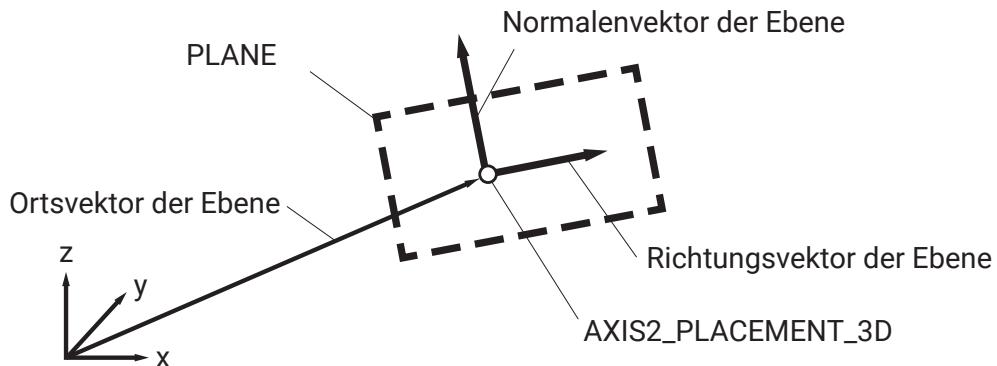


Abbildung 4: Visualisierung *PLANE*

**CYLINDRICAL\_SURFACE** Beschreibt analog zur *PLANE* eine zylindrische Ebene im Raum. Diese kann man sich vorstellen als Mantelfläche eines unendlich langen Zylinders. Die Lage des Zylinders wird auch hier durch eine *AXIS2\_PLACEMENT\_3D* definiert und hat als zusätzlichen Parameter eine Zahlenwertangabe zum Radius. In der folgenden Abbildung ist zu erkennen, dass die Achse des Zylinders aus dem ersten Vektor des lokalen Systems gebildet wird und der zweite Vektor parallel zur Deckfläche des Zylinders liegt.

**VERTEX\_POINT** Definiert einen Punkt des Bauteils im Raum. Dieser wird definiert durch einen *CARTESIAN\_POINT*. Der Unterschied besteht darin, dass es sich bei einem *VERTEX\_POINT* um

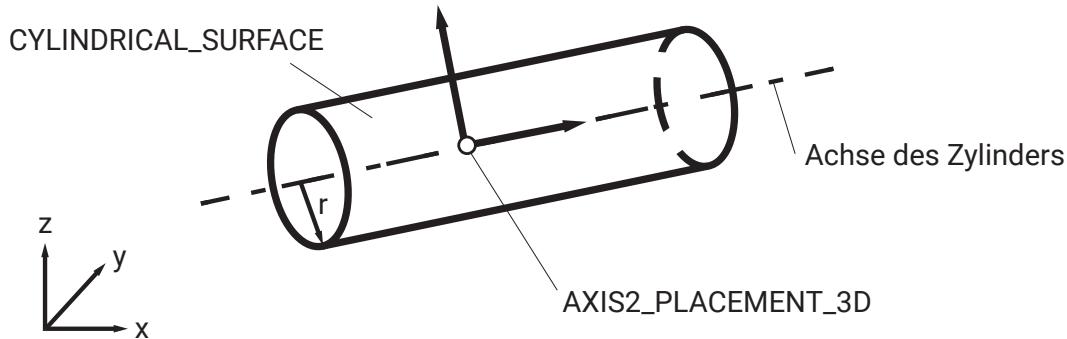


Abbildung 5: Visualisierung CYLINDRICAL\_SURFACE

einen tatsächlichen Punkt des Bauteils (beispielsweise einen Eckpunkt einer Körperkante) handelt und nicht um eine bloße Referenz.

**EDGE\_CURVE** Beschreibt eine Körperkante. Dabei kann es sich um verschiedene Kantenarten, wie Kreise, Linien oder Splines handeln. Definiert werden Anfangs- und Endpunkt der *EDGE\_CURVE*. Diese werden als Körperpunkte (*VERTEX*) definiert. Handelt es sich um eine geschlossene Kante, beispielsweise einen Kreis, dann sind Anfangs- und Endvertex identisch. Außerdem weist eine *EDGE\_CURVE* eine Orientierung auf. Diese wird durch einen boolschen Wert (wahr oder falsch) angegeben und definiert so eindeutig Anfangs- bzw. Endpunkt der Körperkante.

**EDGE\_LOOP** Beschreibt einen Kantenverbund aus mehreren Körperkanten. Dieser Kantenverbund ist geschlossen und enthält als Parameter die Liste der enthaltenen Körperkanten.

**FACE\_OUTER\_BOUND** Beschreibt die äußere Umrandung einer Körperfläche und wird durch eine *EDGE\_LOOP* beschrieben. Dies kann zum Beispiel ein Polygon aus verschiedenen Linien oder Kurvenstücken sein. Solch eine äußere Umrandung kann aber auch durch einen Kreis definiert sein.

**FACE\_BOUND** Definiert die innere Umrandung einer Körperfläche einer *EDGE\_LOOP*. Dies kann, wie bei einer *FACE\_OUTER\_BOUND*, ein Polygon aus verschiedenen Linien oder Kurvenstücken sein. Solch eine innere Umrandung kann aber auch durch einen Kreis definiert sein.

**ADVANCED\_FACE** Beschreibt eine tatsächliche Körperfläche des in der STEP-Datei formulierten Bauteils. Definiert ist die Fläche zum einen durch eine Liste an Umrandungen (BOUNDS) zum anderen durch die Angabe einer PLANE bzw. CYLINDRICAL\_SURFACE, in welcher diese Umrandungen liegen. Diese Körperflächen sind in ihrer räumlichen Ausdehnung begrenzt. Die Menge aller ADVANCED\_FACES einer STEP-Datei ergeben zusammengesetzt das beschriebene Modell des Bauteils.

Abbildung 6 zeigt den Aufbau einer planaren ADVANCED\_FACE aus einer äußeren Umrandung in Form eines Polygons (EDGE\_LOOP aus mehreren LINE-Entitäten) und einer inneren Umrandung durch einen Kreis.

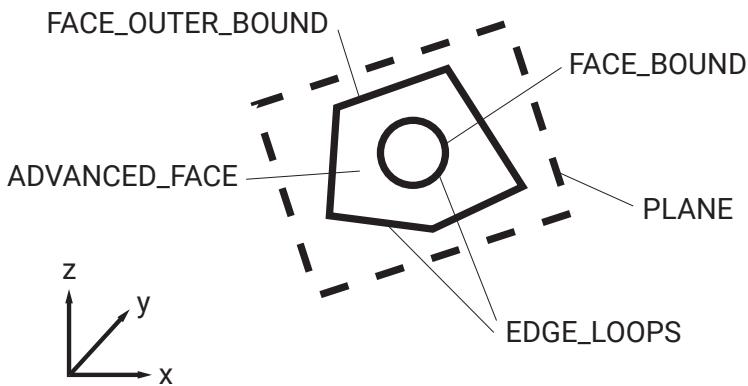


Abbildung 6: Visualisierung planare ADVANCED\_FACE

#### 4.1.3 Bildung einzelner STEP-Entitäten

Nach der Eingrenzung der zu realisierenden STEP-Entitäten und der Erklärung dieser folgt in diesem Abschnitt die Beschreibung, wie aus einer Zeile der STEP-Datei ein konkretes Java-Objekt gebildet wird.

Zuerst müssen alle implementierten STEP-Entitäten als Java-Klassen beschrieben werden. Diese dienen als Muster für die verschiedenen Objekte der STEP-Datei. Die oben aufgeführten Klassen sind im Projektordner „./src/main/java/com.jandoant/stp\_entities“ zu finden. Diese enthalten die entsprechend der STEP-Dokumentation aufgeführten Attribute für jede Entität.

In der Klasse `StpEntityBuilder` im Projektordner „./src/main/java/com.jandoant/builder“ wird dies umgesetzt. Diese Klasse wird im Programmablauf für jede Zeile der STEP-Datei einmal aufgerufen. Übergeben wird ihr die Zeichenkette der jeweiligen Zeile. Aus dieser übergebenen Zeichenkette kann die Klasse nun mit Aufruf der Funktion `extractStpEntity()` ein entsprechendes Objekt der beschriebenen STEP-Entität bilden. Dazu wird die beschreibende Zeichenkette in

ihre drei Bestandteile (ID, Klassenname, Attributliste) zerlegt. Über die Kenntnis des Klassennamens kann die Klasse `StpEntityBuilder` entscheiden, welcher Entitätstyp erzeugt werden soll. In einem weiteren Schritt werden die Elemente der Attributliste extrahiert und ein konkretes Java-Objekt einer STEP-Entität mit all seinen konkreten Attributen kann gebildet werden. Es ist festzustellen, dass es zu diesem Zeitpunkt noch nicht möglich ist, die tatsächlichen Referenzobjekte in der erzeugten Entität festzuhalten. Vorerst kann aus der Zeichenkette nur die jeweilige ID der Referenzobjekte gespeichert werden. Die Zuordnung kann erst erfolgen, wenn alle Zeilen der STEP-Datei in Java-Objekte ausgelesen wurden.

#### 4.1.4 Abbildung des gesamten STEP-Modells in Java

Das Programm ist, wie im vorherigen Abschnitt gezeigt, in der Lage, aus einer einzelnen Zeile der vorliegenden STEP-Datei ein konkretes Java-Objekt zu extrahieren. Nun folgt in diesem Schritt die Decodierung der gesamten Datei. Nach Beendigung dieses Vorgangs ist die komplette geometrische Bauteilstruktur im Programm abgebildet und kann darauffolgend weiter verarbeitet werden.

Der Vorgang dieser Decodierung erfolgt durch die Klasse `StpModelBuilder`, welche sich im Projektordner „`./src/main/java/com.jandoant/builder`“ befindet. Wird diese Klasse im Programmablauf instanziiert, so wird ihr der absolute Pfad, unter dem die zu verarbeitende STEP-Datei liegt, übergeben. Mit dieser Information hat die erzeuge Instanz den Zugriff auf alle Zeilen der Datei, welche die Bauteilgeometrie beschreiben. Mit Aufruf der Funktion `parseFile()` wird die übergebene STEP-Datei verarbeitet. Dabei werden zunächst alle Zeilen der Datei durchlaufen. Jede Zeichenkette, die eine STEP-Entität beschreibt, also mit dem #-Symbol beginnt und mit einem Semikolon endet, wird in einer Liste dieser Zeichenketten zwischengespeichert. Es ist dabei unerheblich, ob die Zeichenkette durch einen oder mehrere Zeilenumbrüche getrennt ist. Dies hat den Effekt, dass die HEADER-Sektion herausgefiltert wird. Diese hat keine Bedeutung für die grundlegende Funktion des Programms, da in diesem Abschnitt der Datei keine STEP-Entitäten beschrieben werden.

Aus der erzeugten Liste an Zeichenketten, welche dem gewünschten Muster entsprechen, werden im folgenden Bearbeitungsschritt unter Anwendung der in Abschnitt [4.1.3 auf der vorherigen Seite](#) dargelegten Methode `extractStpEntity` der Klasse `StpEntityBuilder` die jeweilig beschriebenen STEP-Entitäten im Java-Code instanziiert und in einer Liste zwischengespeichert. Dabei ist anzumerken, dass ausschließlich die in der Klasse `StpEntityContract` im Projektordner „`./src/main/java/com.jandoant/stp_entities`“ festgehaltenen STEP-Entitäten decodiert werden. Alle Zeichenketten, die nicht eine solche Entität beschreiben, werden ignoriert. Das führt dazu, dass ausschließlich die geometrischen Informationen des Modells extrahiert werden. An-

dere Angaben, die in der STEP-Datei festgehalten sein können, werden bei dieser Implementierung nicht berücksichtigt, da sie für die grundlegende Funktion des Programmes unerheblich sind.

Nach dem Durchlaufen aller Zeilen und anschließender Instanziierung der Java-Objekte entsprechend der STEP-Beschreibung liegt zu diesem Punkt eine Liste mit allen vorkommenden geometrischen STEP-Entitäten im Programm vor. Allerdings ist damit den Anforderungen noch nicht entsprochen. Nach der Decodierung der einzelnen Zeichenketten haben die Java-Objekte noch keine Relation zueinander. Wie in Abschnitt [4.1.3 auf Seite 26](#) erläutert, sind die Referenzentitäten, auf die sich eine STEP-Entität bezieht, bisher nur mit ihrer ID erfasst. Eine Zuordnung der Objekte passiert nun aus vorliegender Liste der erzeugten Java-Objekte. Dazu wird die Methode `convertFromIds()` aufgerufen, welche in jeder Java-Klasse, die eine STEP-Entität beschreibt, implementiert ist. Mit Aufruf der Methode durch das betreffende Java-Objekt wird die Entitätsliste durchlaufen. Dabei werden alle Entitäten, deren ID mit einer der IDs, die in diesem Java-Objekt hinterlegt sind, herausgefiltert und dem Java-Objekt zugewiesen. Nachdem dieser Vorgang für alle Objekte der Liste ausgeführt wurde, sind alle Referenzen über tatsächliche Java-Objekte hergestellt.

Die im Programmablauf aufgerufene Methode `parseFile()` der Klasse `StpModelBuilder` gibt nun in einem finalen Schritt alle *ADVANCED\_FACES*, die in der STEP-Datei durch Zeichenketten beschrieben wurden, als tatsächliche Java-Objekte an das Programm zurück. Dies geschieht in Form einer Liste. Diese *ADVANCED\_FACES* referenzieren wie oben beschrieben alle Objekte, durch welche sie selbst definiert werden. Somit ist die gesamte Bauteilstruktur im Programm abgebildet und kann im weiteren Ablauf des Programms verarbeitet werden.

## 4.2 Diskretisierung der Körperflächen

Die Diskretisierung der Oberflächen wird unter Einführung eines lokalen Koordinatensystems für jede einzelne Körperfläche (*ADVANCED\_FACE*) realisiert. Ziel ist dabei die Umwandlung der Punkte der Körperflächen, welche durch die STEP-Datei in globalen Koordinaten ausgedrückt sind, in ein lokales Koordinatensystem zu überführen und im Anschluss zu verändern. Dabei werden alle Punkte der jeweiligen Fläche einer Koordinatentransformation unterzogen. So besitzt jeder Punkt neben seiner Beschreibung im Weltkoordinatensystem mit Werten in x-, y- und z-Richtung gleichzeitig auch Definitionen bezüglich eines lokalen Koordinatensystems in u-, v- und w-Richtung. Die folgende Abbildung [7 auf der nächsten Seite](#) verdeutlicht dies.

Das lokale Koordinatensystem ist spezifisch für jede Körperfläche und ermöglicht eine generalisierte Bearbeitung der diskretisierten Punkte. Jede dreidimensionale Körperfläche wird so in eine zweidimensionale Fläche mit Ausdehnung in u- und v-Richtung überführt. Die w-Koordinate,

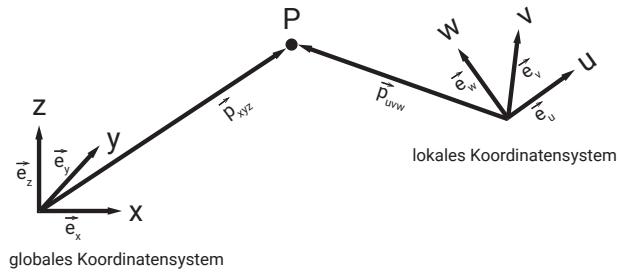


Abbildung 7: Beschreibung eines Punktes in verschiedenen Koordinatensystemen

welche senkrecht zur Fläche zeigt, ist dabei für die Idealgeometrie  $w = 0$ . Dieses Verfahren wird als „UV-Mapping“ bezeichnet. Mit der Anwendung der Deformationsfunktionen ändert sich die Koordinate der Punkte in dieser Richtung. Eine Rücktransformation in das globale Koordinatensystem bildet diese Veränderung an der deformierten Geometrie ab.

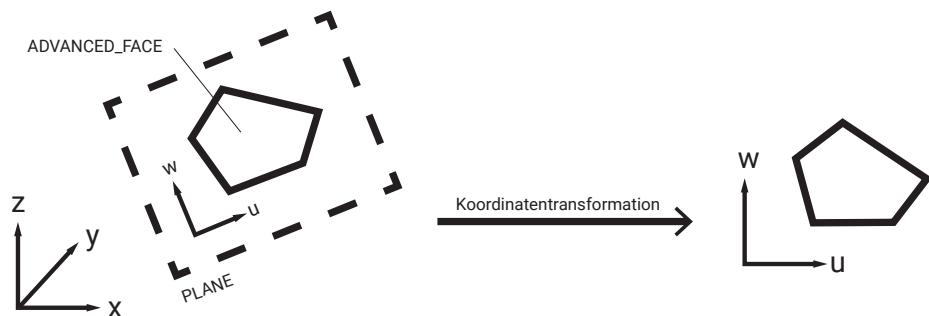


Abbildung 8: UV-Mapping einer Körperfläche

#### 4.2.1 Basistransformation

Um eine Basistransformation zwischen Welt- und lokalem Koordinatensystem durchführen zu können, ist es notwendig, sowohl die Basiseinheitsvektoren des Weltkoordinatensystems als auch die des lokalen Systems zu kennen. Die Basiseinheitsvektoren des globalen Koordinatensystems sind für alle Punkte des STEP-Modells stets dieselben und werden wie folgt definiert.

$$\vec{e}_x = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad \vec{e}_y = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \quad \vec{e}_z = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \quad (1)$$

Die Menge aller Punkte einer Ebene wird in der Geometrie durch die Formel

$$E : \vec{x} = \vec{x}_0 + r \cdot \vec{x}_{R1} + s \cdot \vec{x}_{R2} \quad (2)$$

eindeutig festgelegt. Dabei steht  $\vec{x}$  für einen beliebigen Punkt der Ebene.  $\vec{x}_0$  beschreibt den Ortsvektor,  $\vec{x}_{R1}$  und  $\vec{x}_{R2}$  die Richtungsvektoren und  $r$  und  $s$  sind beliebige Laufparameter, um die Lage des Punktes als Linearkombination der Richtungsvektoren genau zu beschreiben.

Die Position einer *PLANE* wird durch eine *AXIS2\_PLACEMENT\_3D* beschrieben. Daraus lassen sich die Elemente einer Ebenenbeschreibung, wie in Formel 2 dargestellt, auslesen. Dabei entspricht der *CARTESIAN\_POINT* der Positionsbeschreibung dem Ortsvektor. Dieser stellt gleichzeitig den Koordinatenursprung des lokalen Koordinatensystems dar. Die erste *DIRECTION* dieser Positionsbeschreibung entspricht dem normalisierten Normalenvektor der Ebene  $\vec{n}$ , die zweite *DIRECTION* einem normalisierten Richtungsvektor  $\vec{x}_{R1}$ . Da in der Lagebeschreibung der Ebene nur der Normalenvektor und ein Richtungsvektor dieser Ebene beschrieben sind, wird der fehlende Richtungsvektor durch Bildung des Kreuzproduktes der beiden für ein rechtshändiges System nach Formel 3 erzeugt.

$$\vec{x}_{R2} = \vec{n} \times \vec{x}_{R1} \quad (3)$$

Ebenso, wie bei einer ebenen Fläche, die durch die STEP-Entität *PLANE* beschrieben wird, wird auch bei zylindrischen Flächen des Typs *CYLINDRICAL\_SURFACE* die Ausrichtung des lokalen Koordinatensystems in einem *AXIS2\_PLACEMENT\_3D* festgehalten. Dabei unterscheidet sich allerdings die Bedeutung der *DIRECTIONS* der Positionsbeschreibung. Der *CARTESIAN\_POINT* des *AXIS2\_PLACEMENT\_3D* beschreibt dabei die Position des Ortsvektors der Fläche, die erste *DIRECTION* die Richtung der Achse  $\vec{n}$  des Zylinders und die zweite *DIRECTION*  $\vec{x}_{R1}$  liegt parallel zur radialen Richtung des Zylinders. Mit Hilfe der Formel 3 lässt sich auch hier ein weiterer Vektor bilden, welcher rechtshändig orientiert senkrecht zur Achse und  $\vec{x}_{R1}$  ebenfalls in radiale Richtung des Zylinders zeigt.

Die lokalen Basiseinheitsvektoren

$$\vec{e}_u = (e_{u_x} \quad e_{u_y} \quad e_{u_z})^T, \vec{e}_v = (e_{u_x} \quad e_{u_y} \quad e_{u_z})^T \text{ und } \vec{e}_w = (e_{u_x} \quad e_{u_y} \quad e_{u_z})^T$$

können somit entsprechend Formel 4 gebildet werden. Darüber wird definiert, in welche Richtung u-, v- und w-Koordinaten jeweils zeigen.

$$\vec{e}_u = \vec{x}_{R1} \quad \vec{e}_v = \vec{x}_{R2} \quad \vec{e}_w = \vec{n} \quad (4)$$

Sind die Basiseinheitsvektoren der beiden Koordinatensysteme bekannt, so lassen sich die jeweiligen Basismatrizen durch spaltenweise Aneinanderreihung der Vektoren bilden. Die Basis des Weltkoordinatensystems ergibt sich damit zur Basismatrix  $B_{XYZ}$  nach Formel 5.

$$B_{XYZ} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (5)$$

Die Basis des Weltkoordinatensystems ergibt sich analog zur Basismatrix  $B_{UVW}$  nach Formel 6.

$$B_{UVW} = \begin{pmatrix} e_{u_x} & e_{v_x} & e_{w_x} \\ e_{u_y} & e_{v_y} & e_{w_y} \\ e_{u_z} & e_{v_z} & e_{w_z} \end{pmatrix} \quad (6)$$

Die Transformationsmatrix, um einen Punkt vom globalen in das lokale Koordinatensystem der Ebene zu transformieren, ergibt sich nach Formel 7.

$$T_{xyz \rightarrow uvw} = B_{uvw}^{-1} \cdot B_{xyz} \quad (7)$$

Die Transformationsmatrix, um einen Punkt wieder aus dem lokalen Koordinatensystem der Ebene zurück in das globale System zu überführen, ergibt sich nach Formel 8 aus der Inversen der in Formel 7 berechneten Transformationsmatrix.

$$T_{uvw \rightarrow xyz} = T_{xyz \rightarrow uvw}^{-1} \quad (8)$$

Hat man einen Punkt  $\vec{p}_{xyz}$  in globale Koordinaten gegeben, so berechnet sich seine Position im lokalen Koordinatensystem entsprechend Formel 9.

$$\vec{p}_{uvw} = T_{xyz \rightarrow uvw} \cdot \vec{p}_{xyz} \quad (9)$$

Analog dazu berechnet man die Rücktransformation eines Punktes  $\vec{p}_{uvw}$  vom lokalen Koordina-

tensystem in das globale Koordinatensystem nach Formel 10.

$$\overrightarrow{p_{xyz}} = T_{uvw \rightarrow xyz} \cdot \overrightarrow{p_{uvw}} \quad (10)$$

Die Berechnung einer speziellen Transformationsmatrix ist in den Klassen `StpPlane`, für planare Ebenen sowie `StpCylindricalFace`, für zylindrische Oberflächen implementiert. Diese befinden sich im Projektordner „`../src/main/java/com.jandoant/stp_entities`“.

Die Methode `getXYZtoUVWTransformationMatrix()` ermittelt die Transformationsmatrix für die Umwandlung eines Punktes in lokale Koordinaten. Die Matrix zur Rücktransformation wird über die Methode `getUVWtoXYZTransformationMatrix()` berechnet. Die Anwendung dieser Matrizen erfolgt in der Klasse `StpCartesianPoint` im selben Projektordner.

Die Methode `baseTransform(Matrix transformationMatrix)` nimmt eine Transformationsmatrix entgegen und gibt einen entsprechend der Formeln 9 bzw. 10 transformierten Punkt aus.

#### 4.2.2 Diskretisierung zylindrischer Flächen

Zylindrische *ADVANCED\_FACEs* werden in einer STEP-Datei durch eine *CYLINDRICAL\_SURFACE* und zwei *CIRCLEs* definiert. Dabei entspricht die *CYLINDRICAL\_SURFACE*, wie in Abschnitt 4.1.2 auf Seite 23 bereits erläutert, der Mantelfläche des Zylinders. Die Begrenzungen des Zylinders ergeben sich aus der Lage der beiden Kreise (STEP-Entität *CIRCLE*), die als *FACE\_OUTER\_BOUNDS* in der STEP-Datei formuliert sind. Der Abstand der beiden Kreise ergibt somit die gesamte Länge der zylindrischen Körperfläche.

Um eine zylindrische *ADVANCED\_FACE* zu diskretisieren, müssen die Positionen der *CYLINDRICAL\_SURFACE* und der *CIRCLEs*, gegeben durch jeweils ein *AXIS2\_PLACEMENT\_3D*, gefunden werden. Im Anschluss werden diese Positionen aus dem globalen in ein lokales Koordinatensystem transformiert. Dies ermöglicht einen generalisierten Diskretisierungsvorgang für Zylinder, unabhängig von ihrer Lage im Raum. Die Transformationsmatrix für eine *CYLINDRICAL\_SURFACE* ergibt sich, wie im vorherigen Abschnitt 4.2.1 beschrieben.

Die Abwicklung der zylindrischen Mantelfläche ergibt ein Rechteck. Dieses entspricht in seiner Breite dem Umfang des Kreisprofils des Zylinders und in der Höhe der Länge des Zylinders. Eine Rechteckfläche ist sehr einfach durch ein gleichmäßiges Punkteraster zu diskretisieren. Dabei ist es notwendig, zuerst die Abwicklung durchzuführen. Über die Transformation der lokalen kartesischen Koordinaten in u-, v- und w-Richtung in lokale zylindrische Koordinaten soll dies umgesetzt werden. Nach erfolgter Transformation in das zylindrische Koordinatensystem kann jeder Punkt auf der Zylinderoberfläche als Punkt auf der abgewickelten Mantelfläche in zylindrischen Koordinaten ausgedrückt werden. Abbildung 10 auf der nächsten Seite soll den

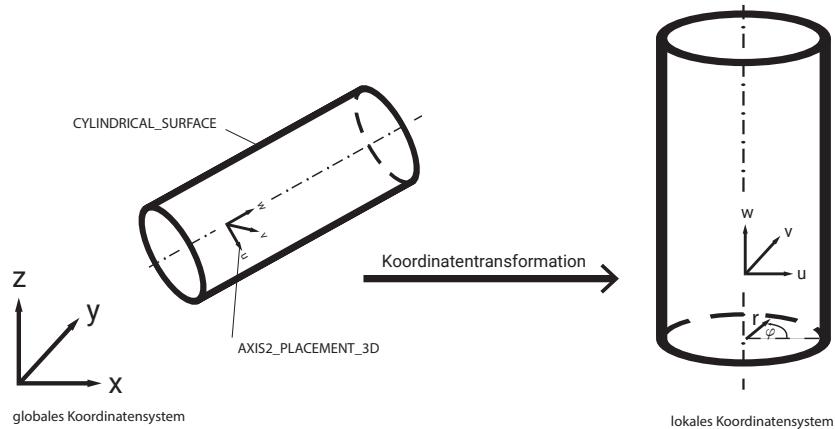


Abbildung 9: Koordinatentransformation CYLINDRICAL\_SURFACE

Sachverhalt darstellen.

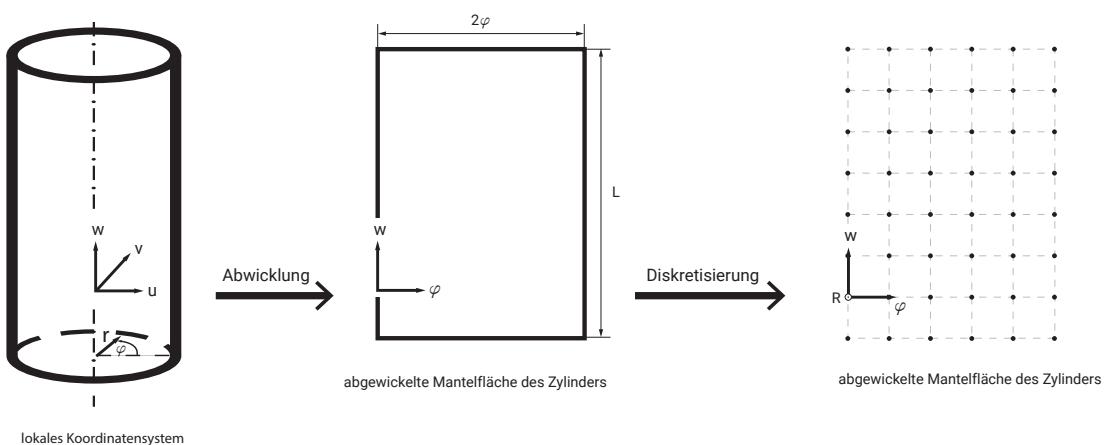


Abbildung 10: Abwicklung und Diskretisierung einer zylindrischen Körperfläche

Die Transformation eines kartesischen Punktes der Zylinderoberfläche kann mit den folgenden Formeln in einen Punkt in zylindrischen Koordinaten umgerechnet werden. Wenn gilt

$$\begin{aligned}\overrightarrow{p_{uvw}} &= \begin{pmatrix} u & v & w \end{pmatrix}^T \\ \overrightarrow{p_{r\varphi w}} &= \begin{pmatrix} \varphi & w & r \end{pmatrix}^T\end{aligned}\tag{11}$$

dann ergeben sich die zylindrischen Koordinaten bei gegebenen kartesischen Koordinaten nach

$$\begin{aligned}r &= \sqrt{u^2 + v^2} \\ \varphi &= \arctan \frac{v}{u}\end{aligned}\tag{12}$$

Der Wert in w-Richtung ändert sich dabei nicht.

Analog dazu lässt sich auch die Rücktransformation in kartesische Koordinaten bei gegebenen zylindrischen Koordinaten durchführen. Auch dabei ändert sich die w-Koordinate nicht.

$$\begin{aligned} u &= r \cdot \cos \varphi \\ v &= r \cdot \sin \varphi \end{aligned} \quad (13)$$

Um die Diskretisierung einer gegebenen zylindrischen *ADVANCED\_FACE* durchzuführen, ist es also notwendig, diese abgewickelte Rechteckfläche mit einem Punkteraster zu versehen. Der Nutzer kann dabei angeben, in wieviele axiale Ringe und wieviele Zylindersegmente die Fläche zerlegt werden soll. Aufgrund dieser Angaben wird das Punkteraster entsprechend erstellt. Das Programm berechnet aus der gewünschten Anzahl der Ringe die Abstände in w-Richtung, aus der Angabe der Anzahl der Zylindersegmente den Abstand in  $\varphi$ -Richtung. Beginnend bei der w-Koordinate des Mittelpunkts des ersten *CIRCLEs* und der Winkelkoordinate  $\varphi = 0$  wird das Raster bis zur w-Koordinate des zweiten *CIRCLEs* und  $\varphi = 2\pi$  in den benötigten Abständen aufgezogen. Jeder erzeugte Punkt des Rasters wird in der Punktewolke der *ADVANCED\_FACE* gespeichert und kann bei Deformation der Fläche verändert werden.

#### 4.2.3 Diskretisierung planarer Flächen

Planare *ADVANCED\_FACES* werden in STEP durch die *PLANE*, auf der sie liegen, und einer bestimmten Anzahl an Umrandungen beschrieben. Die Menge aller Punkte, die zu dieser *ADVANCED\_FACE* gehören, ergibt sich aus der Schnittmenge aller Punkte der *PLANE* und den Punkten innerhalb der positiven Umrandungen (*FACE\_OUTER\_BOUNDS*) ohne die Punkte, die innerhalb der negativen Umrandung liegen. Dies ist durch folgende Abbildung zu visualisieren.

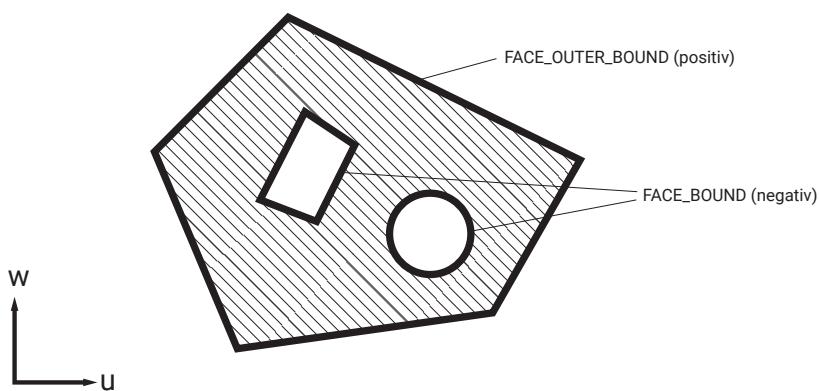


Abbildung 11: Planare *ADVANCED\_FACE* mit negativen Umrandungen

Bei der Diskretisierung planarer Flächen wird unterschieden zwischen kreisförmigen Umrandun-

gen und Umrandungen, die aus einem Polygon bestehen.

Als erstes soll die Diskretisierung von positiven kreisförmigen Umrandungen beschrieben werden. Dazu wird die kreisförmige Fläche in Ringe und Segmente unterteilt. Der Nutzer kann für jede Umrandung die Anzahl dieser angeben und somit die Detaillierung der Diskretisierung steuern. Die Schnittpunkte dieser Elemente sollen als Objekte der Klasse `StpCartesianPoint` in der Punktewolke der Körperfläche gespeichert werden.

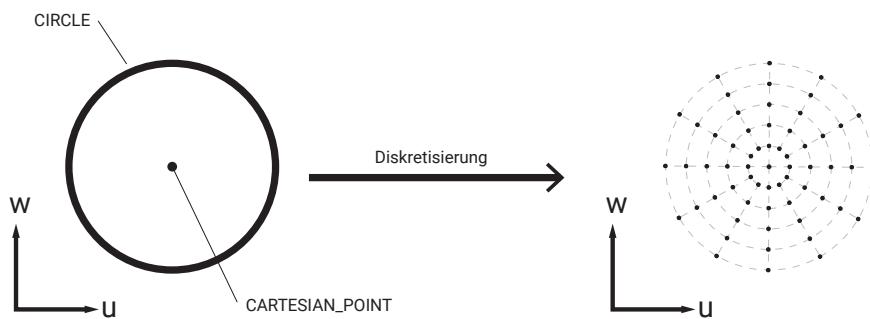


Abbildung 12: Diskretisierungsmuster Kreis

Wie in Abschnitt [4.1.2 auf Seite 23](#) beschrieben, besteht eine einzelne Umrandung einer `ADVANCED_FACE` aus einem Kantenverbund (`EDGE_LOOP`). Bei kreisförmigen Umrandungen wird dieser Kantenverbund genau aus einer Kante des Typs `CIRCLE` gebildet. Diese Entität enthält Informationen zur Position des Kreises (in Form einer `AXIS2_PLACEMENT_3D`) und dessen Radius. Mit dieser Kenntnis kann die Lage des Kreises, bezogen auf das lokale Koordinatensystem, bestimmt werden. Dazu muss auch der Mittelpunkt des Kreises in das lokale Koordinatensystem transformiert werden. Ausgehend vom Radius des zu diskretisierenden Kreises und der Angabe der Anzahl an geforderten Ringen, errechnet das Programm den radialen Abstand der Ringe zueinander. Aus der Anzahl der geforderten Segmente sind in gleicher Weise auch die Öffnungswinkel der Segmente bekannt. Um nun die diskretisierte Punktemenge der Kreisfläche zu erzeugen, wird ein Startpunkt gewählt, welcher auf dem Kreis liegt. Dieser wird um den Mittelpunkt des Kreises und den Öffnungswinkel der Kreissegmente gedreht. Seine neue Position wird in der Punktewolke der betreffenden `ADVANCED_FACE` gespeichert. Danach wird dieser Vorgang wiederholt, bis die Summe der überstrichenen Öffnungswinkel  $360^\circ$  beträgt. Als Ergebnis dieses Vorgangs entsteht nun ein diskretisierter Ring mit dem Radius des Kreises. Nach Bildung des ersten Ringes wird ein neuer Startpunkt gewählt und von diesem aus analog ein weiterer Ring erzeugt. Dieser Vorgang wird so oft wiederholt, bis die geforderte Anzahl an Ringen hergestellt ist.

Die Diskretisierung positiver polygonaler Umrandungen geschieht ein wenig anders. Dabei sollen sowohl die Kanten als auch der Innenraum des Polygons entsprechend Abbildung 13 in einem vom Nutzer wählbaren Abstand der Punkte diskretisiert werden. Dazu sollen in einem ersten Schritt alle Kanten des Polygons im vorgegebenen Abstand unterteilt werden. Danach soll ein Punkteraster mit Punkten dieses Abstandes über das Innere des Polygons gelegt werden. Dazu ist es notwendig, alle Eckpunkte, die das Polygon besitzt, zu finden. Das geschieht über den Kantenverbund der (*EDGE\_LOOP*) Umrandung. Dieser ist, anders als bei kreisförmigen Umrandungen, aus mindestens 3 Kanten aufgebaut. Diese Kanten sind vom Typ *LINE*. Aus der einzelnen Kante lassen sich sowohl Anfangs- als auch Endpunkt der *LINE* auslesen. Somit lassen sich die Eckpunkte des Polygons extrahieren. Nachdem die Menge der Eckpunkte des Polygons ermittelt wurde, werden alle diese Punkte in das lokale Koordinatensystem der Ebene unter Anwendung der Transformationsmatrix umgerechnet. Aus der Kenntnis der Eckpunkte kann nun ein Punkt entlang der Kanten im vorgegebenen Abstand bewegt werden. Nach jedem Einzelschritt wird der Punkt der Punktwolke der *ADVANCED\_FACE* hinzugefügt. Um das Innere des Polygons zu diskretisieren, muss zunächst das umgrenzende Rechteck gefunden werden. Innerhalb der Grenzen dieses Rechtecks wird ein Punkteraster im vorgegebenen Abstand erzeugt. Für jeden Punkt wird individuell entschieden, ob er inner- oder außerhalb des Polygons liegt. Liegt ein Punkt dieses Rasters innerhalb des Polygons, dann wird er zur Punktwolke der Körperfläche hinzugefügt.

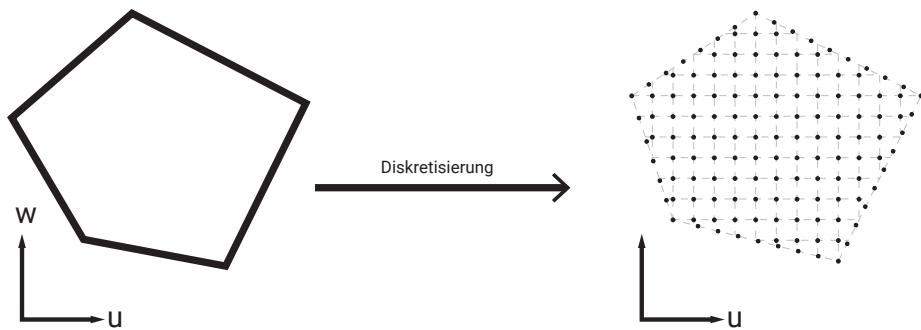


Abbildung 13: Diskretisierungsmuster Polygon

Alle Punkte, die innerhalb einer negativen Umrandung (kreisförmig oder polygonal) liegen, sollen aus der bestehenden Punktwolke der betreffenden *ADVANCED\_FACE* entfernt werden. Dazu wird für jeden bestehenden Punkt geprüft, ob dies der Fall ist. Um allerdings die Kanten negativer Umrandungen in der Punktwolke zu behalten, werden diese entsprechend der Parameter unterteilt und der Punktwolke hinzugefügt.

### 4.3 Deformation der Körperflächen

Die Deformation der einzelnen Punkte einer Körperfläche wird über die Anwendung dreidimensionaler Funktionen realisiert. Die Punkte liegen zu diesem Zeitpunkt des Programmablaufs in lokalen Koordinaten vor. Dabei haben Punkte, die auf einer *PLANE* liegen Koordinaten in u-, v- und w-Richtung. Bei diesen werden die Werte in w-Richtung durch die jeweilige Deformationsfunktion der Form  $w = w_0 + f(u, v)$  verändert. Punkte, welche auf einer *CYLINDRICAL\_SURFACE* liegen haben hingegen lokale Koordinaten in  $\varphi$ -, w- und r-Richtung. Die R-Koordinate wird dabei durch eine Deformationsfunktion der Form  $r = r_0 + f(\varphi, w)$  berechnet. Die Werte  $w_0$  bzw.  $r_0$  stellen dabei den Wert der jeweiligen Koordinate vor Anwendung der Deformation dar. Es können für jede Körperfläche unterschiedlich viele Deformationen beaufschlagt werden. Dies führt zu einer Kumulation der Verschiebungen. Damit kann man unterschiedliche Oberflächenabweichungen überlagern. Die verschiedenen Deformationsfunktionen sind im Quellcode unter dem Projektordner „./src/main/java/com.jandoant/deformation“ zu finden. Jede Klasse in diesem Ordner stellt jeweils eine Deformationsfunktion dar. Diese implementieren alle das Interface `DeformationFunction` im selben Ordner. Das dient dazu, dass alle Deformationsfunktionen der oben beschriebenen Form entsprechen und der Funktionswert stets von zwei Variablen abhängig ist. Zur Anwendung kommt eine Deformationsfunktion durch den Aufruf der Methode `applyDeformationFunction(DeformationFunction df)` der Klasse `StpAdvancedFace` im Ordner „./src/main/java/com.jandoant/stp\_entites“. Durch die Anwendung der Funktion addieren sich die dritte Koordinate des jeweiligen Punktes und der Funktionswert der Deformationsfunktion an der Stelle des Punktes. So wird die w- bzw.  $\varphi$ -Koordinate des durch die Deformation verschobenen Punktes gebildet. Im Programm sind ein paar beispielhafte Funktionen definiert. Aufgabe einer Weiterentwicklung des Programms besteht in der Erweiterung und Optimierung dieses Angebotes an Deformationsfunktionen um Oberflächenabweichungen realistisch abbilden zu können.

Im Folgenden sollen die bereits implementierten Funktionen kurz vorgestellt werden. Dabei werden die Funktionen zur Veränderung planarer Flächen dargestellt. Diese lassen sich aber ebenfalls für zylindrische Flächen anwenden, wenn man u- und v-Koordinate jeweils durch  $\varphi$  und w-Koordinate des Zylinders ersetzt.

**Konstanter Faktor** Auf die Fläche wird ein konstantes Offset aufgegeben. Bei ebenen Flächen führt dies zu einer Absenkung oder Anhebung der Fläche in Normalenrichtung der Ebene. Bei Zylindrischen Flächen führt die Anwendung zu einer Erhöhung oder Verringerung des Radius. Die Funktion folgt der folgenden Vorschrift. Sie ist implementiert in der Klasse `DeformConstant`

und erhält als Parameter den zu verwendenden konstanten Wert.

$$f(u, v) = c_0 \quad (14)$$

**Linear in einer Richtung** Wird auf eine ebene Fläche diese Funktion aufgegeben, dann bildet sich eine schiefe Ebene in der angegebenen Richtung. Bei der Anwendung der Funktion auf zylindrische Flächen ergibt nur die Anwendung in axialer Richtung Sinn. So kann bei zylindrischen Flächen eine konische Form hergestellt werden. Die Funktion folgt der folgenden Vorschrift und erhält als Parameter den Anstieg  $c_1$ , die Verschiebung  $c_0$  der Punkte senkrecht zur Ebene bzw. Achse und die Richtung der Neigung.

$$f(u, v) = a_1 \cdot u + c_0 \quad \text{bzw.} \quad f(u, v) = a_1 \cdot v + c_0 \quad (15)$$

Abbildung XYZ stellt die Funktion grafisch dar. Implementiert ist die Funktion in der Klasse DeformLinear.

**Linear in zwei Richtungen** Ähnlich wie bei der vorher beschriebenen Funktion erzeugt diese Deformation eine schiefe Ebene. Allerdings ist die erzeugte Fläche in zwei verschiedenen Richtungen geneigt. Sie folgt der folgenden Gleichung.

$$f(u, v) = a_1 \cdot u + b_1 \cdot v + c_0 \quad (16)$$

Diese Funktion ist implementiert in der Klasse DeformBiLinear. Sie erhält die Parameter  $a_1$ ,  $b_1$  und  $c_0$ . Diese Funktion wird in Abbildung XYZ visualisiert.

**Quadratisch in einer Richtung** Diese Deformationsfunktion funktioniert analog zur linearen Deformation in einer Richtung. Es wird eine gekrümmte Fläche hergestellt. Bei zylindrischen Flächen ergibt sich eine ballige Form, wenn die Funktion in axialer Richtung angewendet wird. Die Funktion wird nach der Formel 17 gebildet und ist in der Klasse DeformQuadratic implementiert. Übergeben werden die Parameter  $a_2$ ,  $a_1$  und  $c_0$ .

$$f(u, v) = a_2 \cdot u^2 + a_1 \cdot u + c_0 \quad \text{bzw.} \quad f(u, v) = a_2 \cdot v^2 + a_1 \cdot v + c_0 \quad (17)$$

**Quadratisch in zwei Richtungen** Diese Funktion kann verwendet werden, um eine konvexe bzw. konkave Fläche herzustellen. Sie ergibt sich nach Formel 18 auf der nächsten Seite und

kann wird in Abbildung XYZ visualisiert. Die Deformation ist in der Klasse `DeformBiQuadratic` implementiert und erhält als Parameter die Koeffizienten  $a_i$ ,  $b_i$  und  $c_0$ .

$$f(u, v) = a_2 \cdot u^2 + a_1 \cdot u + b_2 \cdot v^2 + b_1 \cdot v + c_0 \quad (18)$$

**Sinusform in einer Richtung** Um eine wellenförmige Oberfläche zu erzeugen kann diese Deformationsfunktion angewendet werden. Sie folgt der Formel 19 und erhält als Parameter die Werte für Amplitude  $a_0$ , Periode  $p$  und Phasenverschiebung  $\phi_0$ .

$$f(u, v) = a_0 \cdot \sin\left(\frac{p}{2\pi} \cdot x - \frac{2\pi \cdot \phi_0}{p}\right) \quad (19)$$

Die Funktion ist in der Klasse `DeformUniDirectionalSine` implementiert und kann folgendermaßen visualisiert werden.

#### 4.4 Ausgabe der verformten Geometrie

Die verformte Geometrie soll als Ergebnis des Programms in einer ASCII-Textdatei ausgegeben werden. Dafür ist es notwendig, dass für alle erzeugten und deformierten Punkte einer Körperfläche diese wieder zurücktransformiert werden. Dies erfolgt entsprechend der Formel 10 auf Seite 32. Für zylindrische `ADVANCED_FACE`s ist es notwendig, die Rücktransformation in lokale kartesische Koordinaten, wie durch Formel 13 auf Seite 34 beschrieben, vor der Basistransformation durchzuführen. Ziel ist die Ausgabe der x-, y- und z-Koordinate jedes Punktes. Dabei entspricht jede Zeile genau einem Punkt. Die Werte sind durch Semikolon getrennt und in der Reihenfolge x, y und z nebeneinander aufgelistet. Die erzeugte Text-Datei kann im Anschluss im Projektordner gefunden werden und mit jedem Textprogramm ausgelesen werden. Die Rücktransformation und anschließende Ausgabe der Ergebnispunkte einer jeden `ADVANCED_FACE` ist in der Funktion `print()` der Klasse `StpAdvancedFace` implementiert.

#### 4.5 Erläuterung eines beispielhaften Programmablaufs

Die Bedienung des Programms ist zum derzeitigen Stand nur innerhalb der Java-Entwicklungsumgebung <sup>2</sup> möglich. Da es sich noch um einen frühen Prototypen einer Simulationssoftware für Fertigungsabweichungen handelt, ist noch keine anwenderfreundliche Programmnutzung

<sup>2</sup>bspw. IntelliJ IDEA (Community Edition) der Firma Jetbrains,  
kostenlos herunterladbar unter <https://www.jetbrains.com/idea/download>

implementiert. Es soll vorrangig um die Demonstration eines möglichen Programmablaufs gehen.

Ein beispielhaftes Programm ist in der `MainClass` im Projektordner „`./src/main/java/com.jandoant`“ hinterlegt. Startet man die `main`-Funktion der Klasse, dann beginnt das Programm mit dem Einlesen der Datei, welche der Instanz der Klasse `StpModelBuilder` übergeben wird. Diese Übergabe, erfolgt in Form einer Zeichenkette, welche der absoluten Adresse der Datei im Ordnersystem des Computers entspricht. Nachdem die STEP-Datei eingelesen ist, wird diese im zweiten Schritt durch den Aufruf der Methode `parseFile()` der Klasse `StpModelBuilder` decodiert und in Java-Objekten abgebildet. Als Ergebnis dieser Operation gibt diese Methode eine Liste mit Instanzen der Klasse `StpAdvancedFace` zurück, welche alle `ADVANCED_FACES` des Bauteils repräsentieren.

Im nächsten Schritt erfolgt die Diskretisierung der Elemente dieser Liste. Dazu wird erst eine Unterscheidung vorgenommen, ob es sich um planare oder zylindrische Körperflächen handelt. Handelt es sich um Flächen, die sich auf einer Ebene befinden, werden zuerst alle positiven und danach alle negativen Umrandungen (*BOUNDS*) diskretisiert und der Punktemenge der jeweiligen `ADVANCED_FACE` hinzugefügt oder davon abgezogen. Der Nutzer kann über die Angabe der Parameter `distanceOfPoints` (Punktstand) für polygonale Umrandungen bzw. `numOfRadialSegments` (Anzahl an Kreissegmenten) und `numOfRings` (Anzahl an Ringen) für kreisförmige Umrandungen die Diskretisierungsparameter, wie in Abschnitt [4.2.3 auf Seite 34](#) beschrieben, festlegen. Handelt es sich bei der betrachteten `ADVANCED_FACE` hingegen um eine zylindrische Körperfläche, so kann der Nutzer die Parameter `numOfRadialSegments` (Anzahl an Zylindersegmenten) und `numOfRings` (Anzahl an Ringen) angeben.

Nach dem Ablauf dieses Programmteils liegen alle Körperflächen mit ihren diskretisierten Punkten in der Liste der Instanzen der Klasse `StpAdvancedFace` vor. Es ist aber zu betonen, dass der Nutzer in diesem beispielhaften Programmablauf die Parameter nur innerhalb des Quelltextes ändern kann. Es erfolgt keine Aufforderung die Parameter einzugeben. Weiterhin werden aufgrund der Vereinfachung des Programms, alle zylindrischen Flächen des Modells mit den gleichen Parametern diskretisiert. Analog dazu werden auch alle planaren kreisförmigen und alle planaren polygonalen Umrandungen mit den gleichen Parametern, die für ihren Typ festgelegt wurden diskretisiert. Das ist aber nur in diesem Beispielprogramm auf diese Weise implementiert, es lassen sich mit den verfügbaren Programmbausteinen auch Programmabläufe gestalten, bei denen individuelle Parameter für jede Körperfläche festgelegt werden können und somit eine flexible Anpassung an den benötigten Detaillierungsgrad möglich wird.

Im folgenden Programmschritt werden die Elemente der Punktewolke der `ADVANCED_FACES` entsprechend der angewandten Deformationsfunktionen verformt. In diesem Beispielprogramm werden die ersten drei `ADVANCED_FACES` des Bauteils mit verschiedenen Deformationsfunkti-

nen beaufschlagt. Über die Funktion `applyDeformationFunction(DeformationFunction df)` der Klasse `StpAdvancedFace` wird eine der möglichen Deformationsfunktionen angewandt. Werden mehrere Funktionen für eine *ADVANCED\_FACE* benutzt, summiert sich ihre Wirkung.

Im letzten Schritt des Programms werden die Koordinaten der Elemente aller Punktewolken des Modells in einer Text-Datei ausgegeben. Dabei werden alle Punkte wieder in das Weltkoordinatensystem transformiert und anschließend in die Text-Datei geschrieben. Die erzeugte Datei ist im Ordner „./src/main/java/com.jandoant/results“ zu finden.

## 5 Fazit und Ausblick

Mit den Überlegungen in dieser Arbeit und dem entwickelten Programm wurde eine Grundlage für ein Software-System geschaffen, welches perspektivisch erwartbare Fertigungsabweichungen bei der Herstellung von Bauteilen simulieren kann.

Es wurde untersucht, welche typischen Abweichungen bei der Fertigung von Werkstücken mit bestimmten Verfahren auftreten. Dabei ist festzustellen, dass qualitativ durchaus charakteristische Fehlerbilder für verschiedene Fertigungsprozesse zu erkennen sind. Allerdings lassen die Untersuchungen nur begrenzt qualitative Aussagen bezüglich der Fehler zu. Das erschwert eine Konzeption der Software mit Hinblick auf die Zielstellung, vom angewandten Fertigungsverfahren direkt auf die zu erwartenden Fehler zu schließen. Es ist aber realistisch, die Art der möglichen Fehler einer Oberfläche nach dem Fertigungsverfahren zu klassifizieren und dem Nutzer bei Auswahl der jeweiligen Körperfläche anzubieten. Die konkreten Werte müssen vom Nutzer auf Grundlage der spezifischen Herstellungseinflüsse selbst ermittelt und getroffen werden. Die zu erwartende Wirkliche Oberfläche des Bauteils ist quantitativ nicht generalisiert vorhersagbar. Mit Kenntnis der Arbeitsweise der verwendeten Werkzeugmaschine, der Produktionsumgebung und der Prozessbedingungen sowie dem Material und dem Verschleißzustand des Werkzeuges lassen sich durch Vergleich mit anderen, unter ähnlichen Bedingungen bereits hergestellten Bauteilen, Vorhersagen treffen. Diese für jeden Fertigungsprozess zu erlangenden Erkenntnisse müssen in die Software einfließen, sodass zu erwartende Fehler kundenspezifisch simuliert werden können.

Das entwickelte Programm ist zum derzeitigen Stand in der Lage, eine STEP-Datei des zu untersuchenden Bauteils einzulesen und in einzelne Geometrieelemente zu zerlegen. Für jedes einzelne Geometrieelement lassen sich kontinuierliche Deformationsfunktionen anwenden, welche die ideale Formgestalt des Bauteils ändern. Als Ergebnis werden die räumlichen Koordinaten einer Punktewolke der veränderten Geometrie in einer Text-Datei ausgegeben.

Es ist zu betonen, dass die vorliegende Software als Grundlage für die Umsetzung einer Simulationssoftware für Fertigungsabweichungen eines vorgegebenen Bauteils zu betrachten ist. Es handelt sich keineswegs um ein bereits praktisch verwendbares Programm. Zum derzeitigen Stand sind nur Bauteile, die entweder planare oder zylindrische Flächen als Geometrieelemente aufweisen, verwendbar. Um andere Geometrien, wie beispielsweise Freiformflächen oder gezogene Profile, verwenden zu können, müsste auf Grundlage der bereits umgesetzten STEP-Entitäten eine Ergänzung der Implementierung anderer Körperflächentypen geschehen. Das entwickelte Programm bietet die Möglichkeit, eine ideale Geometrie mit gewünschten Verformungen zu beaufschlagen. Allerdings muss der Katalog um weitere möglichen Deformationsfunktionen erweitert werden. Die bisher verfügbaren Funktionen sollen die angedachte

Funktionsweise des Programms verdeutlichen, sind aber noch nicht an die eigentlichen Fertigungsfehler angepasst und optimiert. Um eine tatsächliche Simulationsfunktion der Software umzusetzen, müssen solche Funktionen gefunden werden, die eine Anpassung der Parameter durch den Nutzer erlauben und realistische Fehler abbilden können.

Bei der Weiterentwicklung des Programms gibt es sehr viel Potential. Neben der angesprochenen Erweiterung des Spektrums an Geometrieelementen und möglichen Deformationen ist auch die Benutzerfreundlichkeit ein wichtiger Aspekt. So ist eine Visualisierung der Geometrie unbedingt notwendig, um die Auswahl der zu verändernden Geometrieelemente für den Nutzer intuitiv zu gestalten. Derzeit ist eine Auswahl einer Körperfläche nur über deren ID möglich, dies stellt jedoch eine sehr umständliche und wenig anwenderfreundliche Art der Bedienung dar. Des weiteren muss ein konkreter Programmablauf, ähnlich dem dargestellten beispielhaften Programmablauf, entwickelt werden. Die Bausteine und Herangehensweise, um dies umzusetzen, sind grundsätzlich durch die Programmierung in dieser Arbeit gegeben.

Durch die Verwendung kontinuierlicher Funktionen können hauptsächlich Gestaltabweichungen nach DIN 4760 [1] umgesetzt werden. Es ist aber perspektivisch auch denkbar, Oberflächenunvollkommenheiten nach DIN EN ISO 8785 [2] zu simulieren. Mit dem aus der Computergrafik bekannten Verfahren des Displacement Mappings, wie beispielsweise in [11] beschrieben, lassen sich über diskrete, zum Beispiel in Form von Graustufenbildern, gegebene Verformungsfelder auf Körperflächen projizieren. Dabei ist in jedem Pixel des Graustufenbildes ein Helligkeitswert gespeichert, der ein Maß für die Verschiebung des Punktes an der Stelle des Pixels darstellt. So lassen sich auch singuläre Strukturen, wie Kratzer oder Ausbrüche, auf der Körperoberfläche darstellen.

Der derzeitig vorhandene Prototyp des Programms muss und kann also noch in vielen Aspekten weiterentwickelt werden, um eine aussagekräftige Simulation typischer Fertigungsabweichungen zu erlauben.

## A Anhang

### A.1 STEP-Beschreibung eines Quaders

```
ISO-10303-21;

HEADER;

/* Generated by software containing ST-Developer
 * from STEP Tools, Inc. (www.step-tools.com)
 */

FILE_DESCRIPTION(
/* description */ (''),
/* implementation_level */ '2;1');

FILE_NAME(
/* name */
'C:\\\\Users\\\\Jan\\\\Documents\\\\Studium\\\\BA\\\\04 Kladde und
Skizzen\\\\Quader.
stp',
/* time_stamp */ '2018-03-15T14:40:41+01:00',
/* author */ ('Jan'),
/* organization */ (''),
/* preprocessor_version */ 'ST-DEVELOPER v16.13',
/* originating_system */ 'Autodesk Inventor 2018',
/* authorisation */ '');

FILE_SCHEMA (('AUTOMOTIVE_DESIGN { 1 0 10303 214 3 1 1 }'))
);

ENDSEC;

DATA;
#10=
    MECHANICAL DESIGN GEOMETRIC PRESENTATION REPRESENTATION
('',( #13 ), #189 );
#11= SHAPE REPRESENTATION RELATIONSHIP ('SRR', 'None',
, #196 , #12 );
#12= ADVANCED_BREP_SHAPE REPRESENTATION ('', (#14) , #188 );
#13= STYLED ITEM ('', (#205) , #14 );
```

```
#14=MANIFOLD_SOLID_BREP('Volumenk\xF6rper1',#107);
#15=FACE_OUTER_BOUND(' ',#21,.T.);
#16=FACE_OUTER_BOUND(' ',#22,.T.);
#17=FACE_OUTER_BOUND(' ',#23,.T.);
#18=FACE_OUTER_BOUND(' ',#24,.T.);
#19=FACE_OUTER_BOUND(' ',#25,.T.);
#20=FACE_OUTER_BOUND(' ',#26,.T.);
#21=EDGE_LOOP(' ',(#71,#72,#73,#74));
#22=EDGE_LOOP(' ',(#75,#76,#77,#78));
#23=EDGE_LOOP(' ',(#79,#80,#81,#82));
#24=EDGE_LOOP(' ',(#83,#84,#85,#86));
#25=EDGE_LOOP(' ',(#87,#88,#89,#90));
#26=EDGE_LOOP(' ',(#91,#92,#93,#94));
#27=LINE(' ',#163,#39);
#28=LINE(' ',#165,#40);
#29=LINE(' ',#167,#41);
#30=LINE(' ',#168,#42);
#31=LINE(' ',#171,#43);
#32=LINE(' ',#173,#44);
#33=LINE(' ',#174,#45);
#34=LINE(' ',#177,#46);
#35=LINE(' ',#179,#47);
#36=LINE(' ',#180,#48);
#37=LINE(' ',#182,#49);
#38=LINE(' ',#183,#50);
#39=VECTOR(' ',#137,10.);
#40=VECTOR(' ',#138,10.);
#41=VECTOR(' ',#139,10.);
#42=VECTOR(' ',#140,10.);
#43=VECTOR(' ',#143,10.);
#44=VECTOR(' ',#144,10.);
#45=VECTOR(' ',#145,10.);
#46=VECTOR(' ',#148,10.);
#47=VECTOR(' ',#149,10.);
#48=VECTOR(' ',#150,10.);
#49=VECTOR(' ',#153,10.);
#50=VECTOR(' ',#154,10.);
```

```
#51=VERTEX_POINT( ,#161) ;
#52=VERTEX_POINT( ,#162) ;
#53=VERTEX_POINT( ,#164) ;
#54=VERTEX_POINT( ,#166) ;
#55=VERTEX_POINT( ,#170) ;
#56=VERTEX_POINT( ,#172) ;
#57=VERTEX_POINT( ,#176) ;
#58=VERTEX_POINT( ,#178) ;

#59=EDGE_CURVE( ,#51,#52,#27,.T.) ;
#60=EDGE_CURVE( ,#52,#53,#28,.T.) ;
#61=EDGE_CURVE( ,#54,#53,#29,.T.) ;
#62=EDGE_CURVE( ,#51,#54,#30,.T.) ;
#63=EDGE_CURVE( ,#55,#51,#31,.T.) ;
#64=EDGE_CURVE( ,#56,#54,#32,.T.) ;
#65=EDGE_CURVE( ,#55,#56,#33,.T.) ;
#66=EDGE_CURVE( ,#57,#55,#34,.T.) ;
#67=EDGE_CURVE( ,#58,#56,#35,.T.) ;
#68=EDGE_CURVE( ,#57,#58,#36,.T.) ;
#69=EDGE_CURVE( ,#52,#57,#37,.T.) ;
#70=EDGE_CURVE( ,#53,#58,#38,.T.) ;
#71=ORIENTED_EDGE( ,*,*,#59,.T.) ;
#72=ORIENTED_EDGE( ,*,*,#60,.T.) ;
#73=ORIENTED_EDGE( ,*,*,#61,.F.) ;
#74=ORIENTED_EDGE( ,*,*,#62,.F.) ;
#75=ORIENTED_EDGE( ,*,*,#63,.T.) ;
#76=ORIENTED_EDGE( ,*,*,#62,.T.) ;
#77=ORIENTED_EDGE( ,*,*,#64,.F.) ;
#78=ORIENTED_EDGE( ,*,*,#65,.F.) ;
#79=ORIENTED_EDGE( ,*,*,#66,.T.) ;
#80=ORIENTED_EDGE( ,*,*,#65,.T.) ;
#81=ORIENTED_EDGE( ,*,*,#67,.F.) ;
#82=ORIENTED_EDGE( ,*,*,#68,.F.) ;
#83=ORIENTED_EDGE( ,*,*,#69,.T.) ;
#84=ORIENTED_EDGE( ,*,*,#68,.T.) ;
#85=ORIENTED_EDGE( ,*,*,#70,.F.) ;
#86=ORIENTED_EDGE( ,*,*,#60,.F.) ;
#87=ORIENTED_EDGE( ,*,*,#70,.T.) ;
```

```

#88=ORIENTED_EDGE(' ',*,*,#67,.T.);
#89=ORIENTED_EDGE(' ',*,*,#64,.T.);
#90=ORIENTED_EDGE(' ',*,*,#61,.T.);
#91=ORIENTED_EDGE(' ',*,*,#69,.F.);
#92=ORIENTED_EDGE(' ',*,*,#59,.F.);
#93=ORIENTED_EDGE(' ',*,*,#63,.F.);
#94=ORIENTED_EDGE(' ',*,*,#66,.F.);
#95=PLANE(' ',#127);
#96=PLANE(' ',#128);
#97=PLANE(' ',#129);
#98=PLANE(' ',#130);
#99=PLANE(' ',#131);
#100=PLANE(' ',#132);
#101=ADVANCED_FACE(' ',(#15),#95,.T.);
#102=ADVANCED_FACE(' ',(#16),#96,.T.);
#103=ADVANCED_FACE(' ',(#17),#97,.T.);
#104=ADVANCED_FACE(' ',(#18),#98,.T.);
#105=ADVANCED_FACE(' ',(#19),#99,.T.);
#106=ADVANCED_FACE(' ',(#20),#100,.F.);
#107=CLOSED_SHELL(' ',(#101,#102,#103,#104,#105,#106));
#108=DERIVED_UNIT_ELEMENT(#110,0.);
#109=DERIVED_UNIT_ELEMENT(#191,0.);
#110=(
MASS_UNIT()
NAMED_UNIT(*)
SI_UNIT($,.GRAM.)
);
#111=DERIVED_UNIT((#108,#109));
#112=MEASURE REPRESENTATION_ITEM('density measure',
POSITIVE_RATIO_MEASURE(1.),#111);
#113=PROPERTY DEFINITION REPRESENTATION(#118,#115);
#114=PROPERTY DEFINITION REPRESENTATION(#119,#116);
#115=REPRESENTATION('material name',(#117),#188);
#116=REPRESENTATION('density',(#112),#188);
#117=DESCRIPTIVE REPRESENTATION ITEM('Generisch',
'Generisch');
#118=PROPERTY DEFINITION('material property','material

```

```
        name' ,#198) ;  
#119=PROPERTY_DEFINITION('material_property','density of  
    part',#198);  
#120=DATE_TIME_ROLE('creation_date');  
#121=APPLIED_DATE_AND_TIME_ASSIGNMENT(#122,#120,(#198));  
#122=DATE_AND_TIME(#123,#124);  
#123=CALENDAR_DATE(2018,15,3);  
#124=LOCAL_TIME(13,38,24.,#125);  
#125=COORDINATED_UNIVERSAL_TIME_OFFSET(0,0,.BEHIND.);  
#126=AXIS2_PLACEMENT_3D('placement',#159,#133,#134);  
#127=AXIS2_PLACEMENT_3D(' ',#160,#135,#136);  
#128=AXIS2_PLACEMENT_3D(' ',#169,#141,#142);  
#129=AXIS2_PLACEMENT_3D(' ',#175,#146,#147);  
#130=AXIS2_PLACEMENT_3D(' ',#181,#151,#152);  
#131=AXIS2_PLACEMENT_3D(' ',#184,#155,#156);  
#132=AXIS2_PLACEMENT_3D(' ',#185,#157,#158);  
#133=DIRECTION('axis',(0.,0.,1.));  
#134=DIRECTION('refdir',(1.,0.,0.));  
#135=DIRECTION('center_axis',(-1.,0.,0.));  
#136=DIRECTION('ref_axis',(0.,-1.,0.));  
#137=DIRECTION('',(0.,-1.,0.));  
#138=DIRECTION('',(0.,0.,1.));  
#139=DIRECTION('',(0.,-1.,0.));  
#140=DIRECTION('',(0.,0.,1.));  
#141=DIRECTION('center_axis',(0.,1.,0.));  
#142=DIRECTION('ref_axis',(-1.,0.,0.));  
#143=DIRECTION('',(-1.,0.,0.));  
#144=DIRECTION('',(-1.,0.,0.));  
#145=DIRECTION('',(0.,0.,1.));  
#146=DIRECTION('center_axis',(1.,0.,0.));  
#147=DIRECTION('ref_axis',(0.,1.,0.));  
#148=DIRECTION('',(0.,1.,0.));  
#149=DIRECTION('',(0.,1.,0.));  
#150=DIRECTION('',(0.,0.,1.));  
#151=DIRECTION('center_axis',(0.,-1.,0.));  
#152=DIRECTION('ref_axis',(1.,0.,0.));  
#153=DIRECTION('',(1.,0.,0.));
```

```

#154=DIRECTION('',(1.,0.,0.));
#155=DIRECTION('center_axis',(0.,0.,1.));
#156=DIRECTION('ref_axis',(1.,0.,0.));
#157=DIRECTION('center_axis',(0.,0.,1.));
#158=DIRECTION('ref_axis',(1.,0.,0.));
#159=CARTESIAN_POINT('',(0.,0.,0.));
#160=CARTESIAN_POINT('Origin',(0.,15.,0.));
#161=CARTESIAN_POINT('',(0.,15.,0.));
#162=CARTESIAN_POINT('',(0.,0.,0.));
#163=CARTESIAN_POINT('',(0.,15.,0.));
#164=CARTESIAN_POINT('',(0.,0.,30.));
#165=CARTESIAN_POINT('',(0.,0.,0.));
#166=CARTESIAN_POINT('',(0.,15.,30.));
#167=CARTESIAN_POINT('',(0.,15.,30.));
#168=CARTESIAN_POINT('',(0.,15.,0.));
#169=CARTESIAN_POINT('Origin',(10.,15.,0.));
#170=CARTESIAN_POINT('',(10.,15.,0.));
#171=CARTESIAN_POINT('',(10.,15.,0.));
#172=CARTESIAN_POINT('',(10.,15.,30.));
#173=CARTESIAN_POINT('',(10.,15.,30.));
#174=CARTESIAN_POINT('',(10.,15.,0.));
#175=CARTESIAN_POINT('Origin',(10.,0.,0.));
#176=CARTESIAN_POINT('',(10.,0.,0.));
#177=CARTESIAN_POINT('',(10.,0.,0.));
#178=CARTESIAN_POINT('',(10.,0.,30.));
#179=CARTESIAN_POINT('',(10.,0.,30.));
#180=CARTESIAN_POINT('',(10.,0.,0.));
#181=CARTESIAN_POINT('Origin',(0.,0.,0.));
#182=CARTESIAN_POINT('',(0.,0.,0.));
#183=CARTESIAN_POINT('',(0.,0.,30.));
#184=CARTESIAN_POINT('Origin',(5.,7.5,30.));
#185=CARTESIAN_POINT('Origin',(5.,7.5,0.));
#186=UNCERTAINTY_MEASURE_WITH_UNIT(LENGTH_MEASURE(0.01)
    ,#190,
'DISTANCE_ACCURACY_VALUE',
'Maximum model space distance between geometric entities
at asserted c

```

```
    onnectivities') ;  
#187=UNCERTAINTY_MEASURE_WITH_UNIT(LENGTH_MEASURE(0.01)  
    ,#190,  
'DISTANCE_ACCURACY_VALUE',  
'Maximum model space distance between geometric entities  
at asserted c  
onnectivities');  
#188=(  
GEOMETRIC REPRESENTATION_CONTEXT(3)  
GLOBAL_UNCERTAINTY_ASSIGNED_CONTEXT((#186))  
GLOBAL_UNIT_ASSIGNED_CONTEXT((#190,#193,#192))  
REPRESENTATION_CONTEXT('','3D')  
);  
#189=(  
GEOMETRIC REPRESENTATION_CONTEXT(3)  
GLOBAL_UNCERTAINTY_ASSIGNED_CONTEXT((#187))  
GLOBAL_UNIT_ASSIGNED_CONTEXT((#190,#193,#192))  
REPRESENTATION_CONTEXT('','3D')  
);  
#190=(  
LENGTH_UNIT()  
NAMED_UNIT(*)  
SI_UNIT(.MILLI.,.METRE.)  
);  
#191=(  
LENGTH_UNIT()  
NAMED_UNIT(*)  
SI_UNIT(.CENTI.,.METRE.)  
);  
#192=(  
NAMED_UNIT(*)  
SI_UNIT($,.STERADIAN.)  
SOLID_ANGLE_UNIT()  
);  
#193=(  
NAMED_UNIT(*)  
PLANE_ANGLE_UNIT()
```

```
SI_UNIT($, .RADIANT.)  
);  
#194=SHAPE_DEFINITION REPRESENTATION(#195,#196);  
#195=PRODUCT_DEFINITION_SHAPE(' ', $, #198);  
#196=SHAPE_DEFINITION REPRESENTATION(' ', (#126), #188);  
#197=PRODUCT_DEFINITION_CONTEXT('part definition', #202, '  
design');  
#198=PRODUCT_DEFINITION('Quader', 'Quader', #199, #197);  
#199=PRODUCT_DEFINITION FORMATION(' ', $, #204);  
#200=PRODUCT RELATED PRODUCT CATEGORY('Quader', 'Quader',  
, (#204));  
#201=APPLICATION_PROTOCOL DEFINITION('international  
standard',  
'automotive_design', 2009, #202);  
#202=APPLICATION_CONTEXT(  
'Core Data for Automotive Mechanical Design Process');  
#203=PRODUCT_CONTEXT('part definition', #202, 'mechanical');  
#204=PRODUCT('Quader', 'Quader', $, (#203));  
#205=PRESENTATION_STYLE_ASSIGNMENT((#206));  
#206=SURFACE_STYLE_USAGE(.BOTH., #207);  
#207=SURFACE_SIDE_STYLE(' ', (#208));  
#208=SURFACE_STYLE_FILL_AREA(#209);  
#209=FILL_AREA_STYLE(' ', (#210));  
#210=FILL_AREA_STYLE_COLOUR(' ', #211);  
#211=COLOUR_RGB(' ', 0.749019607843137, 0.749019607843137, 0  
.749019607843137);  
ENDSEC;  
END -ISO -10303 -21;
```

Codebeispiel 4: Beschreibung eines Quaders in STEP

## Abbildungsverzeichnis

1	Ausschnitt aus der Istoberfläche zur Beurteilung der Gestaltabweichung . . . . .	6
2	DIN EN ISO 8785 - Oberflächenunvollkommenheiten . . . . .	8
3	Visualisierung <i>CIRCLE</i> . . . . .	24
4	Visualisierung <i>PLANE</i> . . . . .	24
5	Visualisierung <i>CYLINDRICAL_SURFACE</i> . . . . .	25
6	Visualisierung planare <i>ADVANCED_FACE</i> . . . . .	26
7	Beschreibung eines Punktes in verschiedenen Koordinatensystemen . . . . .	29
8	UV-Mapping einer Körperfläche . . . . .	29
9	Koordinatentransformation <i>CYLINDRICAL_SURFACE</i> . . . . .	33
10	Abwicklung und Diskretisierung einer zylindrischen Körperfläche . . . . .	33
11	Planare <i>ADVANCED_FACE</i> mit negativen Umrandungen . . . . .	34
12	Diskretisierungsmuster Kreis . . . . .	35
13	Diskretisierungsmuster Polygon . . . . .	36

## Literatur

- [1] DIN Deutsches Institut für Normung e.V. *DIN 4760: Gestaltabweichungen*. Berlin, Juni 1982.
- [2] DIN Deutsches Institut für Normung e.V. *DIN EN ISO 8785: Oberflächenunvollkommenheiten*. Berlin, Oktober 1999.
- [3] Berend Denkena und Hans Kurt Tönshoff. *Spanen: Grundlagen*. 3., bearb. u. erw. Aufl. VDI-Buch. Heidelberg u.a.: Springer, 2011. ISBN: 978-3-642-19771-0.
- [4] Reimund Neugebauer, Hrsg. *Werkzeugmaschinen: Aufbau, Funktion und Anwendung von spanenden und abtragenden Werkzeugmaschinen*. Berlin: Springer Vieweg, 2012. ISBN: 978-3-642-30077-6.
- [5] Jochen Dietrich und Heinz Tschätsch. *Praxis der Zerspanetechnik: Verfahren, Werkzeuge, Berechnung*. 11. Auflage. Wiesbaden: Springer Vieweg, 2014. ISBN: 9783658049225.
- [6] Eberhard Paucksch, Sven Holsten, Marco Linß und Franz Tikal. *Zerspanetechnik: Prozesse, Werkzeuge, Technologien: mit 45 Tabellen*. 12., vollst. überarb. und erw. Aufl. Studium. Wiesbaden: Vieweg + Teubner, 2008. ISBN: 978-3-8348-0279-8. doi: [10.1007/978-3-8348-9494-6](https://doi.org/10.1007/978-3-8348-9494-6).
- [7] Herbert Schönherr. *Spanende Fertigung*. Oldenbourg-Lehrbücher für Ingenieure. München: Oldenbourg, 2002. ISBN: 3486250450.
- [8] Horst Winkler. *Zerspanungstechnologie: Leitfaden für die Praxis*. Düsseldorf: VDI-Verl., 1990. ISBN: 3184009882.
- [9] Bernd-Rüdiger Meyer und Dirk Falke. *Maßhaltige Kunststoff-Formteile: Toleranzen und Formteilengineering*. München: Hanser, 2013. ISBN: 978-3-446-43687-9.
- [10] Inc. Step Tools. *STEP Merged AP Library*. 2017. URL: [https://www.step-tools.com/stds/stp\\_aim/html/](https://www.step-tools.com/stds/stp_aim/html/) (besucht am 05.06.2018).
- [11] László Szirmay-Kalos und Tamás Umenhoffer. „Displacement Mapping on the GPU - State of the Art“. In: *Comput. Graph. Forum* 27.6 (2008), S. 1567–1592. doi: [10.1111/j.1467-8659.2007.01108.x](https://doi.org/10.1111/j.1467-8659.2007.01108.x).

## **Selbstständigkeitserklärung**

Ich erkläre, dass ich die vorliegende Arbeit selbstständig und unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche gekennzeichnet.

---

Chemnitz, 06.06.2018

Unterschrift des Verfassers