FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Interactive Variability Analysis for Initial Sample Testing of Industrial CT Data

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Visual Computing

eingereicht von

Johanna Schmidt
Matrikelnummer 0025558

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
Mitwirkung: Dipl.-Ing Laura Fritz

Wien, 15.11.2011

(Unterschrift Johanna Schmidt)

(Unterschrift Betreuung)

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Interactive Variability Analysis for Initial Sample Testing of Industrial CT Data

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Visual Computing

by

Johanna Schmidt
Registration Number 0025558

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
Assistance: Dipl.-Ing Laura Fritz

Vienna, 15.11.2011

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Johanna Schmidt
Krongasse 20/11, 1050 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Johanna Schmidt)

Acknowledgements

In the following lines I would like to gratefully acknowledge all those who gave me the possibility to complete this thesis and for their long-lasting support and encouragement.

Firstly, I like to take this opportunity to express my gratitude to the people who provided scientific support to make this work possible. I would like to thank my supervisor Laura Fritz for piquing my interest in the topic of non-destructive testing for metal castings parts, as well as for encouraging me in implementing my ideas and for her continuous support. Furthermore, my sincere thanks go to Meister Eduard Gröller for his helpful advices and proof-reading this thesis.

This work has been carried out at the VRVis Research Center for Virtual Reality and Visualization (http://www.vrvis.at), located in Vienna. I would like to thank the VRVis for giving me the opportunity to use their office facilities when working on this thesis and for providing me a workplace, as well as for the pleasant and inspiring working environment.

I also would like to thank my fellow students and colleagues Fritz-Michael Gschwantner and Andreas Monitzer for their helpful advices. Fritz, thanks for the nice time we shared at VRVis.

My special thanks go to my family for their mental support during my whole studies. I would like to especially thank my parents, Magdalena and Franz Schmidt, for their continuous encouragement and financial support during my time at the university. Thanks for enabling me to head towards an academic career. Additionally, I would like to thank my brother and sisters for their interest in the topic and for never stopping to ask questions. This still is a great help to see things from a different point of view and obtain a broader perspective on the topic.

I would like to thank Christian for his support and my friends for keeping me grounded during my studies.

Abstract

Casting is a manufacturing process by which melt is poured into a mould with a cavity of the desired object form and then allowed to solidify. Due to the pouring process as well as during solidification *casting defects* like shrinkage, pores or cracks may appear that can influence the cast objects' mechanical properties and usability. Designing a new type of casting always requires the production of a new pattern first. Before release of serial production a series of initial samples is created. During this evaluation step possible problems are identified, that may occur for example due to the mould layout. Often non-destructive testing techniques are used to inspect single samples. Non-destructive testing (NDT) defines a group of analysis techniques used to evaluate the properties of materials or components without causing damage. A common NDT technique is industrial Computed Tomography (CT), which produces 3D cross-sectional images of an object from flat X-ray images.

In the course of this master's thesis effort, a new approach for first article inspection is introduced. The goal of this approach is to explore a series of industrial CT volumes, with the objective to evaluate the casting design. Using the pipeline investigations of the reliability of the production can be performed. The variability analysis is based on a segmentation approach for defect detection which enables to differentiate discontinuities from the rest of the volume.

In case regions of high defect concentration should be identified, a clustering approach is applied to all samples within the series. The input data set used for clustering consists of the defects' centres of gravity. This way the clustering algorithm can be applied to a set of data points in 3D space. Two different clustering approaches have been implemented (*mean-shift* and *k-means++* clustering). In the further course the cluster information is used to analyse the variability of the sample series, since they represent regions of high defect concentration. The analysis of large defects, in contrast, directly operates on the data produced by the defect detection stage. The data of all samples is compared to find possible overlaps. For the variability analysis it is important to know whether regions can be found that contain defects in more than one trial sample. The identification of such regions shows that the casting process tends to produce errors in certain parts of the object, which may necessitate a change in the manufacturing process parameters. The visualisation of the pipeline is done by using a 2D transfer function approach in a 3D rendering environment, where the results can be explored individually by the user.

Kurzfassung

Beim Fertigungsverfahren des Gießens entsteht aus Schmelze, die in einen Formhohlraum gegossen wird und dann erstarrt, ein neuer fester Körper einer bestimmten Form. Durch den Vorgang des Eingießens bzw. durch das Erstarren können sich im neuen Werkstück Defekte wie Lunker, Poren oder Risse bilden, welche die mechanischen Eigenschaften und die Verwendbarkeit des Werkstückes beeinflussen können. Da der Hohlraum einer Gussform die Form des fertigen Werkstückes bestimmt, ist es notwendig, beim Entwurf einer neuen Art von Gusskörper zuerst einen Prototyp der neuen Gußform zu produzieren. Bevor diese die Serienreife erhält müssen Testserien angefertigt werden. Diese werden benötigt, um mögliche Produktionsprobleme erkennen zu können, welche sich zum Beispiel durch die Auslegung der Gussform ergeben können. Oft werden Methoden der zerstörungsfreien Werkstoffprüfung (engl. *non-destructive testing*, kurz NDT) eingesetzt, welche es erlauben, die Qualität eines Werkstückes zu untersuchen ohne es zu beschädigen. Eine beliebte Technik ist hier die industrielle Computertomographie (CT), welche aus Röntgenbildern eines Objektes Schnittbilder in 3D erzeugen kann.

Im Rahmen dieser Diplomarbeit wird eine neue Methode zur Evaluierung von Testserien vorgestellt, welche zur Überprüfung der Serienreife einer neuen Modelleinrichtung erstellt wurden. Ziel ist es, innerhalb von industriellen CT Volumsdatensätzen Regionen mit einer hohen Konzentration an Defekten bzw. besonders große Defekte zu erkennen und über eine Serie von Objekten zu verfolgen. Die Analyse basiert auf einem Segmentierungsalgorithmus welcher es erlaubt, Defektregionen vom Rest des Volumendatensatzes zu unterscheiden.

Soll die Analyse Regionen mit einer hohen Konzentration an Defekten erkennen, so wird zuerst eine Clusteranalyse durchgeführt. Zum Vergleich wurden zwei verschiedene Clusteralgorithmen (*mean-shift* und *k-means++*) implementiert. Dabei dienen die Schwerpunkte der berechneten Defekte als Eingabedatenmenge. Im weiteren Verlauf konzentriert sich die Variabilitätsanalyse auf die berechneten Cluster, da diese Regionen mit einer hohen Konzentration an Defekten repräsentieren. Im Gegensatz dazu arbeitet die Analyse von besonders großen Defekten direkt mit den durch die Defektanalyse gewonnenen Daten. Ziel ist es, die Daten aller Testobjekte zu vergleichen, um Überlappungen feststellen zu können. Cluster bzw. große Defekte, die in einem Großteil der Testobjekte vorkommen, sind für die Analyse von großer Bedeutung. Sie repräsentieren Regionen, die gefährdet sind, während des Herstellungsvorgangs Defekte zu entwickeln, was eine Änderung der Produktionsparameter notwendig machen kann. Die Visualisierung der Pipeline wurde mittels 2D Transferfunktion in einem 3D Volumenrenderer realisiert, um eine individuelle Bearbeitung der Ergebnisse durch den Benutzer zu ermöglichen.

Contents

CHAPTER 1

Introduction

Casting is a versatile process for producing engineered work pieces by pouring melt under high pressure into moulds, then letting to solidify and breaking out of the mould subsequently [93]. A mould is a hollow object of a certain material (e.g. fireclay, ceramic, limestone, plumb or steel) whose cavity defines the geometry and shape of the component to be produced. Moulds can be designed to produce complex shapes with a high degree of accuracy and repeatability. Therefore, casting is suitable for a wide variety of complex shapes that would be difficult or uneconomical to be constructed by other manufacturing processes. Castings can be found in thousands of consumer, commercial and industrial products - ranging from automobiles to toys.

The most common casting process is *metal casting* [4], which is also one of the oldest metal working techniques in the world. Liquid metal or metal compositions are used as casting materials to be poured into a mould and then allowed to solidify (Figure 1.1). After being broken out of the mould, the work piece usually is finished with grinding, sanding, and polishing before being sold. A wide variety of materials can be used to make moulds for metal casting, whereas the material needs to be strong and durable enough to withstand the temperatures of hot metal (e.g. wood, limestone or ceramic). To do metal casting, people usually need access to a foundry, a facility which has been specially outfitted for work with hot metals. Some foundries are equipped to handle a range of metals, including metals which need a very high temperature for working.

By their nature, all cast bodies will exhibit to some degree *casting defects* [79]. Casting defects are defined as irregularities in the casting that are undesired. Defects that have an adverse effect on the casting's mechanical properties and usability have to be removed or corrected. Other defects that to not have this effect are also called *discontinuities* or *imperfections*. They can be tolerated since the casting still remains useful and in tolerance. *Shrinkage defects* occur during solidifaction when feed metal is not available to compensate for shrinkage. Open shrinkage defects are open to the atmosphere whereas closed shrinkage defects are irregularities that form within the casting. The formation of bubbles within the casting is called *gas porosity*. Melt can hold a large amount of dissolved gas which bubbles within the material as it cools down. Gas porosity may present itself on the surface of the casting or the pores may be trapped

Figure 1.1: *Metal casting process. Liquid metal is poured into a mould where it is then let to solidify. Image property of the Austrian Foundry Research Institute.*

inside the metal. Nitrogen, oxygen and hydrogen are the most encountered gases in cases of gas porosity. To prevent this kind of defect, the casting material may be melted in a vacuum, in an environment of low-solubility gases or under a flux that prevents contact with the air. Another possibility is to add materials to the melt that remove gases (e.g. oxygen can be removed from copper by adding phosphorus). Hydrogen formations can be avoided by first drying the mould before puring the melt. Metal casting defects especially include *misruns*, *cold shuts*, and *metallurgical defects* [38]. Misruns occur because the liquid metal does not completely fill the mould's cavity, leaving unfilled parts. Cold shuts emerge from the fact that two fronts of liquid metal do not fuse properly during solidification. Both types of defects can be caused by certain characteristics of the casting material (chemical composition, low temperature) or by the form of the mould's cavity (too narrow cross-sections, sharp edges, insufficient ventilation). Two types of metallurgical defects exist: *hot tears* and *hot spots*. Hot tears are failures in the casting that occur as the casting cools and can be prevented by proper mould design. Hot spots are areas on the surface of casting that become very hard because they cooled more quickly than the surrounding material and can be avoided by proper cooling practices.

Whenever a new type of work piece should be produced, it is necessary to design a new pattern first. The development of a new mould consists of three major stages: *product design*, *tooling development* and *foundry trials* [4]. During the product design stage product requirements like overall shape, individual features, wear resistance and material composition are analysed. The tooling development stage can be further classified as the design of the mould's main cavity and leads to the generation of a new pattern. Within the last development stage trial castings are produced in the foundry. The trial casting objects are inspected to see if they include external or internal defects. Based on the results, the manufacturing process may be modified and certain

parameters (like the melt temperature or the mould's form) may be adopted to improve casting quality. Afterwards the newly constructed mould can be released for serial production.

Often non-destructive testing techniques are used to inspect the test samples of a foundry trial run. Non-destructive testing (NDT) defines a group of analysis techniques used to evaluate the properties of materials or components without causing damage (Section 2.1). It is an essential tool in construction engineering and manufacturing (e.g. automotive industry, heavy machinery industry and plant industry). Since NDT does not permanently alter the object being inspected it is often the only possibility to evaluate the overall product. Common NDT methods include ultrasonic (UT), magnetic-particle (MT), radiographic (RT), but also industrial Computed Tomography (CT) scanning. This is a process which uses flat X-ray images to produce 3D representations of the scanned object.

1.1 Problem Statement

Foundry engineers are interested in developing an automatic method for analysing the results of the foundry trial stage when developing a casting. Foundry trials consist of a series of test castings that have all been produced using the same mould and identical production process parameters. The test castings need to be inspected carefully to identify inner and outer defects, to detect possible problems within the production process. The defect detection is nowadays often done by NDT (Section 2.1). Since 3D CT has become common in NDT in recent years [9], researchers already concentrated on the topic of defect detection within industrial CT volumes (Section 2.1.1). Defect detection approaches can be used to analyse the occurrence of defects within castings, whereas these algorithms are intended to be applied to individual castings only. However, when analysing foundry trials, defect occurrence within the whole series of test castings has to be evaluated.

The goal of the analysis is to determine whether defects occur within the castings and whether correlations between the castings can be found (Figure 1.2). Foundry engineers are interested in the fact whether

- regions exist that contain a high concentration of defects, or
- very large defects occur,
- that appear in more than one of the castings of a certain foundry trial.

The occurrence of defects in a number of parts means that the production process has a certain probability in producing defective parts. Depending on the number, types and size of defects, this may necessitate adjusting the production process parameters (e.g. metal temperature, ventilation or the mould's shape).

The input data for a foundry trial analysis consists of the digitised test castings and their corresponding defect detection information. The castings are inspected by NDT, therefore the defect detection approach has to adapted according to the NDT technique used (e.g. CT or UT). The foundry trial analysis is independent of the chosen NDT technique, because it is based only on the results of the defect detection algorithm.

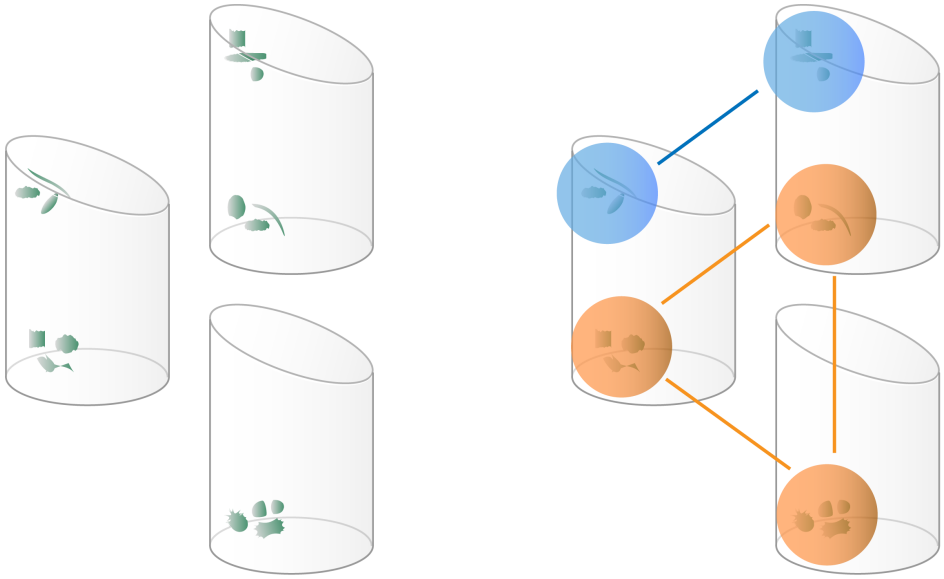


Figure 1.2: *A foundry trial for a new pattern consists of a series of test castings which were produced using the same mould, therefore they all exhibit the same shape and dimensions. Each of the castings may contain castings defects, which are illustrated as dark regions in (a). For analysing foundry trials, it is important to know whether regions of high defect concentration exist that occur in more than one test casting. Such regions are illustrated in (b) - defects in the blue region are present in two of the test castings, whereas defects in the orange region occur in all castings of the foundry trial.*

Foundry engineers are especially interested in a visual representation of the analysis results. The regions or large features appearing in more than one test casting are marked so that they are immediately visible to the user. The basis for the visualisation is the shape of the casting, since this will later on represent the work piece to be produced. Based on the visualisation results foundry engineers are able to evaluate the uniformity of the production process and the suitability for serial production.

1.2 Pipeline Overview

The aim of this master's thesis is the implementation of a variability analysis pipeline for the foundry trial stage when developing a new mould. The analysis explores a series of 3D industrial CT volumes which represent test castings of a certain foundry trial. The CT data originates from a test data set produced by the Austrian Foundry Research Institute (Section 3.1). After analysing the test castings (Chapter 3), visual tools are provided to the foundry engineers for exploring the analysis results (Chapter 4). The different stages of the pipeline are illustrated in Figure 1.3.

In a first step defects have to be detected within the given test castings. For this the defect

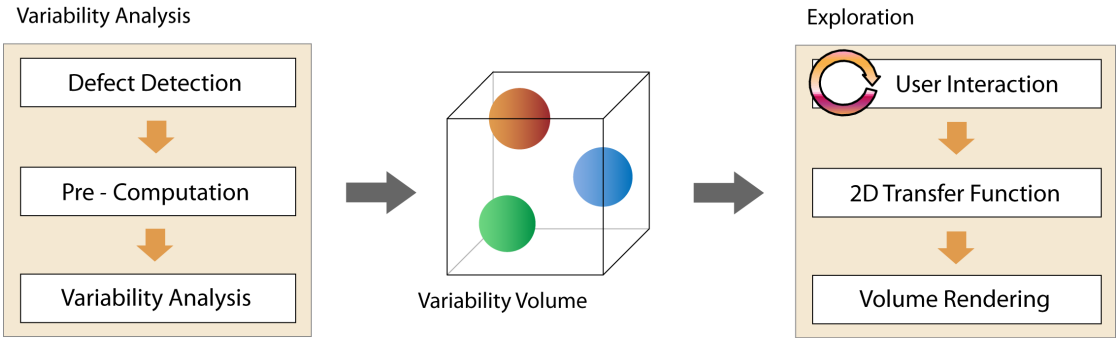


Figure 1.3: *Overview of the variability analysis pipeline presented in this thesis. The pipeline consists of two main stages: the variability analysis and the exploration stage. Within the variability analysis, defects returned by a defect detection mechanism may be pre-processed (e.g. clustered) and then transferred to the actual variability analysis mechanism. This results in the creation of a variability volume, which forms the basis for the subsequent exploration stage. For the user, this is the most visible part. Results are visualised using a 2D transfer function approach specified by the user's input parameters.*

detection approach described by Hadwiger et al. [34] is used, a new segmentation method to identify casting defects based on a region growing approach. This results in a list of defects for every available test casting. In the further course the variability analysis does either concentrate on individual defects, or on regions inside the objects with a high concentration of defects. The identification of such regions is done by a clustering approach, whereas the defects' centres of gravity are used as input data points. This results in a list of clusters of a certain size for every trial object. Then it is observed if any of the available clusters, or defects, can be retrieved in any other castings of the foundry trial. If so, this means that the casting process tends to produce errors in these parts of the castings. The most important and visible process step for the user is the exploration stage. It is used by the foundry engineer to decide on the next steps concerning the current foundry trial. The provided exploration techniques allow the user to view the results of the variability analysis.

1.3 Thesis Structure

This thesis is organized as follows: Chapter 2 contains some basic fundamentals considered important to the understanding of the implementation, together with an overview of the current state of the art. The defect detection algorithm where this thesis is based on is described within this chapter in Section 2.1.2. Chapter 3 discusses the implementation of the variability analysis and introduces the algorithms and data sets that have been used. Chapter 4 is dedicated to describe the visualisation of the results and how they can be explored individually by the user. A discussion and quantification of the results can be found in Chapter 5. The master's thesis then closes with a conclusion and some suggestions for further work (Chapter 6).

CHAPTER 2

Fundamentals and State of the Art

In this chapter the fundamentals and current state of the art concerning the main topics of this thesis are discussed. This involves several topics from visualisation like non-destructive testing, industrial CT data, 3D volume rendering, GPU-based direct volume rendering and cluster analysis.

The first Section (2.1) is dedicated to an introduction to non-destructive testing techniques in the special case of industrial CT data. The principle of data acquisition from industrial CT is described as well as defect detection algorithms and non-destructive testing techniques within the field of foundry technology. Section 2.2 concentrates on different aspects of 3D visualisation of volume data. This involves 3D visualisation of industrial CT data, GPU-based rendering and 1D as well as 2D transfer functions. The last Section (2.3) describes the two cluster analysis algorithms used in this thesis, mean-shift and k-means clustering.

2.1 Non-destructive Testing

Non-destructive testing (NDT) specifies a wide range of analysis techniques used in science and industry to evaluate the properties of a material, component or system without causing damage to it. NDT works for many different objects like machine parts, cast metal, ceramics and compound materials, wood, as well as for plastics. Beside non-invasive techniques like ultrasonic, magnetic particle and radiographic approaches, computed tomography (CT) has become more and more important in NDT. Its use has created powerful new possibilities, but also new challenges in the field of NDT, which are described in Section 2.1.1. For NDT applications, the visualisation of data is just one part of the work flow. Other tasks like defect detection, computation of properties like material porosity and accurate comparisons have to be carried out beforehand. Defect detection, which is used to detect errors like cavities, pores, cracks, or inhomogeneities within industrial work pieces, is described in Section 2.1.2.

2.1.1 Industrial CT

X-ray computed tomography (CT) is an imaging method employing tomography where images are created by computer processing. Already a method most common in medicine, CT was subsequently extended and is now used for a variety of industrial tasks [40]. However, compared to medical CT, industrial CT uses a different principle which is referred to as *cone beam CT* (Figure 2.1). Such scanners utilize a narrow, focussed cone beam of radiation which scans an industrial work piece that is placed on a rotary element. At each angular position of a 360 degree turn a 2D image of the work piece is recorded. The complete series of penetration images allows reconstructing the 3D representation of the scanned work piece. This allows for much faster data acquisition, as the data required for multiple slices can be acquired in one rotation.

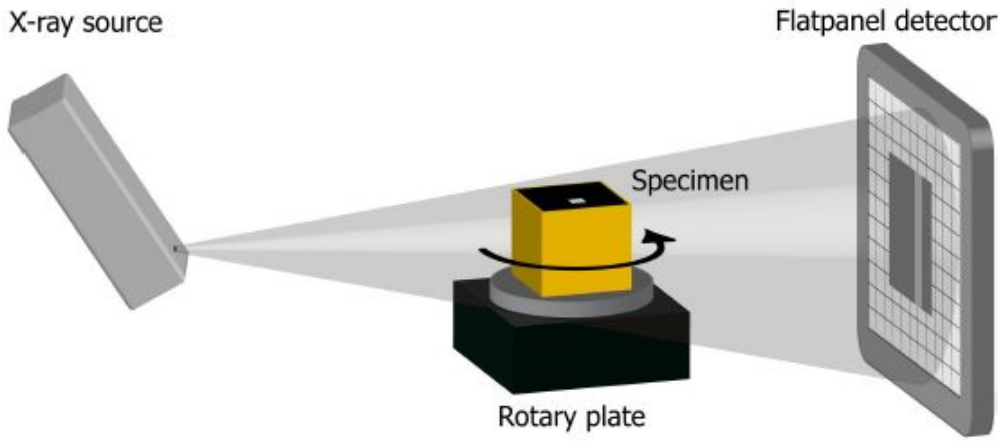


Figure 2.1: *Industrial CT scanner in cone beam design. The x-ray (left) sends out a narrow cone beam of radiation which scans a work piece placed on a rotary element (centre). At each angular position of a 360 degree turn a 2D image of the work piece is recorded (right) [40].*

Another advantage of industrial CT over medical CT data is its accuracy. Since in this field only objects are examined which are not harmed by X-rays, industrial CT systems can work with higher resolutions than those employed for medical purposes [52]. Industrial CT scanners are continuously advanced in order to achieve higher resolutions [40]. This allows highly detailed measurements and examination of the scanned work pieces.

2.1.2 Defect Detection

An important task in NDT is the evaluation of the properties of a certain material or work piece. For this, operations like defect detection, computation of properties like material porosity and accurate comparisons have to be carried out before visualisation. This Section is dedicated to the field of defect detection only, since this is the basis for the variability analysis presented in this master's thesis. Furthermore, only defect detection on CT data will be considered.

Related Work

Defect detection describes the task of automatically identifying discontinuities within materials, which can be cracks, holes or inclusions. Defects may be of various shapes and sizes. The first defect detection approaches working with CT data concentrated on the evaluation of 2D images [6,41]. Later approaches already enabled defect detection within 3D CT data [31,68,91]. Huang et al. [46] were the first to introduce region-growing into the field of defect detection for non-destructive testing of CT data. Hadwiger et al. [34] developed an approach for defect detection and interactive volume exploration. Unlike the approach by Huang et al [46], their technique allowed to interactively explore the volume. The segmentation was not re-computed during visualisation, but only if different parameter settings should be used.

Defect Detection and Quantification by Hadwiger et al.

The defect detection and interactive volume exploration approach by Hadwiger et al. [34] is explained in more detail in this Section, since the variability analysis described in this thesis builds on it (Section 3.2).

The presented pipeline approach consists of two stages: a complex *pre-computation* stage, and an interactive *exploration* stage. Although the exploration is the conceptually most important and most visible part for the user, the basis for interactivity has to be determined in a pre-computation step first. This partition of the two stages enables to interactively explore the volume. The segmentation is not re-computed during visualisation, but only if different parameter settings should be used.

During the pre-computation stage defects are detected via region growing in multiple passes. In each pass, for each seed candidate, a region is grown as far as possible given two size thresholds. These limits assure that the detected defects are larger than a given minimum as well as smaller than a given maximum size. If a grown region satisfies these two criteria, it is transferred into a so-called *feature*. In all region growing passes after the first one, completely new features are started using seed candidates as well as existing features are grown further until they reach a region growing limit (Figure 2.2). The thresholds are adapted accordingly in every region growing pass to allow new features to be created and existing features to grow further.

The region growing progress for each voxel is recorded in *feature size curves* and can be later retrieved during the exploration stage. Starting at a value of 0, the size of the emerging features is recorded in each iteration step. Seed voxels that start new features record the entire growth curve for this feature. Additional voxels that become part of this feature share the feature growth curve of the seed voxel. This leads to one representative size curve for every feature, which is stored in a *feature growth table*. Each row in this table illustrates the sampled growth curve for a feature over time, whereas each column represents a time step. The feature growth table is stored in a 16-bit two-channel 2D texture for rendering. All per-voxel information needed to reconstruct full feature size curves is stored in a 3D *feature volume*. For the visual exploration, this volume is stored in a 16-bit two-channel 3D texture. During rendering, texture fetches from the feature volume and the feature growth table yield everything needed to reconstruct the voxel's feature size curve.

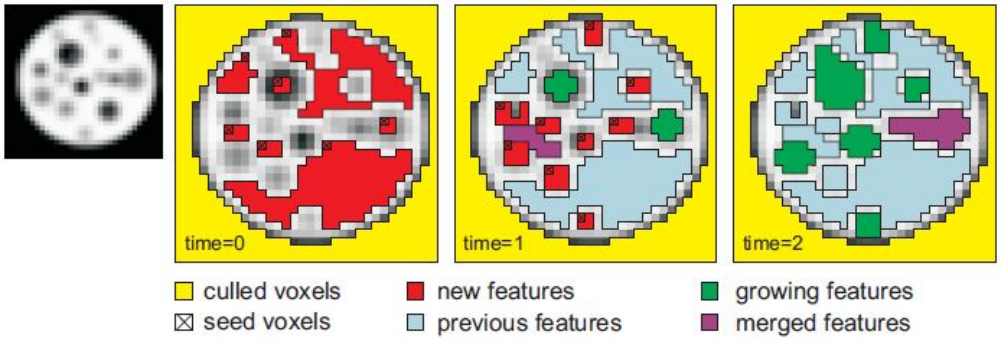


Figure 2.2: *The region growing of the pre-computation step is performed in multiple passes. In each pass, new features can be created or features may merge. Existing features may grow in all passes except in the first one [34].*

The exploration stage is the most visible one for the user where it is possible to classify features based on certain parameters as well as pick single features. The current classification is shown in a 3D volume view in real-time. The feature regions are mapped to colour and opacity based on user-defined selections. An exemplary visualisation can be seen in Figure 2.3.

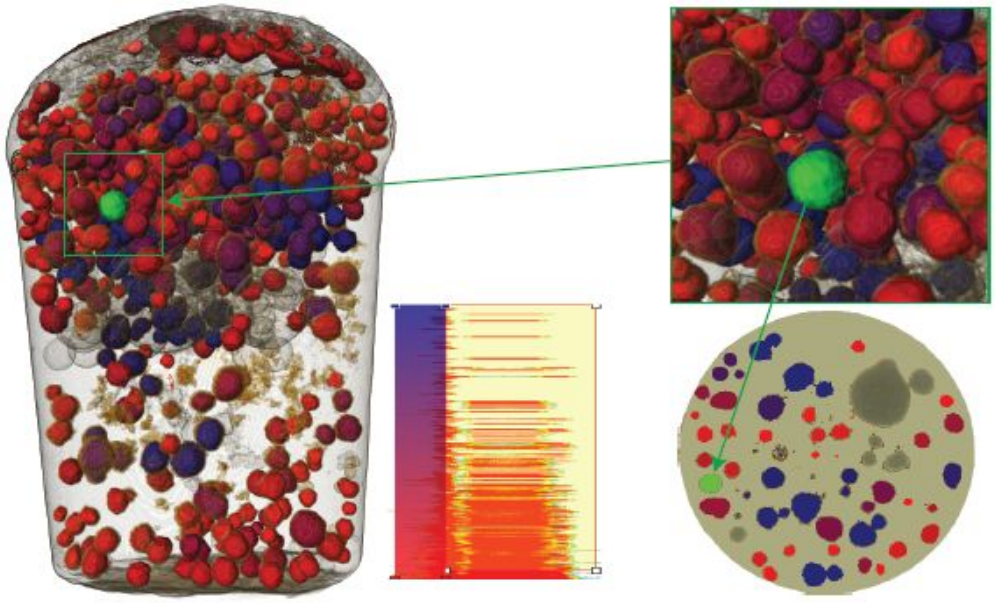


Figure 2.3: *Feature exploration example. The feature sizes are visualised with a colour gradient from red (small) to blue (large), whereas very large features have been set to transparent. The user-defined selection, which is done by using widgets in a 2D transfer function over feature size and voxel density, can be seen in the middle of the Figure. Furthermore, a single feature has been picked which is marked in light green [34].*

2.1.3 Non-destructive Testing for Metal Castings

Casting is a manufacturing process where melt is poured into a mould and let to solidify. Afterwards, the new work piece is broken out of the mould. The most common casting process is *metal casting* where metal or metal compositions are used as casting materials. Usually a foundry is needed to do metal casting that is equipped with facilities to handle hot metal. By their nature, all cast bodies will exhibit to some degree *casting defects*. They define conditions in a cast body that affect the object's mechanical properties and usability and therefore must be corrected or removed. Further information on casting can be found in Chapter 1. Several approaches have been adopted to enable automatic evaluation of castings. This Section concentrates on NDT approaches in the field of metal casting, since this is the topic relevant for this master's thesis.

The first automated inspection approaches by Borener and Stecker [8], Hecker [39] and Herold et al. [42] were based on single 2D X-ray images. Mery and Filbert [69, 70] extended these approaches to track defects over a sequence of X-ray images (Figure 2.4). The casting inspection approaches implemented for 2D images were later extended to be used with 3D computed tomography (CT) data [9]. Hytros et al. [48] proposed an approach where CT data is used to monitor the solidification progress in aluminium castings. Simon and Sauerwein [87] used CT data to make inferences on the quality of the produced castings. Pabel et al. [71] implemented an approach to verify the impact of a suboptimal melt quality on the static and dynamic mechanical properties of aluminium castings.

Whenever a new type of work piece should be created, it is necessary to create an appropriate mould first. Georgeson et al. [33] proposed an approach where the process of mould development is supported by the use of CT data. However, use of CT during mould development is still an open line of research. The most recent applications concentrated on developing techniques for flow simulations to predict how the newly created mould will behave during casting [47, 78, 92]. Kor et al. [56] proposed a formal optimization method based on a parameter system that formally describes the complexities of the mould development process.

2.2 3D Volume Rendering of Industrial CT

Volume rendering describes a set of techniques used to display a 2D projection of a 3D discretely sampled data. It comprises several rendering approaches such as *direct volume rendering*. This Section concentrates on the topic of volume rendering of industrial CT data sets, since this is the topic relevant for this master's thesis. An analysis of all recent findings in the area of 3D volume rendering will go beyond the scope of this Section, wherefore the reader may refer to previous work [12, 35, 62] for a brief introduction to volume rendering techniques.

As mentioned in Section 2.1.1, a great advantage of industrial CT is the use of high ray densities that guarantee high accuracy within the data. Industrial CT volumes are generally quite large with voxels commonly stored with 16 bits of precision. This leads to several hundred MB to one or more GB of raw data per scan which need to be processed and analysed. Therefore, approaches had to be developed that allow this huge amount of data to be processed. Reinhart et al. [80] and Wilson et al. [98] implemented efficient representation techniques to allow in-

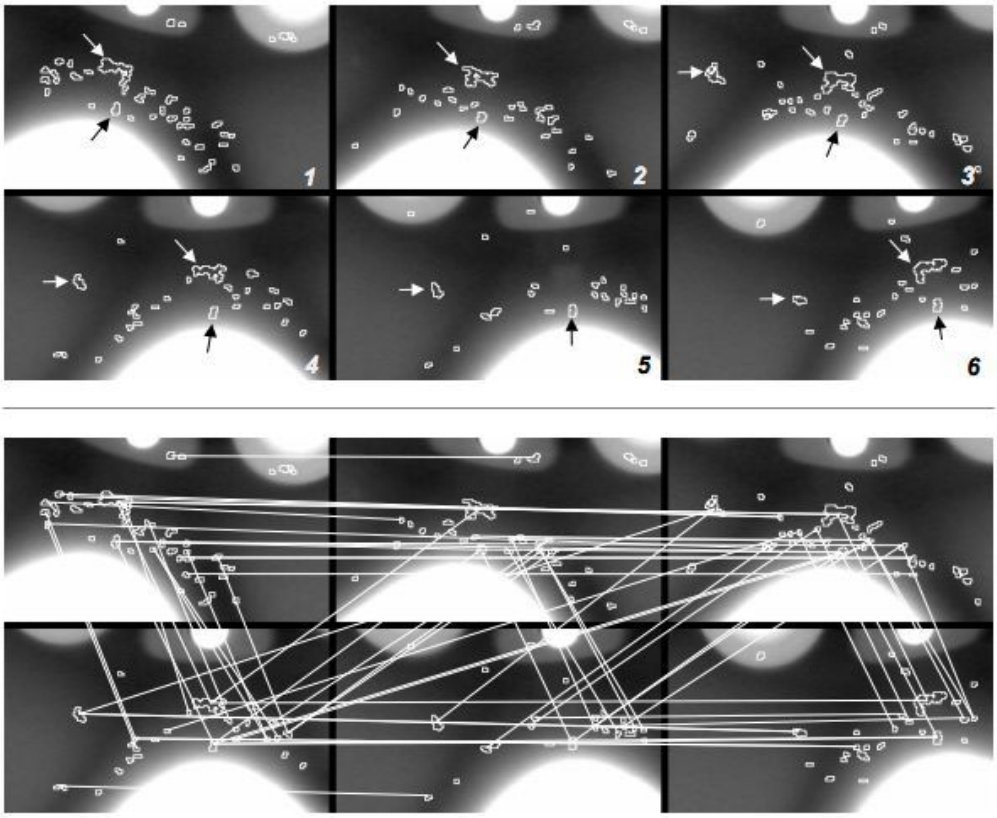


Figure 2.4: *Tracking of defects over series of X-ray images. In the first image (top) potential defects are detected within a series of X-ray images of a casting (arrows indicate false detections). The potential defects are then tracked over the series of X-ray images based on their spatial location, which can be seen in the second image (bottom) [70].*

teractive exploration. Huang et al. [46] developed a set of visualization techniques for feature defect detection and modelling that is able to handle high resolution industrial CT data. Frey and Ertl [28] proposed a parallel multi-resolution volume rendering approach for visualizing large data sets. Hadwiger et al. [36] implemented a brick caching approach to overcome the texture size limitations on the GPU. The volume textures are divided into smaller bricks and only the active bricks are packed into a single 3D brick cache texture which is loaded into the GPU memory.

2.2.1 GPU-Based Direct Volume Rendering

Direct volume rendering (DVR) is a volume rendering technique where every sample value is mapped to a certain opacity and colour value. DVR enables the rendering of volume data without attempting to impose any geometric structure on it. Instead, it operates directly on the volume data and simulates the interaction between light and a volume. Several optical models can be

found that describe light interaction with volume densities considering different parameters like absorption, emission, scattering and shadows [67].

The mapping of volume densities to optical properties like opacity and colour is done through a *transfer function*. In the simplest case, this is a lookup table mapping a density value to a certain RGBA value (colour and opacity). However, this concept has been already extended to the concept of multi-dimensional transfer functions (see also Section 2.2.2).

The most popular approach for DVR is *ray-casting*, which was first proposed by Levoy [60] in 1988. It is an *image-based* rendering technique, since the computation emanates from the output image and not the input volume data. In this approach a ray is cast for every pixel of the image plane through the available volume data. The goal of ray-casting is to approximate the volume rendering integral for every viewing ray. In the first step, interior data values are sampled along the ray at evenly spaced positions. Since rays often lie in between several voxels, it is necessary to interpolate between nearby voxels to determine the sample value. In the next step, the classification step, colour and opacity values are assigned (using a transfer function) to the sample points along the ray, and afterwards they are shaded. At the end, during the composition step, the final pixel value is determined based on the defined contribution of classified and shaded sample values (Figure 2.5).

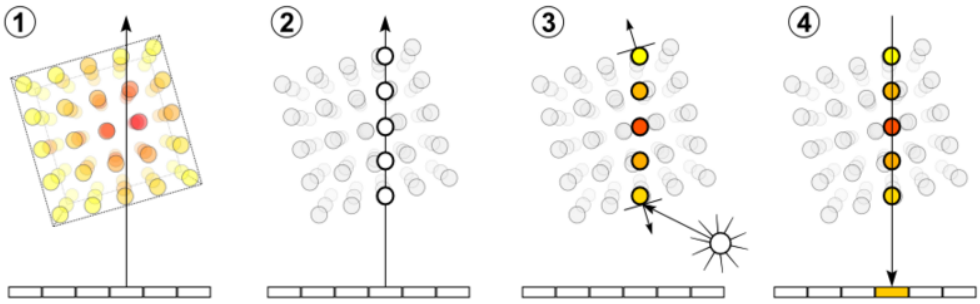


Figure 2.5: *Volume ray casting in four steps. In (1) a ray is cast for every pixel of the image plane through the volume. Then sampling points have to be determined along the ray, which often lie between voxel positions and therefore have to be computed through interpolation (2). In the classification stage (3), sample points are assigned a colour and opacity by using a transfer function, and they are shaded. At the end, the final pixel value is determined by compositing the sample values along the ray (4) [96]*

There are several possibilities in which order these steps can be performed. The term *pre-classification* refers to the process of assigning colour and opacity to the samples before interpolation. *Post-classification*, on the other hand, describes the process of applying interpolation before classification (as described in the previous paragraph). Pre- and post-classification will produce different results. The results of both rendering pipeline approaches will only be equal in case the transfer function is constant or identity [10].

The concept of ray-casting can be adapted to be executed by a fragment program on the GPU. *GPU-based ray-casting* benefits from the parallel architecture of modern GPUs as pixel

values can be computed in parallel. Ray-casting computations on the GPU are performed by executing the following steps:

1. *Ray Determination*: To step along a ray, its entry position, its length and its direction have to be computed. For this a bounding box of the volume is rendered with colour coding of each vertex position in object space. Two images are rasterised, where the first one rasterises the front faces of the geometry and the second one rasterises the back faces. The first image determines the ray entry positions.

2. *Ray Direction*: A direction texture is calculated by the difference of the back faces and the front faces (Figure 2.6). The ray directions are stored in the RGB values of the textures and the alpha values determine the ray lengths.

3. *Ray Traversal*: The fragment program now uses the calculated textures to initialise rays and determine sampling points along them. When fetching samples, hardware-native tri-linear filtering is used. The fetched density values are mapped to colour and opacity according to the transfer function settings.

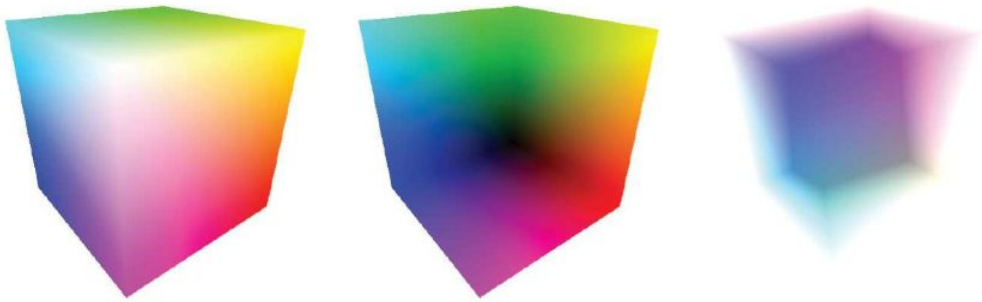


Figure 2.6: *Setup for GPU-based ray-casting. From left to right: Front faces, back faces, and direction texture [84].*

Several performance optimisations can be defined to avoid unnecessary computations, like *empty space skipping* or *early ray termination*. For empty space skipping, the volume is divided into smaller sub-blocks containing a fixed number of voxels. Only non-empty blocks form the data-dependent bounding geometry which is passed to the GPU. Early ray termination is used to stop the ray traversal in case it has left the volume or when the composited opacity value reaches a certain threshold.

The first GPU-based ray-casting algorithms were implemented by Roettger et al. [82] and Kruger and Westermann [58]. They used multi-pass ray-casting where the ray traversal process is still initiated by the CPU. Several approaches exist that propose different applications for GPU-based ray-casting, for example for the rendering of quadratic surfaces [86], for landscape visualisation [66] or for rendering virtual globes [22]. Scharsach [83] implemented a GPU-based ray-casting approach that allowed fly-through applications like virtual endoscopy. Other applications concentrated on the GPU-rendering of large data sets [45] and on industrial CT in particular [36, 100].

2.2.2 2D Transfer Functions

In general, a transfer function is used in direct volume rendering to assign optical parameters like colour and opacity to values of a volume data set. The transfer function defines how different parts of the volume will be rendered, and therefore designates the amount of information that is displayed in the volume renderer. In the simplest case a transfer function T is defined by a lookup table based on the implementation described in Equation 2.1. $f(x, y, z)$ describes a scalar function that returns the density value at position (x, y, z). This density value is mapped to a colour defined by R, G, B and an opacity value α.

$$T(f(x, y, z)) = R, G, B, \alpha \tag{2.1}$$

The specification of an appropriate transfer function is a crucial task which is difficult to accomplish. There are two main reasons for this: Usual interfaces for setting transfer functions (based on moving control points defining a set of linear ramps) are not guided by the dataset in question. Furthermore, transfer functions are inherently non-spatial, since their assignment of colour and opacity does not include spatial position as a variable. 2D transfer functions overcome these problems in most instances. They allow better discrimination between various structures in the volume data, since more dimensions are involved. Therefore, the use of 2D, 3D or even multi-dimensional transfer functions became more and more popular in recent years [76].

The first 2D transfer function was actually proposed by Levoy [60] in 1988, who added the gradient as the second dimension to his transfer function implementation. The application for multi-dimensional transfer functions was initiated by Kindlmann and Durkin [53] who implemented a semi-automatical approach for transfer function design. The resulting transfer function was still one-dimensional, but they included the gradient magnitude and the second order directional derivative of the scalar field into their computations. True multi-dimensional transfer functions were first proposed by Kniss et al. [55]. They used a 3D transfer function to classify samples during volume rendering. Furthermore, the authors proposed an interactive user interface for multi-dimensional classification. A similar technique was later used by Correa and Ma [21] who implemented a visibility-driven approach where users can manage a complex set of transfer functions on the basis of histograms.

Several other 2D transfer function applications have been developed. Hladuka et al. [44] presented a curvature-based transfer function approach with the aim to distinguish between different shape classes. Their approach was later advanced by Kindlmann et al. [54] by combining an implicit formulation of curvature with a convolution-based reconstruction of the field (Figure 2.7). Applications can be found in the field of surface smoothing, visualisation of iso-surface uncertainty and non-photorealistic rendering. Another approach using multi-dimensional transfer functions for non-photorealistic shading styles has been adopted by Bruckner and Gröller [11]. Additional lighting models are used to combine different shading styles interactively in a single transfer function (Figure 2.8).

Other approaches aim at improving the quality of renderings in the medical field by applying 2D transfer functions. Zhou et al. [104] proposed an approach where distance information was combined with the original volume to assign optical properties. The user defined a focal point of interest to which the Euclidean distance of all sample positions is calculated. A similar distance-

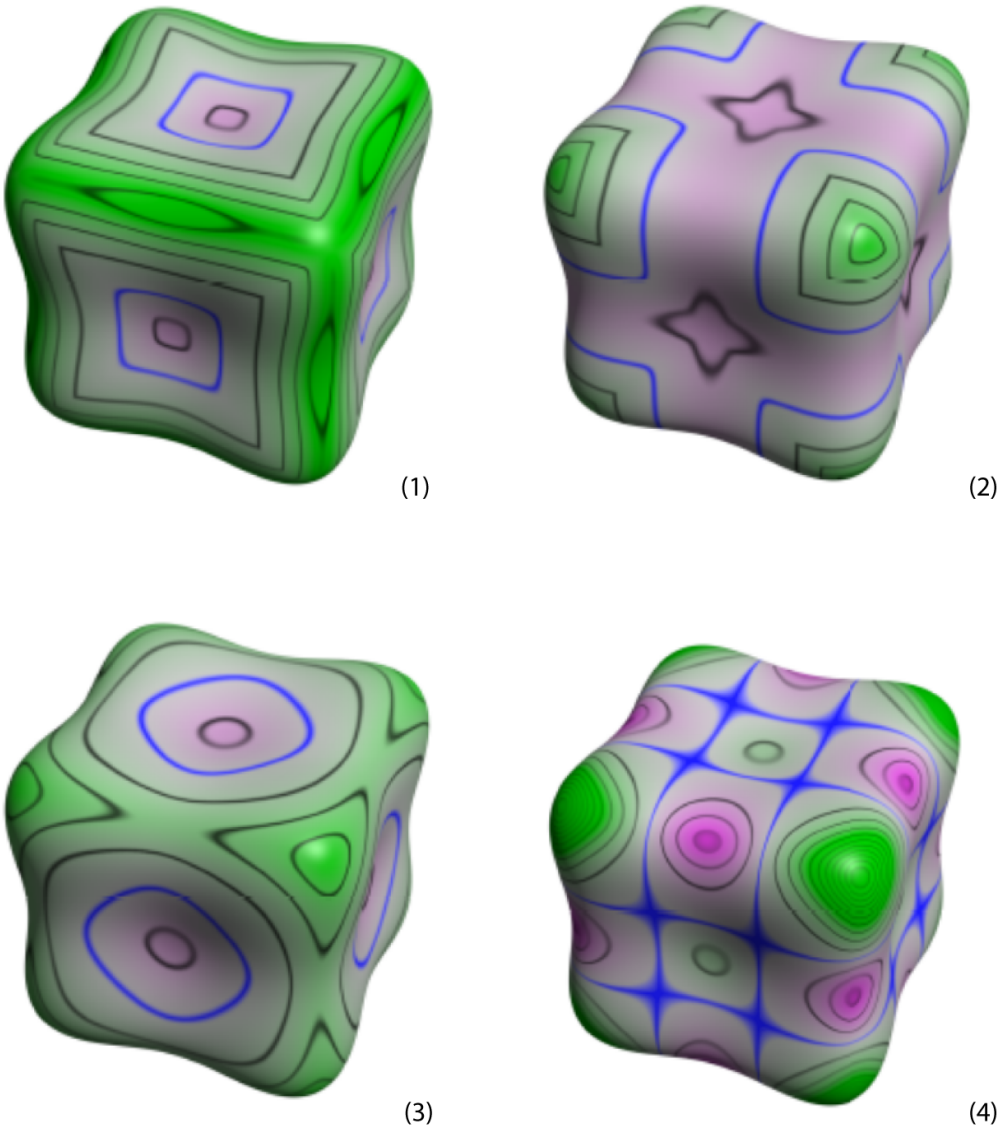


Figure 2.7: *Four different curvature measurements. First principle in (1), second principle in (2), mean in (3) and Gaussian curvature in (4). Zero curvature is illustrated in blue; all other iso-curvature contours are displayed in black [54].*

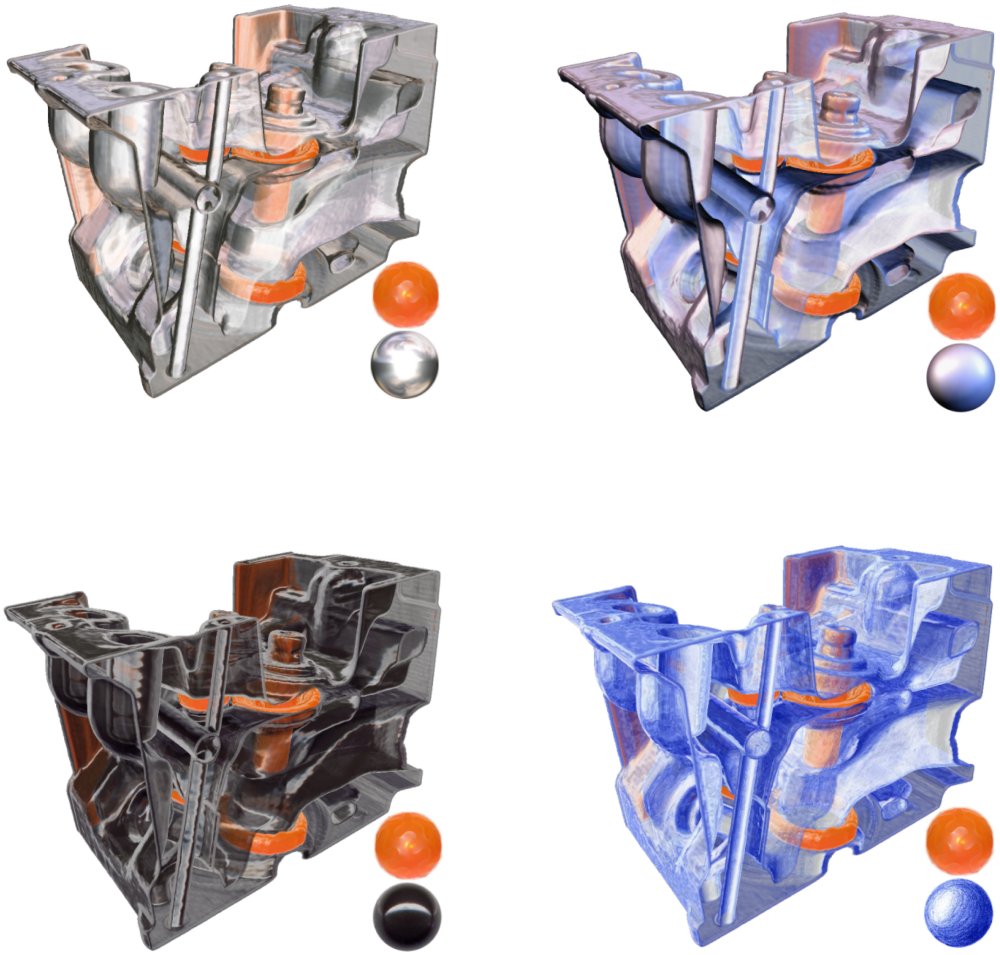


Figure 2.8: *Non-photorealistic rendering using multi-dimensional transfer functions. Different non-photorealistic rendering approaches are combined with certain style transfer functions. Using these style transfer functions, a multitude of different shading styles can be combined in a single rendering. In this figure four different style transfer functions are illustrated, where the lighting models are shown in the bottom right respectively [11].*

based approach was implemented by Tappenbeck et al. [90]. Rezk-Salama et al. [81] provided interfaces with semantic information based on semantic models that are defined from reference data sets. The relevant structures were then represented by one or more transfer functions (Figure 2.9).

2.3 Cluster Analysis

Cluster analysis or clustering is the task of partitioning a set of objects into groups (called clusters) so that the objects in the same cluster are more similar to each other than to those in other

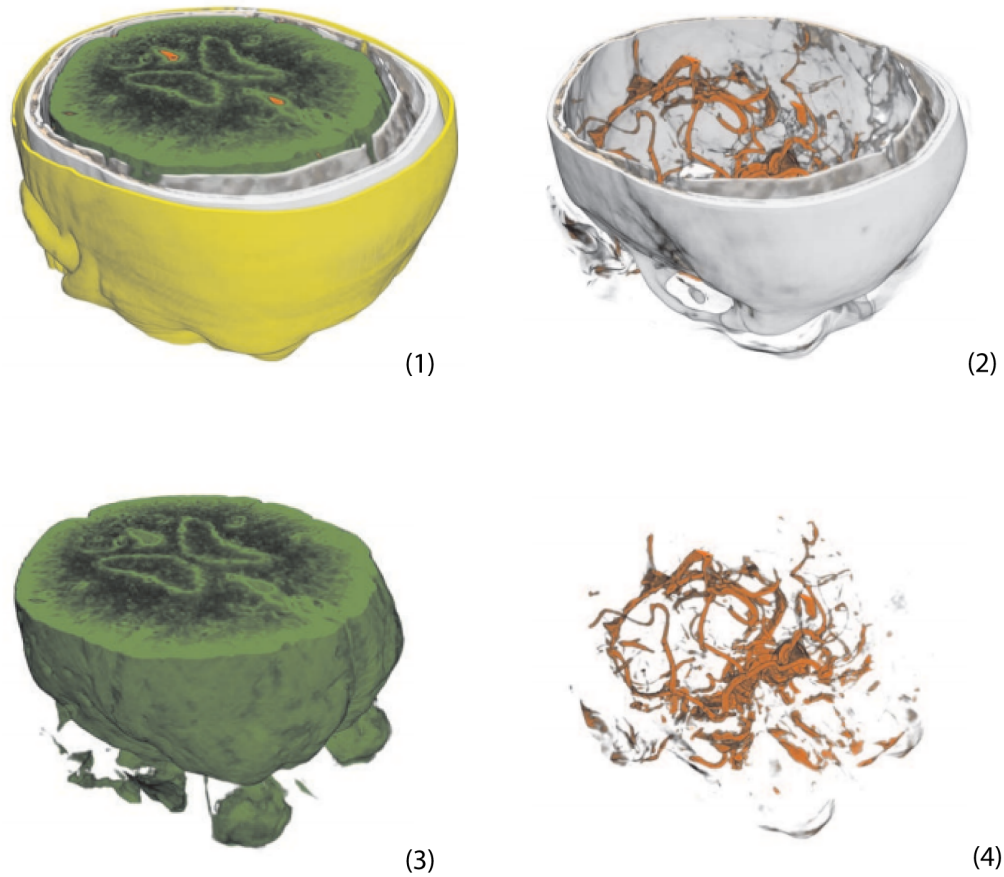


Figure 2.9: *Semantic rendering of medical images. In Semantic models each entity, defined from reference data sets, is represented by one or more transfer function primitives. This allow extracting relevant structures from a computed tomography (CT) angiography data set. In this example the list of entities comprise skin (1), bone (2), brain tissue (3) and blood vessels (4) [81].*

clusters. An important step is to select a distance measure which will determine how the similarity of two patterns is calculated [51]. Cluster analysis methods can be divided into the two groups of *hierarchical* and *non-hierarchical* approaches. Hierarchical methods construct a clustering tree, whereas non-hierarchical methods sequentially assign each pattern to a cluster [89]. The two clustering approaches described in this Section (*k-means* and *mean-shift*) both belong to the group of non-hierarchical methods.

2.3.1 k-means Clustering

k-means clustering is a well-known example of a non-hierarchical, non-parametric cluster analysis method. Originally adopted by MacQueen [64] in 1967, it became a „popular, simple and practically useful“ [89] approach for applying cluster analysis. The idea of k-means clustering is that the number of clusters K is already known before starting the analysis. To start the al-

gorithm, K initial cluster starting points (exemplars) are identified. The k-means method then consists of two main stages: In the first stage, all available patterns are allocated to one of the K clusters according to a certain distance measurement. The cluster exemplars are re-computed every time a pattern has been added to the cluster. This way it is assured that the cluster exemplars represent the centres of gravity of the patterns within a cluster (Figure 2.10). In the second stage, the exemplar positions are considered as final cluster positions and all patterns are allocated to the closest cluster respectively.

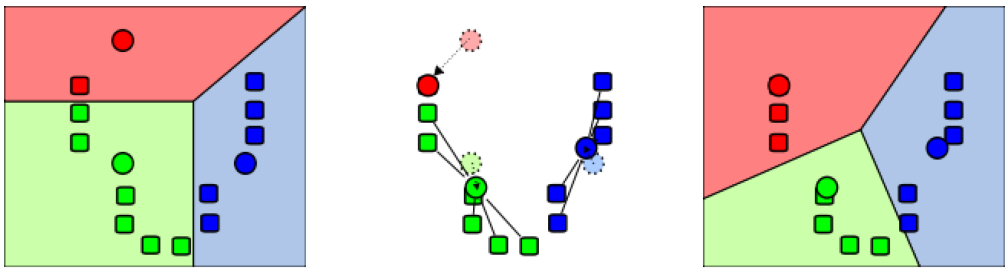


Figure 2.10: *k-means iterations. In the first step (1) patterns are assigned to their nearest clusters respectively, whereas the cluster exemplars are illustrated with circles. Then the exemplars are re-computed to represent the centres of gravity of the cluster patterns (2). The patterns are re-assigned to their nearest clusters respectively (3) according to the new positions of the cluster exemplars. These three steps are iteratively repeated until the final cluster positions have been found [95]*

In summary, the k-means algorithm consists of the following steps [89]:

1. Define the number K of clusters
2. Initialise the cluster starting points $v_1, v_2, ...v_K$, where some patterns are chosen to serve as cluster starting points.
3. First pass: Allocate patterns to the K clusters (do not process those patterns that were used to initialize clusters). Re-compute the cluster exemplars after a pattern has been added to the cluster. Repeat this step until the cluster exemplars do not change any more.
4. Second pass: Let the final exemplars be exemplars of the resulting clusters. Allocate all patterns to the final clusters by using the same distance criterion as on the first pass.

Lloyd [63] proposed a local search solution for implementing the k-means algorithm in 1982 that is still widely used today [1, 37, 97]. Due to its simplicity, the k-means method has its limitations and many variations exist. Kanungo et al. [50] adapted the classic k-means approach to use a kd-tree as the only data structure to improve the algorithm's performance. Elkan [25] used the triangle inequality to accelerate the algorithm. Frahling and Sohler [26] extended the classic k-means algorithm by using core sets (i.e. small weighted sets of points that approximate the original point set). Furthermore, the concept of k-means has been adapted for parallel programming [57, 102].

Especially the accuracy, but also the speed of k-means can be greatly improved by choosing the initial starting points in another mode than just randomly. Arthur and Vassilvitskii [3] implemented their *k-means++* algorithm in 2006 which follows the implementation steps of the classic k-means algorithm, but uses a more sophisticated way of choosing the initial staring points. In general, the initialisation approach aims at spreading the K initial cluster centers away from each other. It consists of the following steps:

1. Choose one center randomly from the given data points.
2. For each pattern v, compute the distance $D(v)$ between v and the nearest starting point that has already been chosen.
3. Choose one new data point as a new starting point. A point v is chosen as a starting point with probability proportional to $D(v)^2$. This way it is ensured that the new data point is probably far away from the already chosen starting points.
4. Repeat Steps 2 and 3 until K starting points have been found.

After all initial starting points have been found, standard k-means clustering is used to find the final cluster positions. However, the improved seeding method gives considerable improvements in the final error of k-means. It successfully overcomes some of the problems associated with other ways of defining initial cluster-centres for k-means clustering [59].

2.3.2 Mean-Shift Clustering

Mean-shift as originally introduced by Fukunaga and Hostetler [30] is a non-parametric, iterative procedure for seeking the maxima (i.e. modes) of a density function given by a set of discrete data samples. The method was revisited by Cheng [13] to develop a more general formulation and to demonstrate its potential use in clustering and global optimization problems. Comaniciu et al. [16, 17] adopted this idea and introduced the use of the mean-shift approach for a number of image processing problems, including segmentation, discontinuity-preserving smoothing and - finally - clustering. *Mean-shift clustering* allows grouping a set of points without having prior knowledge of the number of clusters in the data set [18].

Mean-shift finds the densest data regions, which are defined by the maxima of a probability density function given by a set of discrete data samples. It avoids estimation of the probability density function by estimating the density gradient instead. The procedure uses search windows (i.e. kernels) which are moved in the direction of the strongest increase in density until convergence. The final locations of the kernels define the density maxima (i.e. modes) of the underlying probability density function of the data.

A mean-shift kernel is defined as decreasing function of the distance from a given point (mean) to points in the data. For initialisation, the kernel is positioned randomly over the data and its centroid is identified. Then it is moved to the location of the identified centroid, which now represents the kernel's mean. The vector determining the kernel's new position is called mean-shift. The re-calculation of the kernel's position is repeated until it does not change any more (i.e. the mean-shift turns zero). A data set can be partitioned into clusters by defining several initial sample points.

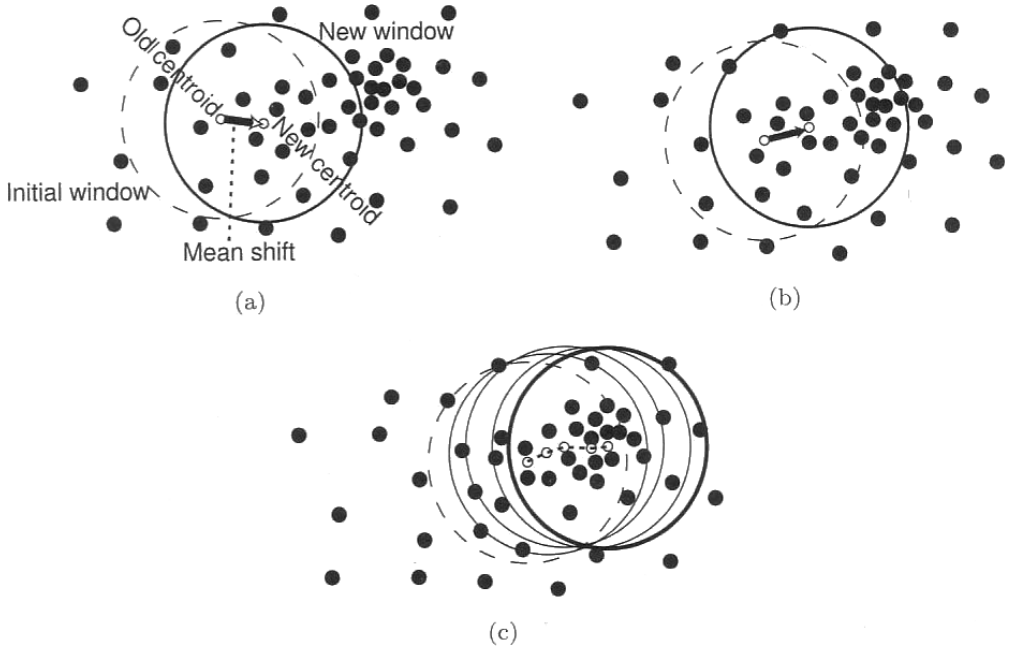


Figure 2.11: *Mean-shift iterations. The initial region of interest is positioned randomly over the data in (a). Then its centroid is determined, and the new region is moved to the location of the identified centroid. The vector determining the positional change is the mean-shift. In the next step (b), a new mean-shift vector is determined and the region is moved accordingly. These steps are repeated until convergence. The final location identifies the local density maximum (i.e. mode) of the probability density function (c). In this Figure it can be seen that patterns covered by a cluster in the initial stage may not be part of this cluster any more when the final position has been found [89].*

A certain mean-shift vector is always proportional to the gradient of the underlying probability density function. In each further iteration step the weight of nearby points is used for re-estimation of the kernel's mean which moves the kernel accordingly (Figure 2.11). Mean-shift vectors are determined in the remaining steps until convergence. In regions of low density the steps of the algorithms are big. The steps decrease as the kernel is moved towards the local maxima. Traditionally radially symmetric kernels are used which are isotropic in shape, although other kernels could also be employed since the mean-shift algorithm does not constrain their shapes [94].

The guaranteed convergence of the mean-shift algorithm to the local maximum is obtained due to the adaptive magnitude of the mean-shift vector. The magnitude of the mean-shift vector always converges to zero. This eliminates the need to choose adequate step sizes for the mean-shift iterations [18]. The convergence speed, however, depends on the kernel employed. For a uniform kernel convergence is achieved in a finite number of steps. Whenever data points are weighted, the mean-shift procedure is infinitely convergent [89]. A small lower bound value of change between steps may be used to stop the convergence process.

Implementations of mean-shift clustering have been proposed by Freedman and Kisilev [27] and Phat et al. [77]. Georgescu et al. [32] as well as Xiao and Liu [99] proposed solutions to the problem that the high computational complexity of the mean-shift procedure limits its practical application in high dimensional and large data sets. Mean-shift clustering has also been adapted for parallel programming to improve its performance [72, 103]. Other applications of the mean-shift procedure are located in the field of real-time object tracking [19] and blob tracking [14, 74].

2.3.3 Complexity

As stated in the first part of Section 2.3, the similarity of patterns grouped together in a cluster is determined by a certain distance measurement. This will influence the shape of the resulting clusters as well as the computational complexity of the approach. Common distance functions are:

- *Euclidean distance*: The 2-norm distance between two points given by the Pythagorean formula.
- *Manhattan distance*: The distance between two points is given by the sum of the absolute differences of their coordinates.
- *Mahalanobis distance*: This distance function differs from the Euclidean distance in that it takes into account the correlations in the data set and is scale-invariant. When the Mahalanobis distance is used, the clustering algorithm will form hyper-elliptical clusters.
- *Hamming distance*: The Hamming distance between two strings of equal length is defined by the number of positions at which the corresponding symbols are different.

Furthermore, the aim of a cluster analysis is to minimize the *within-cluster variance* e_K^2 for each cluster as defined in Equation 2.2. n_k is the cardinality of cluster k and $x_{i,k}$ is the i-the pattern of cluster k. The reader may refer to Sonka et al. [89] for a brief deduction of e_K^2.

$$e_k^2 = \sum_{i=1}^{n_k} (x_{i,k} - m_k)^T (x_{i,k} - m_k) \tag{2.2}$$

The square error of the clustering results E_K^2 is then defined as the sum of the within-cluster variances (as defined in Equation 2.3). Therefore, the aim of a clustering algorithm is to minimize the square error criterion E_K^2.

$$E_K^2 = \sum_{k=1}^{K} e_K^2 \tag{2.3}$$

In this thesis, the *Euclidean distance* is used as distance function for implementing the clustering. Together with the minimization argument mentioned in Equation 2.3, this leads us to the following definition:

„The problem of partitioning a set of vectors in Euclidean space into subsets (clusters) with the criterion of minimizing the sum of squares of the distances from the cluster elements to the cluster centers [...] is known in the literature as the MSSC (Minimum-Sum-Of-Squares Clustering) problem.“ [23].

It can be shown that the one-dimensional variant of the MSSC problem can be solved in polynomial time [23], but all other variants (even the two-dimensional case) are NP-hard [2, 65]. Therefore, it is not possible to efficiently provide an optimal clustering result in an automated way.

CHAPTER 3

Implementation

The goal of the variability analysis pipeline is to help foundry engineers decide on a cast mould's suitability for serial production (Section 1.1). The goal of the complex analysis stage, whose components are described in this chapter, is to inspect the given volume data sets and make investigations of the reliability of the production. Although it is a technically complex and time consuming task, only minimal user input is required for this stage. The variability analysis aims at analyzing a foundry test run, therefore the input data consists of a set of volume data sets (i.e. more than one). These data sets represent a series of digitised castings that have all been produced by the same mould. All analysis computations are based on an initial defect detection step [34] which identifies given casting defects in the volume data sets. The subsequent variability analysis step aims at identifying volume parts with certain criteria that recur within other volumes of the foundry test run.

The first Section (3.1) concentrates on the input data used for implementing the prototype of this thesis. The initial defect detection step is described in Section 3.2. Based on this data are the variability analysis computations, which are described in Section 3.3. The results of the analysis are stored in a way so that they can be used for the exploration, which is described in Section 3.4.

Terms definitions:

- *Prototype*: A prototype represents a new type of casting that is planned to be mass-produced. No digital representation of the prototype exists at this production stage. Therefore it has to be calculated from the available casting data sets (Section 3.4.1).
- *Test mould*: A test mould that generates the prototype needs to be produced before release into serial-production to test the entire manufacturing process.
- *Test runs*: Test runs are test casting processes using the test mould. The resulting castings have to be inspected carefully to identify possible problems in the manufacturing process.
- *Object*: Castings that belong to a certain test run are called objects. They have been digitised by a 3D industrial CT scanner to be able to inspect them by NDT techniques.

3.1 Data Acquisition

The variability analysis approach described in this thesis aims at analyzing a casting test run and it has been implemented to work with 3D industrial CT data. Therefore, it is assumed that the input data for this approach consists of a set of 3D CT scans (i.e. more than one) which represent a series of objects. There are no restrictions on the size and/or resolution of the scans or on the number of data sets. However, it is important that the size and resolution of the data sets are consistent across the input set. Otherwise it will not be possible to compare or integrate the data obtained from these different measurements. Furthermore, the spatial registration of the data sets was not part of the implementation, so it is assumed that the data sets are already aligned. The reader may refer to Hill et al. [43] or Penney et al. [75] for further information on volume data set registration.

The data sets used in this thesis were provided by the Austrian Research Foundry Institute (*Österreichisches Gießerei-Institut - ÖGI*). They created a test data set consisting of three objects. The development of defects was simulated by drilling holes into the objects (Figure 3.1). Four regions containing boreholes were created per object. It it has been taken care of that some of these regions occur at the same spatial location in every of the test objects. Afterwards the modified castings were digitised by using a 3D industrial CT scanner to provide volume data sets for the visualisation and analysis computations.

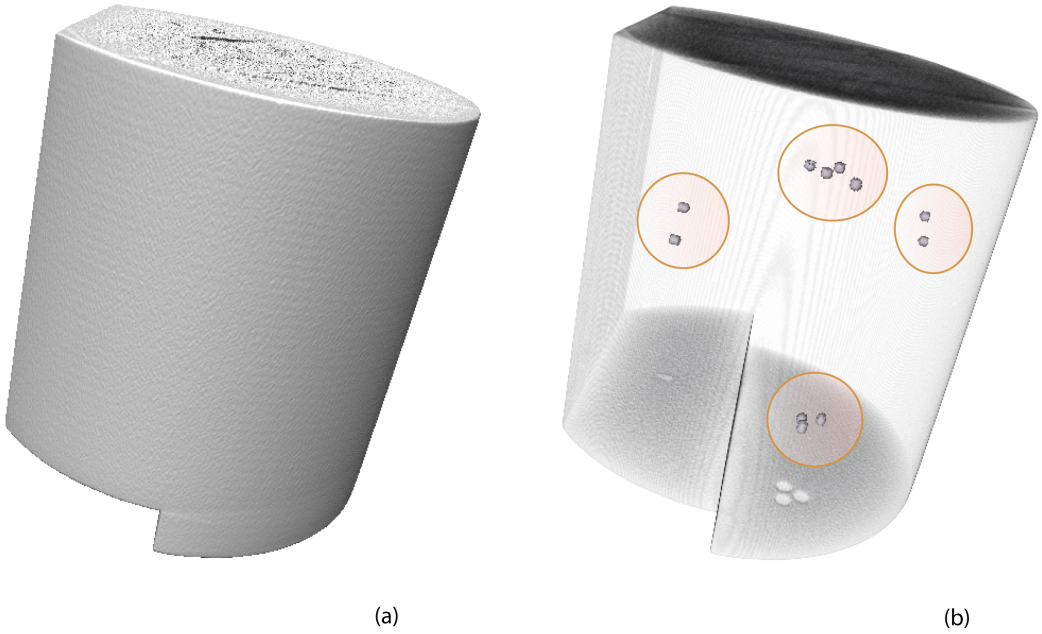


Figure 3.1: *Test run provided by the Austrian Research Foundry Institute for implementing the variability analysis. Three metal castings were produced using the same mould. After solidification holes have been drilled into the objects to simulate the development of large defects. The dimensions of the objects can be seen in (a), and the boreholes are marked in (b). Both figures have been produced by rendering the 3D volume data set representation of the objects.*

3.2 Defect Detection

The variability analysis supplies information about the distribution of defects over a series of test castings. Therefore, it is necessary to detect casting defects within the given data sets before the analysis can be started. Further information on defect detection can be found in Section 2.1.2. No information about possible casting defects in the given input data is known beforehand. The defect detection produced by the segmentation algorithm proposed by Hadwiger et al. [34] was used as the input data for this thesis. Defect detection is started separately for every volume data set, since it might be necessary to adapt parameters according to individual object parameters (e.g. different defect dimensions).

After applying the method, a list of defects (*features*) is created for every given casting object (see also Section 2.1.2 for a more detailed description of the algorithm). A *feature* consists of a set of connected voxels and is the virtual representation of a physical defect. Since all further computations in this thesis are based on virtual object representations (instead of physical parameters), the term *feature* will now be used to refer to casting defects.

Features can be classified by several parameters (e.g. feature size, roundness ...), and this information is used to control the feature exploration via a 3D transfer function approach. Using a stack of 2D histograms features can be selected or rather hidden by the user, based on certain parameters (Section 2.1.2). For the variability analysis, these parameters allow to individually select certain feature classes that should be used for the variability analysis computations. Since not all available defects influence the stability and safety of a casting, foundry engineers may decide to eliminate certain feature classes from further computations (Figure 3.2).

3.3 Variability Analysis

Variability analysis aims at finding object parts that exhibit certain criteria and recur within the given test run. Such object parts can either be areas of a certain size within the objects, but can also be represented by single data points. Based on the object parts that are used as input, two types of variability analyses can be identified: a *region-based* and a *feature-based* approach. The region-based analysis works with regions of high feature concentration, whereas the feature-based approach is working with individual feature voxels.

3.3.1 Region-based Variability Analysis

An analysis approach very favoured by foundry engineers is the region-based variability analysis. It first finds regions of high feature concentration within objects and then tries to track these regions over the complete test run. In case regions tend to appear in more than one object, this means that the manufacturing process has a certain probability to produce defects in these areas - which may necessitate a change in certain production process parameters.

Clustering

As a method to identify regions of high feature concentration we decided to use a clustering algorithm. Clustering approaches divide a set of data points into groups based on their similarity.

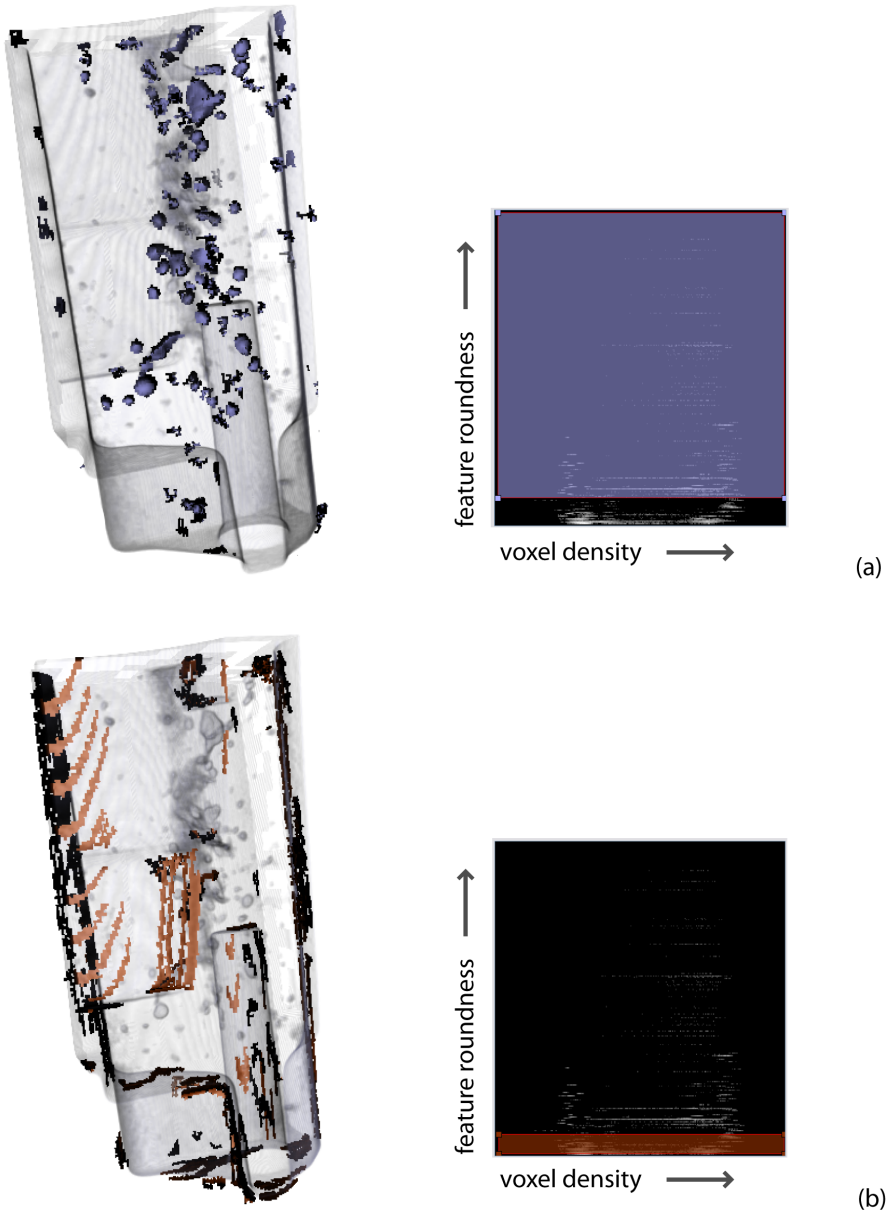


Figure 3.2: *2D histograms are used to explore features through a 3D transfer function. Histograms visualise certain feature parameters (here: voxel density against feature roundness) and allow the user to individually select features based on these parameters. A selection based on the roundness, like the one indicated in (a), would omit lenghty features like cracks or* cold shuts*. The deselected features are shown in (b). The widget size in the y-direction (roundness) defines which feature should be included in the selection. The widget dimension in the x-direction (voxel density) defines which parts of the feature should be visible. For the variability analysis, the foundry engineer can use these 2D histograms to refine the variability analysis by concentrating only on certain feature sets.*

The similarity measurement used for computing the clusters is the Euclidean distance (Section 2.3).

Features are not represented by a single data point, but by a connected set of voxels. To be able to cluster this information we decided to choose the centre of gravity as a distinctive representative for every feature. Equation 3.1 shows how the centres of gravity are computed: For a certain feature i, the sum of all voxel positions V_n is build and then divided by the total number of voxels v. Applying this formula to every feature f_i leads to a set of 3-dimensional vectors S_i. The 3-dimensional centres of gravity S_i are used as input for the data clustering algorithm.

$$S_i = \frac{1}{v} \sum_{n=0}^{v} V_n \tag{3.1}$$

Additional user-input is required to specify some clustering-specific parameters. We implemented two different clustering approaches, *mean-shift* and *k-means++*, to be able to compare the different results (Section 5.1.1).

Mean-shift is a method to seek the maxima of a density function given by discrete data samples. It uses a kernel of a certain size to iteratively re-estimate its mean based on the nearby data points (Section 2.3.2). The approach is started from a random list of sample points. The iteration is repeated until the mean's position does not change any more - then the mean represents the maxima of the density function at this point. Therefore, the user has to define two parameters for this clustering approach:

- *Sample Points:* We decided to use a regular grid of initial sample points which is spanned over the whole volume data set. The user can define the total number of grid points (e.g. 64, 125, 216 ...).

- *Kernel Size:* The user can define the size of the mean-shift kernel by specifying a radius value.

Both parameters greatly affect the clustering result (Section 5.1.1). If the grid of initial sample points is not dense enough, it may happen that several maxima will not be found. A very dense grid increases the chance to find all available maxima, but increases the computational effort as well. The kernel size defines how many nearby data points are considered when re-estimating the kernel's mean. It has to be chosen carefully, since a kernel size that is too small will result in slow convergence. On the other hand, a kernel size that is too big might merge several modes. However, it is guaranteed that the final convergence of the mean-shift clustering is independent of the chosen kernel size (Section 2.3.2).

To speed up the calculations, two rules are defined:

- The iteration is stopped and the cluster is discarded if a mean-shift kernel does not cover any data points. This way all initial sample points that are only surrounded by air are eliminated from further calculations.

- The iteration is also stopped if a mean reaches a position that has already been reached by another mean. In this case both iteration paths would lead to the same result; therefore it is only necessary to follow one of the ways completely.

Although computationally still intensive, the mean-shift algorithm performs very fast which enables the user to try out different parameter settings when deciding on a certain clustering. The quality of different clusterings is evaluated visually by the user as described in Section 4.2.1.

k-means++ clustering uses the iterative k-means clustering algorithm together with a more sophisticated way of selecting the initial data samples. For classic k-means clustering the initial data points are selected randomly, whereas k-means++ clustering aims at spreading the initial cluster centers away from each other. The process is briefly described in Section 2.3.1.

Unlike the mean-shift algorithm, k-means++ always takes into account all available data points to adjust the given mean values (Section 2.3.1). The initial means are chosen from the set of data points and then iteratively re-estimated according to the nearby data points. The k-means++ algorithm needs to know in advance how many clusters should be created. Therefore, the user has to define only one parameter: the *number of clusters*. All other computations (selection of initial sample points and iterative mean re-estimation) are executed automatically. Like the mean-shift algorithm, the k-means++ algorithm performs very fast which enables to give immediate feedback to the user.

The user has to compute the clustering for every given object separately. The reason for this is, like for the defect detection, that it may be necessary to adjust parameters according to certain object individualities. Visual feedback is provided to the user to help decide whether the current clustering is appropriate (Section 4.2). At the end a list of regions of high feature concentration (clusters) is created for every object (Figure 3.3).

Every cluster has the following parameters:

- *Mean*: The location of the cluster's centre of gravity, which in fact defines the location of the cluster. It consists of a 3-dimensional vector, defining a voxel position.

- *Size*: It is assumed that all clusters are sphere-shaped; therefore the size defines the radius of the cluster. The size is computed after the iteration stopped and the cluster's final position has been found (Section 2.3.1). Then the cluster size is defined as the distance from the mean to the outmost feature covered by the cluster.

- *Number of features*: The number of features covered by this cluster.

It may happen that clusters overlap in case that their means are close together. Then features will be covered by more than one cluster. Although solutions exist to avoid this [5, 61], we finally decided to allow cluster overlaps. The reason is that further analysis stages do not concentrate on individual features, but on the clusters itself. Therefore, it is important to keep the full cluster information and not to find an optimal division of the underlying data points. However, overlaps have to be considered during exploration (Section 4.2).

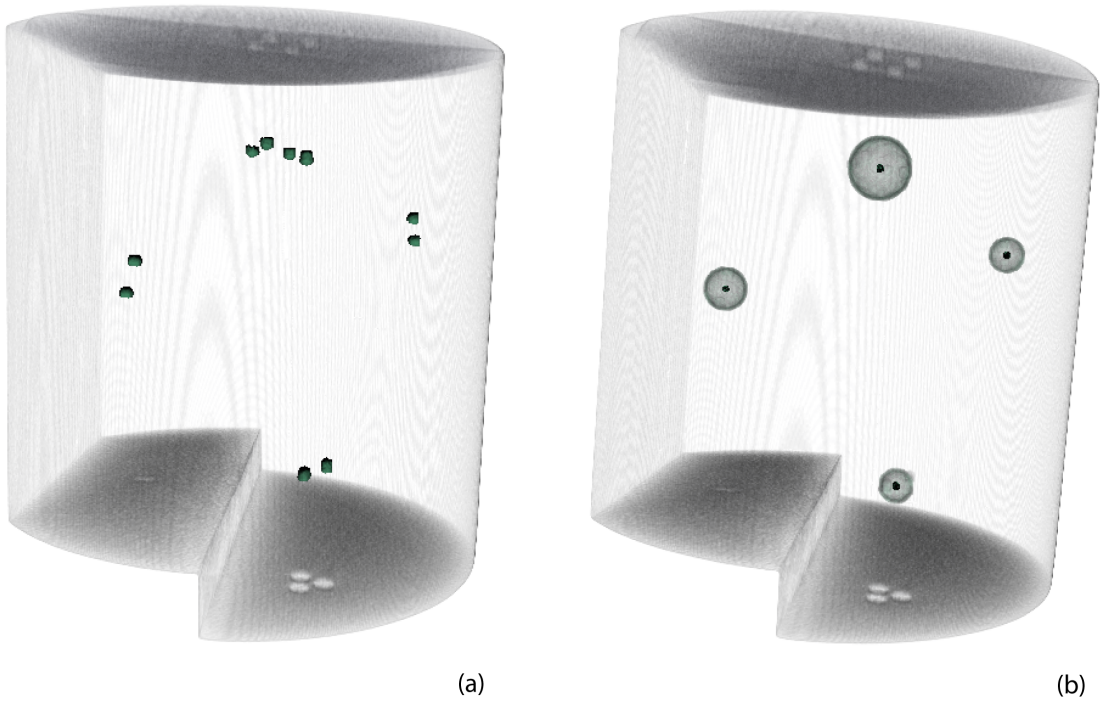


Figure 3.3: *This figure shows how a clustering for the* 11 *selected features in (a) may look like. The clustering has been calculated using the mean-shift algorithm, which has divided the data into* 4 *clusters (b). The location and size of the final clusters are illustrated, whereas their means are painted as dark points. Other examples for clusterings can be found in Figures 3.4 and 3.5.*

Variability Analysis

The region-based variability analysis approach aims at tracking regions of high feature concentration (RHCs) through a test run. RHCs are represented by clusters which have been computed during the clustering stage. The variability analysis now uses this information to answer the question whether RHCs exists that tend to appear in more than one object within a given test run. If this is the case, this means that the manufacturing process has a certain probability to produce defective parts.

The variability analysis merges the cluster information of different objects computed during the clustering step. Cluster merging is done based on their spatial location, whereas a cluster's location is defined by its mean. Since it is required that all given volume data sets are aligned, it can be assumed that clusters at approximately the same spatial position actually refer to the same RHC within the prototype (Figure 3.6). Therefore, such clusters are merged to form one RHC in the further process.

However, due to differences in the feature distribution within the given objects it is very likely that cluster means do not exactly exhibit the same voxel positions in all data sets (although they refer to the same RHC). Therefore, the variability analysis algorithm tolerates small shifts between the cluster means. Two overlapping clusters will be still assumed to represent the same

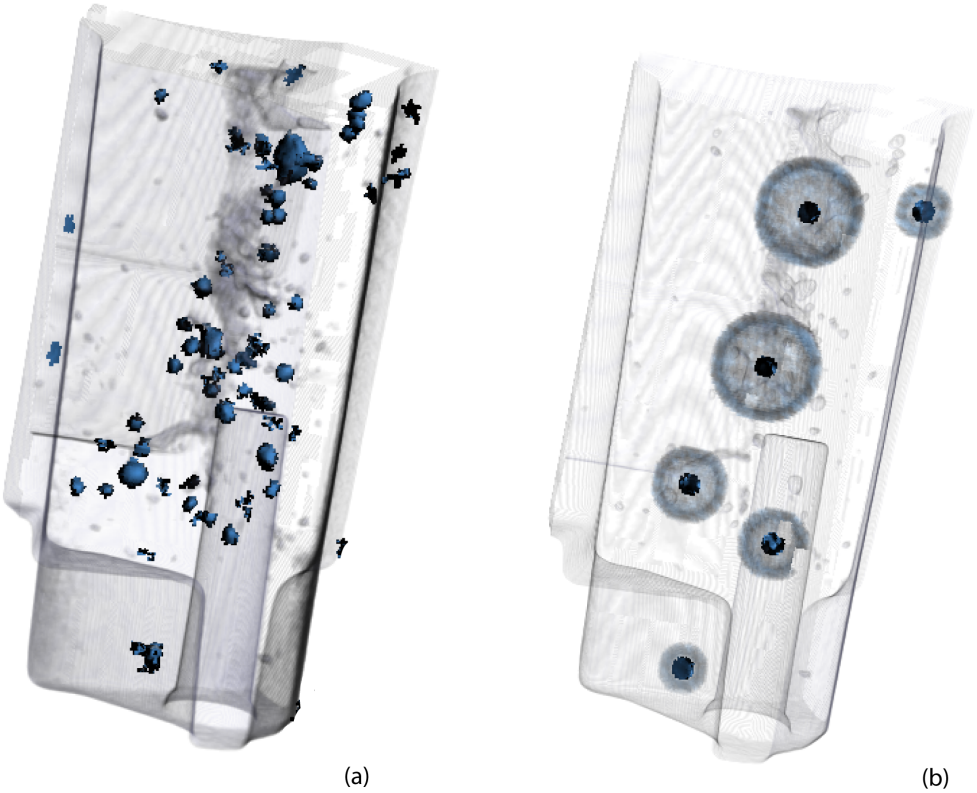


Figure 3.4: *This figure shows how a clustering for the* 259 *selected features in (a) may look like. The clustering has been calculated using the mean-shift algorithm, which has divided the data into* 6 *clusters.*

RHC, if their overlap is great enough. Their means both have to be covered by the other cluster respectively. In summary, two clusters are merged if:

- their means are located at the same voxel position,
- or their means are not located at the same voxel position, but the mean of the first cluster is covered by the second cluster and vice versa.

Using these rules it is possible to start with a set of clusters of one object and find all existing counterparts in all other given objects. This somehow tracks clusters through the given test run (Figure 3.7). The tracking is done for every cluster available within every object O_i to not lose any information. This leads to a final list of clusters available within the prototype P. These final clusters can then be classified by counting the number of objects they appear in. The test run produced for this thesis consists of three objects. Therefore, it was possible to identify a maximum of three different cluster *variability classes*: *1* - cluster appears in one object, *2* - cluster appears in two objects, *3* - cluster appears in all three objects. The higher the variability

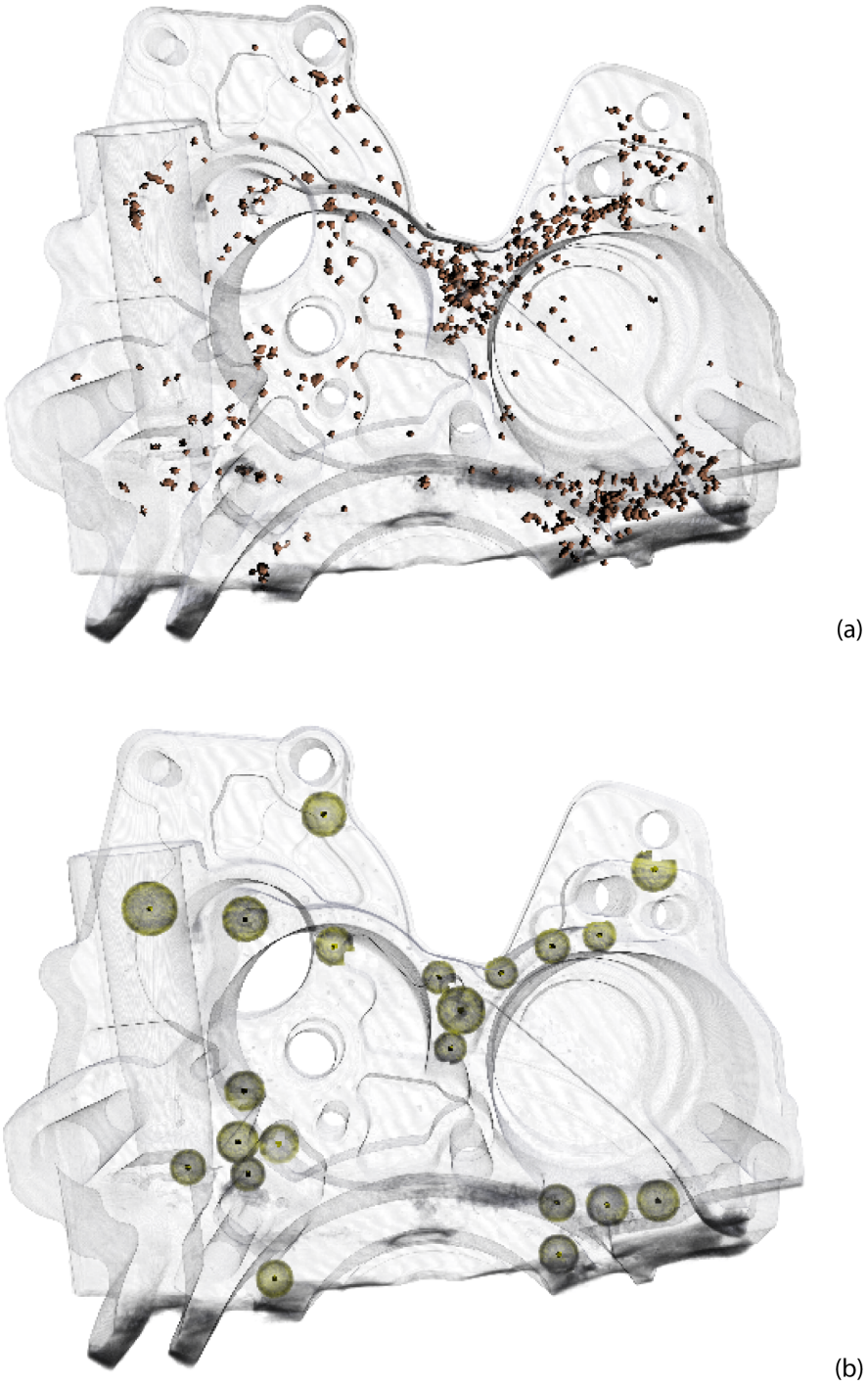


Figure 3.5: *This figure shows how a clustering for the* 2563 *selected features in (a) may look like. The clustering has been calculated using the mean-shift algorithm, which has divided the data into* 21 *clusters. Some of the clusters seem to overlap, but they are actually located successively.*

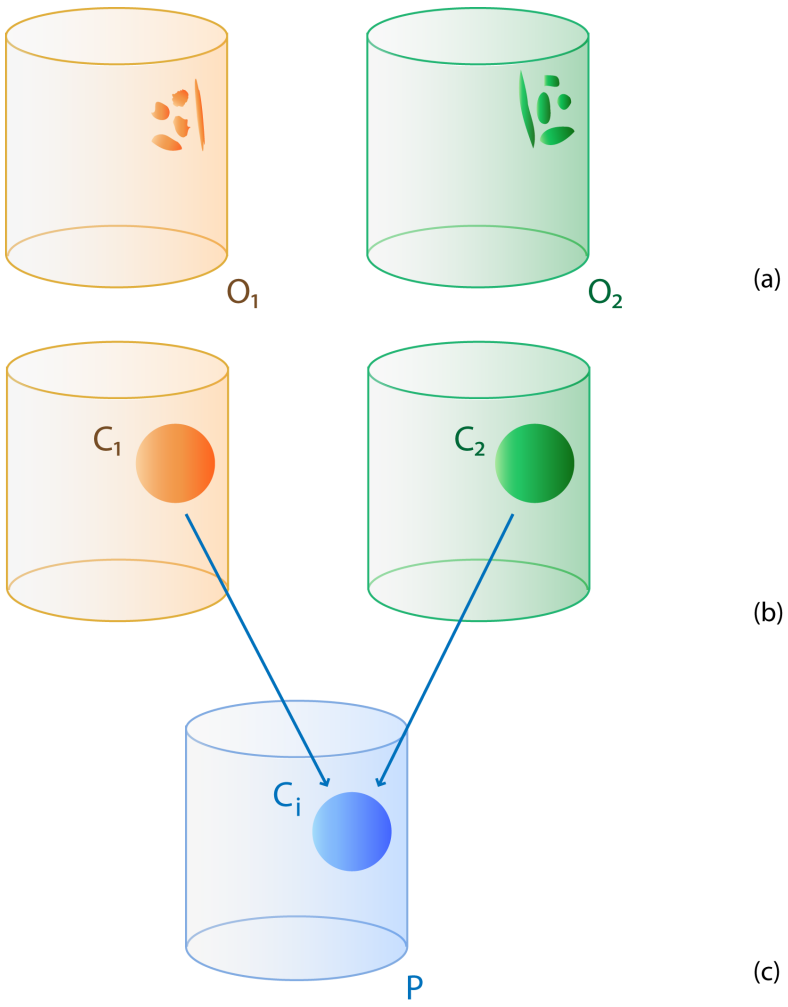


Figure 3.6: *Two objects O_1 and O_2 in (a) contain a set of features that are concentrated in a certain region respectively. After data clustering these regions are represented as two clusters, named C_1 and C_2 in (b). Since the data sets are aligned, the locations of the clusters C_1 and C_2 are located at the same spatial location within the prototype P. Therefore, the clusters C_1 and C_2 can be summarised as the cluster C_i within the prototype P in (c).*

class, the more objects exhibit a cluster, and the more important it is therefore to carefully inspect this region when analysing the manufacturing process.

The parameters of the resulting clusters are computed as the arithmetic average of the cluster values they are connected to. Equation 3.2 shows how the size s_i, mean m_i and number of features f_i are computed for a cluster C_i. The number n represents the number of clusters the current cluster C_i is connected to.

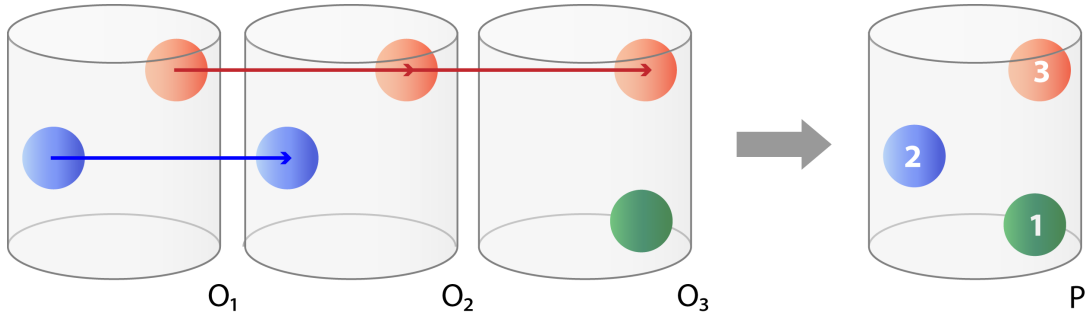


Figure 3.7: *Three objects O_{1-3} contain a set of clusters respectively. For the top cluster (painted in red) it is possible to track it over the whole test run since it can be found in all objects. The middle cluster (painted in blue) can only be found in the objects O_1 and O_2. The third cluster (painted in green) is only available in the last object. This leads to a classification of the clusters within the prototype P in three classes (*1,2 *and* 3*) based on their availability within the test run.*

$$s_i = \frac{1}{n}\sum_{c=0}^{n-1} s_c, m_i = \frac{1}{n}\sum_{c=0}^{n-1} m_c, f_i = \frac{1}{n}\sum_{c=0}^{n-1} f_c \tag{3.2}$$

During the merging phase all clusters in all objects are inspected to see if cluster pairs in other objects can be found. If this is the case, the cluster counterparts are marked to avoid multiple assignments. Algorithm 3.1 gives a description of the algorithm in pseudo code. There it can be seen that the analysis is started for every cluster within all objects of the input set (line 1 and 2). The function *ClusterList(object)* returns all clusters that belong to a certain object. Only clusters that have not already been merged are taken into account (which is checked by the function *State(cluster, state)*. Then the function *EqualPositions(cluster, cluster)* tests whether the positions of two clusters are equal (line 6), where small shifts between the means are allowed. If a new cluster l is found that belongs to the same region as the current cluster c, it will be added to the merge list of c (line 7). Then c's size is updated (line 8), and l's state is set to $merged$ (line 9).

Applying the variability analysis described in this Section to a test run leads to a list of clusters for a certain prototype that can be classified by their variability parameter v. v is defined by the number of objects the cluster appears in (i.e. how many times it could be merged with another cluster from another object). Furthermore, each cluster is sphere-shaped and has a position (mean) m, a size s and a number of corresponding features f. This information can then be used to enable the visual exploration for the user (see Section 4.3.1).

3.3.2 Feature-based Variability Analysis

Another type of analysis approach described in this thesis is the feature-based variability analysis. Foundry engineers are not only interested in the fact whether regions of high feature concentration appear in more than one object, but also in the fact whether large features can be

Algorithm 3.1: Region-based variability analysis

input : set of objects S_O
output: list of clusters P_C for a prototype P

```
1  v = 0;
2  for object o in S_O do
3      for cluster c in ClusterList(o) do
4          if State(c, 'merged') == false then
5              for object p in S_O and o != p do
6                  for cluster l in ClusterList(p) do
7                      if EqualPositions(c, l) == true and State(l, 'merged') == false then
8                          Merge(c, l);
9                          CalculateNewSize(c, l);
10                         SetState(l, 'merged');
11                     end
12                 end
13             end
14         end
15     end
16 end
```

tracked in the test run. In the case where large defects appear in some of the objects, it is important to know whether these are just outliers or whether the casting material tends to produce holes or does not completely fill the mould in certain parts. The feature-based variability analysis approach directly uses the defect detection information without computing regions of high feature concentration first. Unlike clusters, features do not exhibit geometric parameters like a centre (mean) or a size value, but they are rather defined by a set of connected voxels (Section 2.1.2). Therefore, when trying to spot overlaps between features of different objects, the analysis approach needs to operate on voxel-based data.

The feature-based variability analysis starts by creating a temporary variability volume V. This volume exhibits the same dimensions as the objects of the test run. Since the objects are aligned, the position of a voxel v_i within an object O_j is the same as the position of a voxel v_i within V. Based on these assumptions, a voxel-based approach can be used to assign the voxels of V a *variability class*. A voxel variability class defines in how many objects a voxel is marked as a feature voxel. To get this information, counters are assigned to every voxel in V. Before starting the analysis, all counters are initialised by being set to zero. During the analysis all objects are scanned, and the counters in V are increased every time object voxels can be found that are marked as feature voxels (Figure 3.8). A high counter at the end means that the voxel v_i within V is a feature voxel that appears in more than one object.

Algorithm 3.2 gives a description of the algorithm in pseudo code. Before starting the analysis, the voxels in volume V have to be initialised (*Initialise(volume)*), which means that their

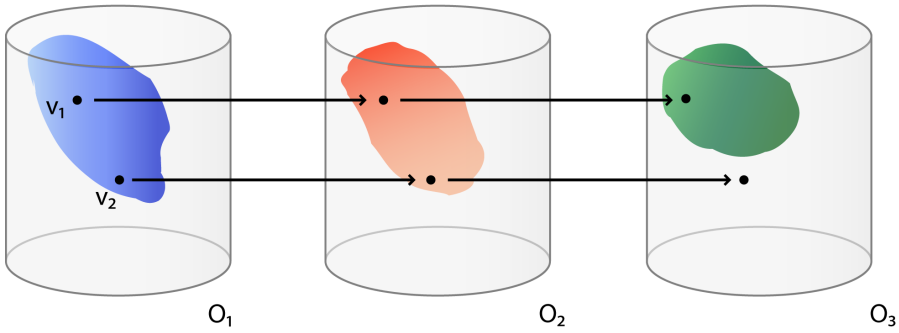


Figure 3.8: *Feature-based variability analysis. This analysis approach directly uses the defect detection information without computing regions of high feature concentration first. For every voxel position v_i it is counted how many times it represents a feature voxel in the objects O_j. In this figure the coloured regions mark large features in the objects O_{1-3}. The voxel position v_1 represents a feature voxel in all three objects, whereas the voxel position v_2 is only covered by a feature in object O_1 and O_2 (not in object O_3).*

counters are set to zero (line 1). Then for every object o the list of corresponding features (previously calculated by the defect detection) is allocated by the function *Features(object)*. Every feature f consists of a set of connected feature voxels (line 3), which can be allocated by the function *FeatureVoxels(feature)*. For every feature voxel v, the counter of the corresponding voxel in V is increased by one (line 4) by using the function *IncreaseCounter(volume, voxel)*.

Algorithm 3.2: Feature-based variability analysis

input : set of objects S_O
output: counter c assigned to every voxel v in variability volume V

```
1 Initialise(V);
2 for object o in S_O do
3 |   for feature f in Features(o) do
4 |   |   for voxel v in FeatureVoxels(f) do
5 |   |   |   IncreaseCounter(V, v);
6 |   |   end
7 |   end
8 end
```

At the end, the counters of all voxels in the variability volume V have been set to a certain value c, where $0 \leq c \leq j$ and j represents the number of objects. A counter value of zero means that no feature voxel can be found at this place in any of the objects. A counter value of one means that features can only be found in one of the objects and therefore they are very likely to be classified as outliers. Voxels with a high counter value show areas of overlapping features of

different objects and will therefore be of high interest. The information stored in V is used to build the data sets needed for exploration (Section 4.3.2).

3.4 Results

After applying the variability analysis computations to a certain test run, the results need to be visualised in a way that they can be explored by the user. This involves the creation of two main data sets: the *prototype volume* and the *variability volume*. The prototype volume is a digital representation of the prototype that builds the basis for the exploration. The variability volume stores the information computed during the analysis stage in a per-voxel format.

3.4.1 Prototype Volume

Since the approach described in this thesis aims at analysing a mould's suitability for serial production, the user's interest concentrates on the mould's cavity. Its shape represents the work piece that should later be produced. However, the prototype exists only as a production plan at this stage and no digital representation of it is available. To visualise it, the prototype's shape has to be gained from the shapes of the available objects within the test run.

A possibility to obtain a digital prototype representation would be to simply select one of the objects and make it become the prototype's representative. However, this may lead to an incorrect solution, since objects may exhibit failures (e.g. misruns) that cause differences from the prototype's outer dimensions. If the wrong object was chosen, the prototype's dimensions will not be displayed correctly. Therefore, it has been decided to reconstruct the prototype based on all available objects instead. The average density volume of all input data sets is computed and assumed to represent the prototype. By averaging the data, object individualities (i.e. small defects or misruns) are filtered out and the prototype's size is reconstructed considering the size of all available objects. The new data set exhibits the same size and resolution as the objects' ones. This new *prototype volume* builds the basis for all further exploration steps (Section 4.1).

3.4.2 Variability Analysis Volume

The variability analysis information computed during the analysis stage has to be prepared in a way that it can be used during the exploration stage (Chapter 4). The most important parameter gained through the analysis is the *variability class* of certain objects parts. Based on the type of variability analysis, variability classes are either assigned to clusters or to feature voxels. The exploration for both analysis types is predominantly based on this parameter.

It has been decided to store it as a a per-voxel information, regardless of the variability analysis type chosen. Since the feature-based analysis is already based on voxel-based information, the results can be directly transferred into the variability volume. In case the region-based approach has been chosen, the resulting clusters are defined by all voxels that are inside them. For both analysis types, a 3D *variability volume* is used to store the variability analysis results. This volume is later used during exploration to build a 3D texture out of it (Chapter 4). The only difference between a region-based variability volume and a feature-based variability volume is the amount of information stored per voxel.

Region-based approach

The region-based variability analysis results in a list of clusters that exhibit the following parameters respectively: *mean*, *size*, *number of features* and *variability class*. Now, for preparing the variability volume, the clusters have to be represented by the set of voxels they cover. When preparing the variability volume, the list of clusters is scanned and every voxel covered by a cluster is assigned the corresponding information. At the end, the following data is stored for every voxel:

- *Variability class*: The variability class the cluster belongs to. It will be set to 0 if no variability information is available - this means that the voxel is not covered by any cluster.
- *Radius*: This value is computed as the distance between the cluster mean and the current voxel's position.
- *Size*: The size of the cluster the voxel belongs to.
- *Control parameter*: This control parameter is set to 0 if the current voxel does not belong to a cluster, and set to 1 otherwise. This parameter currently reflects the same information as the *variability class*, but it has been introduced in case it might be needed for future work (Chapter 6).

The variability class information is the most important one since it defines the visibility and colour of this voxel. Its value is consistent for all voxels covered by a cluster. The information about the radius and the cluster's size are needed to reconstruct the voxel's position within the cluster. This is on the one hand needed to adapt the voxel's opacity according to its distance from the cluster mean, and on the other hand it allows hiding cluster parts by specifying the transfer function accordingly (Section 4.3.1).

Feature-based approach

The feature-based variability analysis results consist of a 3D volume where a variability class is assigned to every voxel. Since this volume is of the same size as the prototype, the variability volume's voxel information is directly used to build the variability volume. The voxels' counter values represent the variability classes. This values range between 0 and i, where i is the number of available objects. A counter value of 0 means that no feature voxel can be found at this position in any of the objects, therefore no variability information is available at this point. In fact the counter values can be simply copied from the feature-based variability analysis results to form the variability volume. This is possible since the data set used for this thesis consists of only three objects. It is possible to handle even more objects, but then it might be necessary to include additional calculations to limit the maximum number of variability classes (Section 6.2.5). In the case of three objects, this results in a maximum number of three variability classes, which is optimal for exploration. If more variability classes are present, it might be necessary to adapt the structure of the 2D histogram (Section 4.3.1) to preserve clearness and usefulness (Section 6.2.5).

CHAPTER 4

Exploration

The exploration stage is the most visible process step for the user. The provided exploration techniques allow the foundry engineer to visually explore the results of the analysis. Within the variability analysis pipeline described in this thesis, two different stages can be found where exploration becomes important: At first visual exploration techniques are used during the region-based variability analysis to decide on whether a current clustering is accurate. Additionally, exploration is used to display the final results of the variability analysis. Both exploration tasks are based on the computations of the previous analysis stages. However, the complexities from the pre-computation steps are hidden to a large extent.

All explorations are embedded into a 3D volume rendering of a volume data set, which is either a casting object or the casting prototype (Section 3.4.1). Using a 2D histogram and user-defined widgets it is possible to select parts within the volume that should be displayed. This way the foundry engineer can select regions of interest and assign them a certain colour.

The first Section (4.1) describes the rendering of 3D volume data sets, which forms the basis for further exploration techniques. The two types of exploration techniques (cluster and results visualisation) are described in Section 4.2 and 4.3.

4.1 Volume Rendering

The high number of data points produced by industrial CT scanners greatly improves the precision and detail of the resulting models, but also increases the size of the volumetric datasets. Furthermore, work pieces used in foundry trials can be very large (e.g. engine parts), which extends the amount of available data even more. Today's graphics hardware offers a large amount of on board memory which allows ray casting algorithms to run on the graphics processing unit (GPU) and achieve real-time frame rates (Section 2.2.1). However, the size of texture memory is still limited. In most cases the rendering of single volume data sets can still be handled. However, the limited memory leads to a problem when trying to include variability analysis specific exploration. Due to the use of a 3D variability class texture (Section 3.4.2), the texture memory

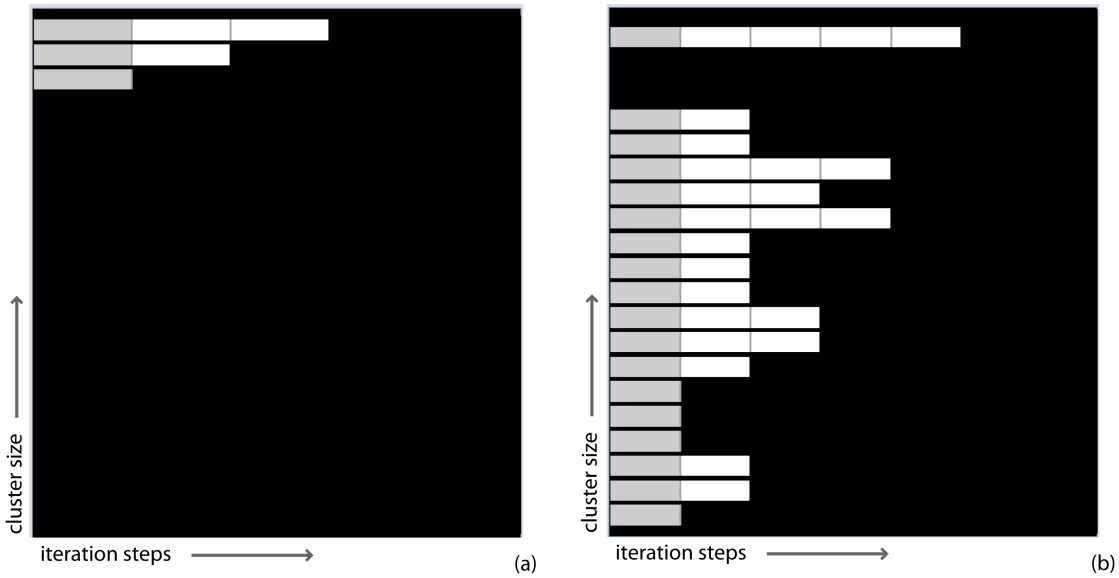


Figure 4.1: *Clustering results visualised by a bar chart. Every cluster is represented by a horizontal bar, which are sorted along the y-axis according to their size. The length of the bars indicates how many iterations it took to find the final position of the corresponding cluster (the first grey part depicts the initial cluster position). A well-defined clustering like in (a) indicates that is has been applied to a well-structured set of data points.* 3 *clusters have been generated in up to* 2 *iteration steps. A more varying result like in (b) is produced when applying the clustering to scattered data points. There it took up to* 4 *iteration steps to find the final positions of all* 18 *clusters. For the user this block chart visualises the number of clusters and the distribution of cluster sizes. Furthermore, this histogram can be used to paint features in colour according to their cluster membership (Figure 4.2).*

allocated is twice the size of the volume data. Even modern GPUs may not be able to handle such memory requirements, especially if the digitised work pieces are of a very large size.

The *high-quality hardware volume renderer* (HVR) used in this thesis is a hardware-based rendering approach which includes several performance optimizations [84]. One of them is *empty space skipping* which represents a solution to handle the memory requirements of large volumetric datasets during rendering. This approach first subdivides the volume into small blocks containing a fixed number of voxels. Then only non-empty blocks with opacity greater than zero become active and form the data-dependent bounding geometry. Since in industrial CT parts a significant number of voxels are usually part of the background (i.e. air), this approach skips unneeded data. In the next step, *culling* determines a list of active bricks (with opacity greater than zero) that are packed into a single 3D brick cache texture, which is loaded onto the GPU. Further information can be found in the work by Laura Fritz [29]. The actual rendering of the volume data set is done by ray-casting in the fragment shader.

4.2 Feature Clustering

When using region-based variability analysis, regions of high feature concentration are obtained by applying a clustering approach to the data (Section 3.3.1). Finding an optimal clustering solution is NP-hard (Section 2.3.3), which makes it impossible to provide an ideal result automatically. Therefore, the user applying the clustering algorithm has to find out whether the current solution is appropriate, or whether the clustering parameters need to be adjusted. In this approach, visual feedback is provided to help the user make a decision. The general idea of this visualisation is to paint features in a user-defined colour based on their cluster membership. Furthermore, the user can reconstruct the cluster creation on a time line. However, the automatic evaluation of clustering results is a subject for future work (Section 6.2.1).

4.2.1 Evaluation of Clustering Results

Clusters can be explored by specifying a transfer function in the 2D domain of *cluster_id* and *iteration_step*. The transfer function is specified by user-defined 2D solid widgets as described by Kniss et al. [55]. The clustering results are displayed in a bar chart view in the transfer function panel (Figure 4.1). Every cluster is represented by a horizontal bar, which are sorted along the y-axis according to the clusters' size. The size of a cluster is given by the number of features it covers. The x-axis of the chart is divided into smaller parts of equal length which represent the iteration steps that were necessary to build the clustering. The longer a certain bar, the more iteration steps it took to find the final position of the corresponding cluster.

The clustering process can be reconstructed by dragging a widget over different block segments, since this will highlight all features covered by the corresponding cluster in the selected iteration step (Figure 4.2). For the user this visualisation is helpful to reconstruct and understand how clusters are created. This can help to answer why certain regions inside the volume will never be covered by a cluster. The last block segments (i.e. the last iteration steps) represent the final positions of the corresponding clusters respectively. Positioning widgets over these segments will display the final results of the clustering by painting features in colours according to their cluster membership (Figure 4.3).

The cluster exploration helps the user decide on the quality of the current clustering result. For example, if the kernel size for mean-shift clustering was set to a too big value, this will result in one big cluster containing all features. On the other hand, if the kernel size was too small, too many clusters containing only a very small number of features would be created. Although mathematically correct, such clusterings will not produce appropriate results when used for further variability analysis computations (Figure 4.3).

Due to the further computations within the variability analysis it had been decided to allow overlaps between final clusters (Section 3.3.1). It may happen that features are covered by more than one cluster, which affects the results of the exploration. Features are painted according to the widget's colour that has been placed at last, so it may happen that the colour of another widget is overwritten. This has to be considered by the user when exploring overlapping clusters. See also future work addressing this topic (Section 6.2.2).

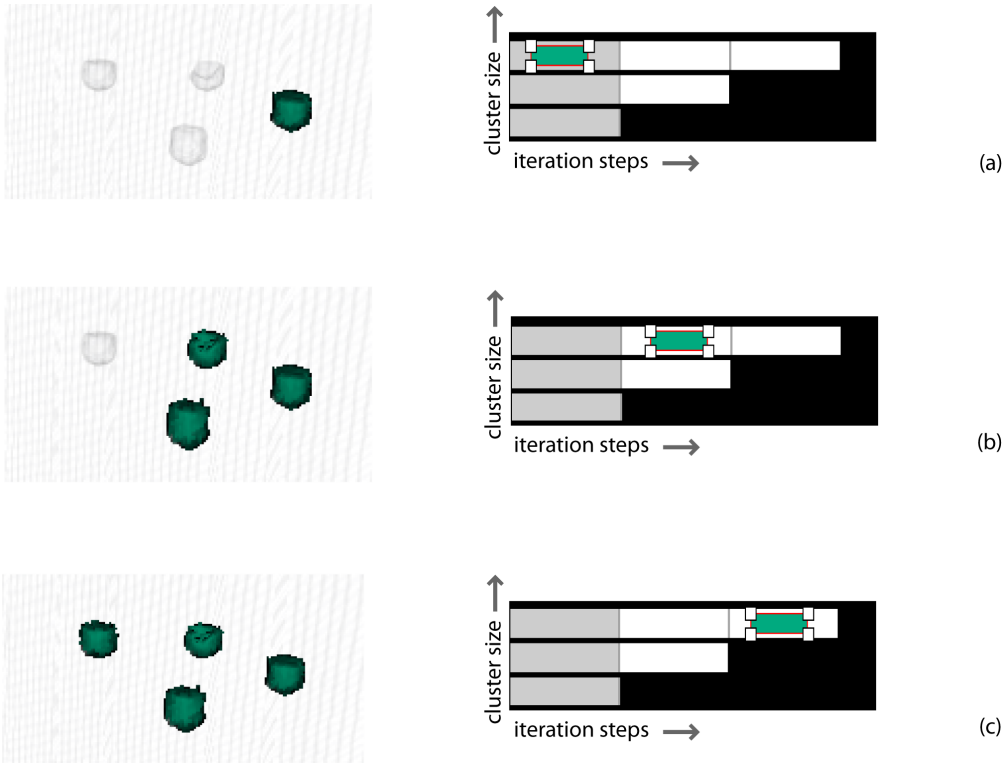
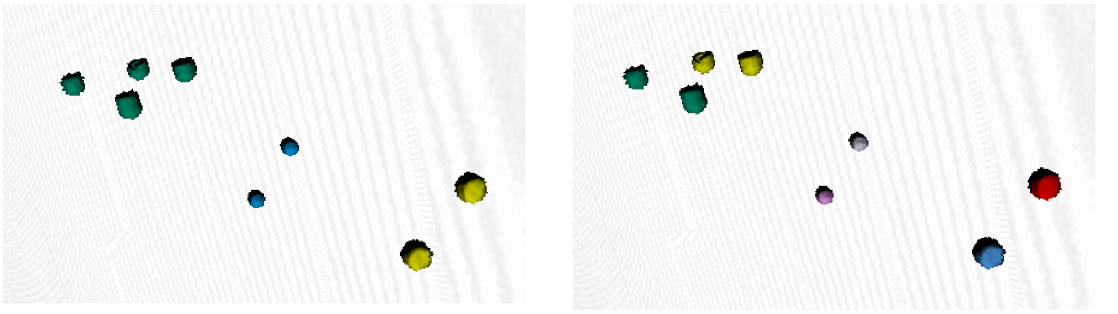


Figure 4.2: *This figure shows how the user can drag a widget over a histogram in the dimension of cluster size and iteration steps to visualise cluster creation process. In the first step, which in fact is the initial cluster position, only one feature is covered by the cluster (a). Based on this data, the cluster's mean is shifted, which makes it cover already three features after the first iteration (b). The mean is shifted again until the cluster covers four features at the end (c).*

4.2.2 Feature Colour Lookup

The visualisation is based on a 2D transfer function in the domain of *cluster_id* and *iteration_step* which is specified by user-defined widgets. For every widget the list of features covered by it can be estimated from the underlying bar chart values. For every bar segments it is known which features are covered by the corresponding cluster in the given iteration step. Additionally, every widget exhibits a user-defined colour. This leads to a creation of a colour lookup table, i.e. a 1D 16-bit texture in OpenGL, mapping feature ids to a certain colour. If no user-defined colour can be found for a feature, its voxels are coloured using the default 1D density texture value. The actual feature id of a voxel is retrieved with a lookup in the feature growth table as described in [29]. The pseudo-code (similar to GLSL) in Algorithm 4.1 illustrates the main steps that need to be done in order to determine the colour and opacity (without shading) of a sample as RGBA. The evaluation of this scheme is done within a GPU fragment shader program during ray-casting.



(a) (b)

Figure 4.3: *Example for validating clustering results. All features covered by a cluster are painted in the corresponding user-defined cluster colour. In (a) an appropriate clustering solution for the given object part is shown, which can be used for further analysis computations. In (b) an example for a non-appropriate clustering is shown. There clusters are too small (they only contain at most two features) to reflect the underlying feature distribution. This results from the fact that the number of clusters has been set too high for the k-means algorithm. In this case, clustering parameters still need to be adjusted.*

Algorithm 4.1: Feature colour lookup

```
1  float density = texture3D(density_volume, sample_coord3);
2  vec2 feat_vox = texture3D(feature_volume, sample_coord3);
3  float birthID = feat_vox.x;
4  float birthTime = feat_vox.y;
5  if (birthID == ID_NONE || birthTime > T_CUR) then
6  |   out = texture1D(tf1D, density);
7  else
8  |   vec2 fsmap = texture2D(growth2D, vec2(T_CUR, birthID));
9  |   float curID = fsmap.y;
10 |   out = texture1D(colour1D, curID);
11 |   if (out.a == 0.0) then
12 |   |   out = texture1D(tf1D, density);
13 |   end
14 end
```

4.3 Variability Analysis Results

The visual representation of the results is the basis for evaluating the results (Chapter 5). The predominant exploration parameter is the *variability class* (Section 3.4.2). According to the variability analysis type used, it is possible to specify the visibility and colour of either clusters or feature voxels respectively. 2D solid widgets as described by Kniss et al. [55] are used to explore the variability classes' space.

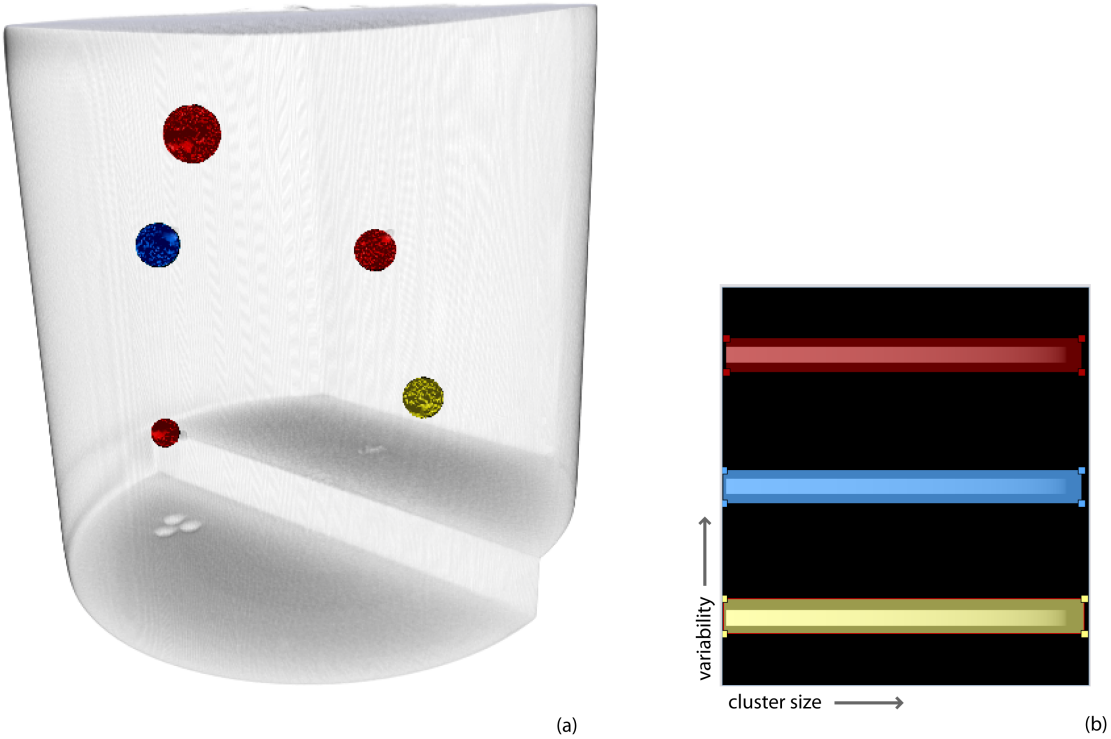


Figure 4.4: *This figure shows the results of the region-based variability analysis. They were created by using the test data set which was prepared for implementing this thesis (Section 3.1) which consists of three objects. A total of three variability classes was created, which can be seen in the block chart in (b). Using this block chart as a 2D transfer function the resulting clusters can be colourized (a). Two regions (painted in red) occur in all available objects, one region (painted in blue) appears in two objects and one region (painted in yellow) appears in one object only.*

All exploration techniques are based on the visualisation of the *prototype volume* (Section 3.4.1). Additionally, the *variability volume* is used to store the variability class of all voxels in the prototype volume. During rendering, a single texture fetch from this 3D *variability texture* yields everything needed to reconstruct the voxel's variability class.

4.3.1 Results of Region-Based Analysis

The region-based variability analysis results in a list of clusters with a certain variability class, position and size. Each cluster represents a region of high feature concentration within the given prototype. Additionally, the variability class defines if a region tends to appear in more than one object within the given test run (Section 3.3.1). Therefore, the visualisation parameters are the *region's location*, the *region's size* and its *variability class*. During exploration, the cluster's visibility and colour can be specified by a user-defined transfer function.

Figure 4.5: *The size of the explored clusters is adjustable via the length of the widgets. The complete clusters are displayed in (a), whereas only the inner homogeneous parts around the mean are shown in (b). This exploration technique allows gaining a better overview over the cluster distribution as well as exploring the full cluster dimensions.*

Cluster Exploration

The clusters' variability classes form the predominant visualisation parameters. The visibility and colour of cluster groups is selected based on their corresponding variability class instead of other cluster parameters (e.g. *id* or *size*). The visualisation parameters specified for a certain variability class are applied to all voxels that exhibit this class. The basis for exploration is a block chart in the two dimensions of *variability* and *cluster size*. Horizontal blocks represent a variability class respectively, whereas the most important class is displayed on top of the y-axis (Figure 4.4). The importance of a variability class is defined by the number of objects it represents (i.e. regions that appear in more than one object are considered to be more important). The length of the bars on the x-axis depict the size of the clusters exhibiting the corresponding variability class. Therefore, the visible clusters can be defined by placing widgets over the blocks representing the variability classes of interest. Additionally, the size of the explored clusters is adjustable by varying the length of the widgets (Figure 4.5). The exploration is the basis for evaluating the results (Section 5.1). It visually prepares the most important information (i.e. clusters' variability class, location and size) and allows the user to interactively explore the results. Regions of high feature concentration have been defined and are displayed to the user based on the number of objects they appear in.

Transfer Function

The visualisation is realised through a 2D transfer function approach, wherein the rendering of the prototype volume is defined by a 1D density transfer function. The cluster exploration is based on the values stored in the *variability volume* (Section 3.4.2). The purpose of the variability volume is to store the per-voxel information needed during exploration. This information is transferred to a 16-bit two-channel 3D texture, e.g., a luminance-alpha texture in OpenGL [85]. The per-voxel information stored consists of the *variability class*, *radius*, *cluster size* and *control parameter* (Section 3.4.2). A single texture fetch from this 3D texture yields everything needed to reconstruct the voxel's variability class and position within the cluster.

The user-defined selection of widgets specifies visibility, colour and size of the explored clusters. The user-defined information is transferred into two 1D lookup tables. The first one, the *variability colour table*, maps variability classes to the corresponding widget colour. The second lookup table, the *size table*, maps the widget's length to a certain opacity value. Therefore, the visibility and colour of cluster voxels can be specified by two texture lookups.

Algorithm 4.2: Cluster Exploration

input: float scaling

```
1  vec4 variability_params = texture3D(variability_texture, samplepos);
2  vec4 radius = texture1D(radius_texture, variability_params.r);
3  float sample_radius = variability_params.r * scaling;
4  float min_radius = radius.r;
5  float max_radius = radius.g;
6  if (sample_radius > min_radius && sample_radius < max_radius) then
7  |   out = texture1D(variability_color_texture, variability_params.r);
8  else
9  |   out = texture1D(tf1D, density);
10 end
```

Algorithm 4.2 illustrates the main steps that need to be done in order to determine the colour and opacity (without shading) of a sample as RGBA. The evaluation of this scheme is done within a GPU fragment shader program during ray-casting. In the first two lines of the algorithm all information necessary to decide on the visibility and colour of the current sample is obtained by two texture lookups. The 3D texture *variability_texture* stores the variability class (*variability_params.r*), the radius (*variability_params.g*), the total cluster size (*variability_params.b*) and a control parameter (*variability_params.a*) of the current sample (Section 3.4.2). The 1D texture *radius_texture* stores the user-defined selection which parts of the clusters are currently visible per variability class. The interval are given by a minimum (*radius.r*) and a maximum (*radius.g*) value. In line 3 to 5 the current sample's radius within the cluster (*sample_radius*) is determined which is needed to decide on the visibility of this parts (Figure 4.5). Based on this information, the sample is either coloured with the user-defined colour (line 7), or it is assigned the colour specified by the 1D density transfer function (line 8).

4.3.2 Results of Feature-Based Analysis

The feature-based variability analysis results in per-voxel counters for every voxel inside the prototype, which denote the voxels' occurrence as feature voxels within a certain number of castings. The only visualisation parameter is the *variability* at every voxel position. During exploration, the colour of certain sets of voxels can be specified by a user-defined transfer function.

Feature Voxel Exploration

The voxels' variability class forms the predominant visualisation parameter. User-defined colours can be specified for certain variability classes, which are then applied to all voxels that exhibit these classes. Voxels with a variability class of 0 are not taken into account for the visualisation, since no variability information is present at these positions.

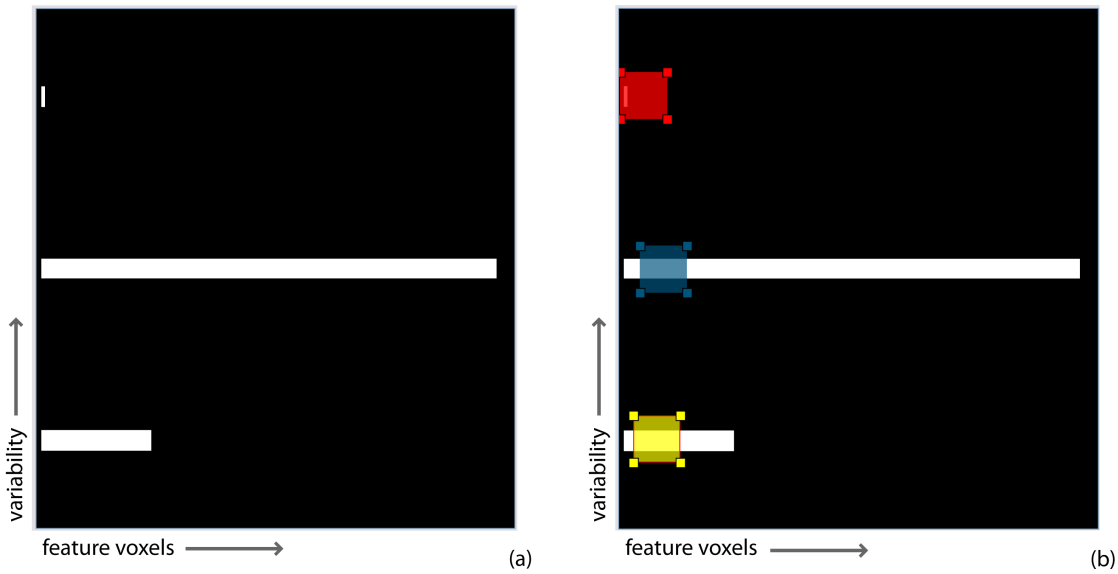


Figure 4.6: *Visualisation of feature-based analysis results. A 2D block chart in the dimensions of* variability *and* number of voxels *is the basis for specifying the transfer function parameters. (a) In this example three variability classes are available, and most of the feature voxels exhibit the middle variability class (*2*). The visibility and colour of feature voxels can be defined by the user by placing widgets over the blocks like in (b). The corresponding feature voxels are painted in the user-defined colour (Figure 4.7). Since the underlying data is 1-dimensional (variability class), the transfer function is defined by the y-position of a certain widget, since this indicates the corresponding variability class. The widget's length as well as its position on the x-axis are not used to control the visualisation.*

The basis for exploration is a 2D block chart in the dimensions of *variability* and *number of voxels*. Horizontal blocks represent a variability class respectively, sorted by their relevance (the most important class is displayed on top). The importance of a variability class is defined by the number of objects it represents (i.e. feature voxels that appear in more than one object are

considered to be more important). The length of the blocks along the x-axis depict the number of feature voxels exhibiting the corresponding variability class. It illustrates how the available feature voxels are split up into the available variability classes. The user specifies colours of feature voxels by placing widgets over the variability class of interest (Figure 4.6).

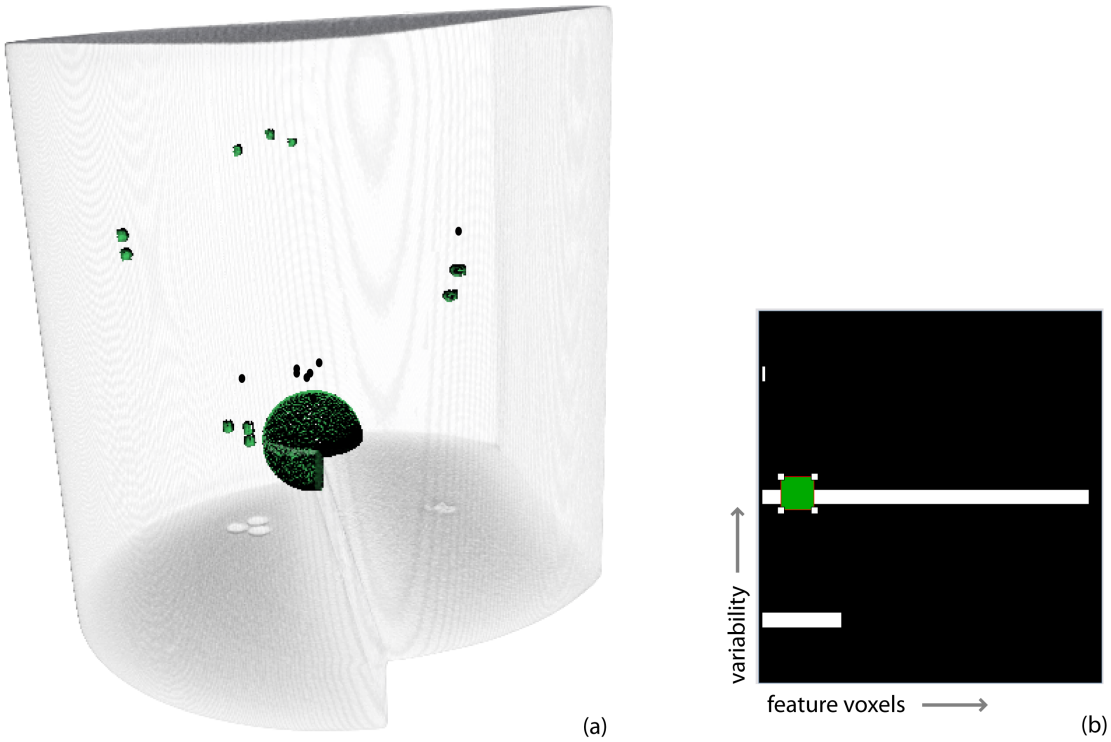


Figure 4.7: *Result of feature-based variability analysis. The results were created by using the test data set which was prepared for implementing this thesis (Section 3.1) to which a hemisphere of feature voxels was added to simulate a hole (Section 4.3.2). Due to the current user-defined selection (b), all feature voxels with a variability of* 2 *are painted in green colour (a). It shows that the hole (as well as voxels of other features) appears in two of the three objects.*

Figure 4.7 shows a typical result of a feature-based variability analysis. It was created using the test data set which was prepared for this thesis (Section 3.1) together with some additional adaptations to simulate a hole. The original data set only contains small features, wherefore additional data was added to create a large feature (the hole). Large features are better suited to be used for the feature-based approach, since they are more likely to produce overlaps within the different objects. This difference can be also seen in Figure 4.7, where small features are hardly recognizable. The user-defined selection used in Figure 4.7 shows all feature voxels that exhibit a variability class of 2 (at a total number of 3 variability classes available). It can be seen that the hole as well as voxels of other features appear in two of the three objects. Furthermore, the initial sizes and shapes of the features is still visible, which is not the case when using the region-based analysis approach.

The main purpose of the feature-based analysis is to check whether large features like holes tend to appear in more than one of test castings. The reason for this is that large features are more likely to produce overlaps with features of other objects than small ones. Since the region-based analysis results are based on the features' centres of gravity and not on their actual shapes and sizes (Section 3.3.1), the feature-based analysis approach is better suited to analyse the extends of large features like holes. Furthermore, the exploration results can be used to analyse the *feature distribution* of the prototype (Section 5.2).

Transfer function

The variability visualisation is realised through a 1D transfer function approach with an additional 1D density transfer function that defines the rendering of the prototype volume. The variability class of all voxels of the prototype volume is stored in the *variability volume* (Section 3.4.2). This information is transferred to a 16-bit two-channel 3D texture, e.g., a luminance-alpha texture in OpenGL [85]. The user-defined selection of widgets specifies visibility and colour of the explored variability classes. The user-defined information is transferred into one 1D lookup table which maps the variability class to a certain colour.

Algorithm 4.3 illustrates the main steps that need to be done in order to determine the colour and opacity (without shading) of a sample as RGBA. The evaluation of this scheme is done within a GPU fragment shader program during ray-casting. In line 1 the variability class for the current sample is fetched from the variability volume (Section 3.4.2) represented by the texture *variability_texture*. In line 2 it is checked whether the current sample exhibits a variability class. If so, the corresponding colour is fetched from the 1D texture *variability_color_texture*. It may happen that the sample exhibits a variability class, but the user has selected to not display voxels of this class. In this case the alpha value of the fetched colour will be zero. Therefore, the sample is assigned the corresponding colour from the 1D density transfer function in this case (line 5). Furthermore, if the sample does not exhibit a variability class, it is also assigned the corresponding value from the texture *tf1D* (line 8).

Algorithm 4.3: Feature Voxel Exploration

```
1 vec4 variability_params = texture3D(variability_texture, samplepos);
2 if (variability_params.a > 0.0) then
3 |   out = texture1D(variability_color_texture, variability_params.r);
4 |   if (out.a == 0.0) then
5 |   |   out = texture1D(tf1D, density);
6 |   end
7 else
8 |   out = texture1D(tf1D, density);
9 end
```

CHAPTER 5

Results

The last stage of the variability analysis pipeline is the interpretation of the results based on the visual representation described in Chapter 4. For the user this is the most important process step, since further steps regarding the mould design process are decided here.

The first two sections of this chapter discuss the results of the variability analysis approaches. Section 5.1 describes the results of the region-based variability analysis and Section 5.2 analyses the final outcome of the feature-based analysis approach. The last Section (5.3) discusses the performance and memory requirements of the described algorithms.

5.1 Region-based Variability Analysis

The region-based variability analysis uses regions of high feature concentration to analyse the variability within a foundry trial. The analysis mainly consists of two parts: feature clustering and variability analysis. Both parts require user input which is assisted by visualisation techniques described in Sections 4.2 and 4.3.1.

5.1.1 Feature Clustering

The cluster visualisation helps the user decide whether a current clustering is appropriate for further calculations. An ideal result cannot be provided automatically since the task of finding an optimal clustering solution is NP-hard (Section 2.3.3). The implemented visualisation paints features in a user-defined colour based on their cluster membership. Furthermore, it is possible to reconstruct the cluster creation on a time line.

For this thesis two different clustering approaches, *mean-shift* and *k-means++*, were implemented to be able to compare the different results. k-means++ clustering uses the iterative k-means clustering algorithm together with a more sophisticated way of selecting the initial data samples, which avoids the sometimes poor clusterings found by the standard k-means algorithm (Section 2.3.1). One important difference between k-means and mean-shift clustering is that

k-means assumes that the number of clusters is already known. This makes this clustering algorithm very sensitive to initialisations. Mean-shift, in contrast, does not assume anything about the number of clusters, but needs some other parameters to be set in advance: the kernel size and the number of initial sample points. The initial parameters, which are defined by the user, may affect the clustering result.

For the k-means algorithm, the only parameter to set is the number of clusters that should be created. This certainly affects the structure of the resulting clustering. Different qualitative results can be created by changing this initial value (Figure 5.1). If the initial value was chosen too small, too many different features will be combined into one cluster. If the number of clusters was chosen too high, the final clusters will contain too few features and will not exhibit any informative value. The k-means algorithm is very sensitive to outliers, because all available data points are considered at every iteration step.

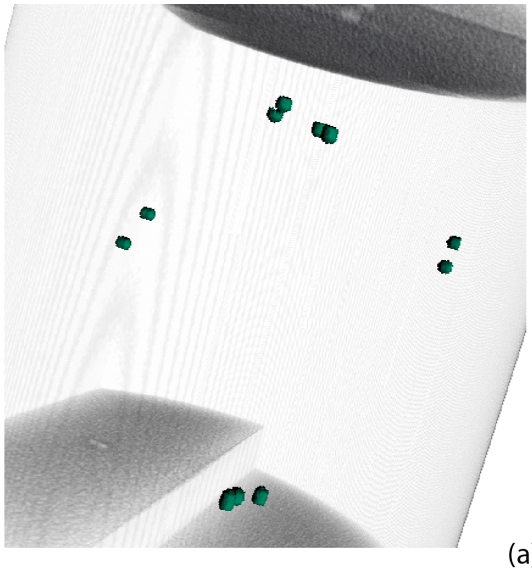
Mean-shift clustering uses kernels of a certain size and shape which are iteratively dragged over the data points until their final position has been found. The user defines the size of the kernels and the number of initial sample points. These parameters may affect the quality of the resulting clustering (Figure 5.2). The kernel size defines how many data points are taken into account during iteration. A too large value may cause clusters to merge, whereas too small values may cause artificial cluster splitting. The number of initial sample points has an effect on the run-time of the algorithm at the one hand and on the resulting clusters on the other hand. If the number of sample points was too high, the computational effort will increase. If too few initial sample points were combined with a too small kernel size, some existing maxima will not be found by the clustering process. In the worst case it all initial clusters will not contain any features, which causes the clustering to fail.

For both clustering algorithms measures have been initiated to improve their performance. Cluster computations are skipped if:

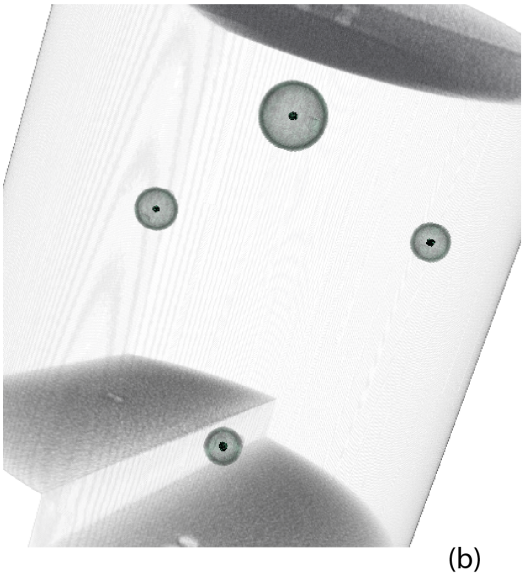
- a cluster is empty, or if
- a cluster reaches a position that has already been touched by another cluster.

This greatly improves the performance even in cases where a high number of initial sample points is used (see also Section 5.3). It has been observed that in a general setting one third of the mean-shift calculations are stopped, because they reach a point that has already been reached by another cluster.

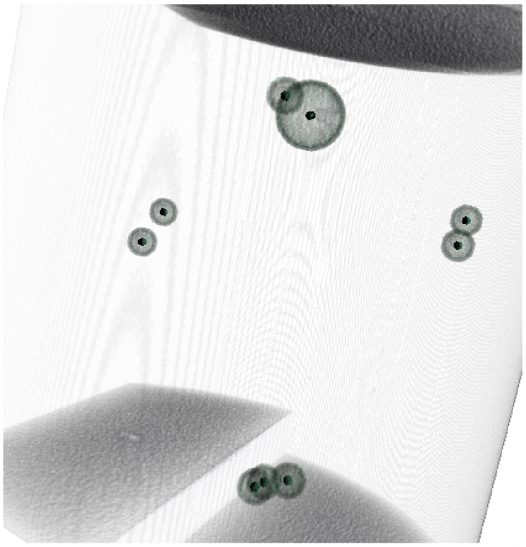
The clustering algorithms have been tested on three different data sets (Figures 3.3, 3.4 and 3.5). Based on these observations it has been decided that the mean-shift clustering algorithm is the most appropriate one for the variability analysis described in this paper. The most important advantage of this approach is that there are no embedded assumptions on the number of final clusters. For the inexperienced user it is difficult to decide on the right number of clusters beforehand by just having a look at the feature data. Furthermore, mean-shift clustering is not as sensitive to outliers as the k-means algorithm.



(a)



(b)



(c)

Figure 5.1: *k-means++ clustering examples. The initial data set consists of a volume data set with* 11 *features (a). Using this data, k-means++ was started twice with two different parameter settings. The resulting clusters and their corresponding means are painted in green. The clustering in (b) was started with an initial value of* 4 *clusters to be created, which results in an appropriate clustering result. In (c) an initial value of* 9 *was chosen, compared to a total number of* 11 *available features. This leads to a bad clustering result where all created clusters (except one) contain just one single feature.*

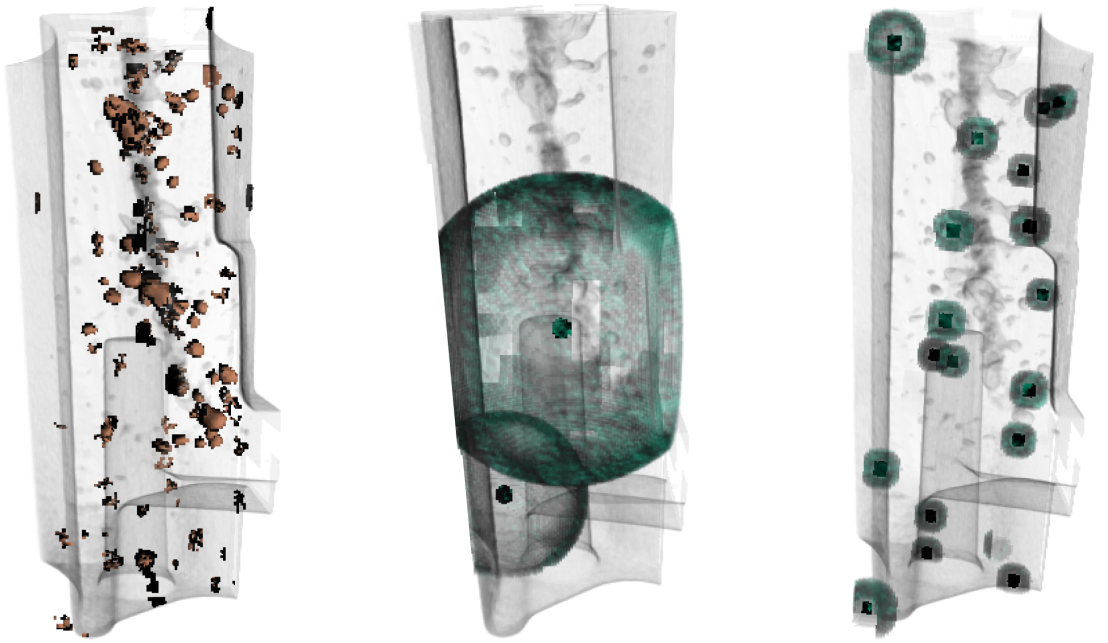


Figure 5.2: *Mean-shift clustering examples. The initial data set consists of a volume data set with several features (a). Using this data, mean-shift clustering was started with two different parameter settings. The resulting clusters and their corresponding means are shown in green. The user-defined value of the clusters' kernel size effects the quality of the resulting clustering. If the kernel size was chosen too large like in (b), only two big clusters were created. The different kernel sizes result from the different underlying feature distribution. In (c) the kernel size was chosen too small and therefore too many clusters were created containing about one feature only. Both results are examples of poor clustering results.*

5.1.2 Variability Analysis Results

The visualisation of the variability analysis results is the basis for the decision on the mould's suitability for serial production. The resulting clusters, which represent regions of high feature concentration, are visualised based on their variability class. As described in Section 4.3.1, the user can select certain variability classes of interest and skip others.

A summary of the complete region-based variability analysis pipeline can be found in Figure 5.3. The input data used for this thesis (Section 3.1) consists of three volume data sets with corresponding sets of features. The features are combined into 4 clusters in every object, and the clusters are then tracked across the complete series of input data sets. The tracking is done based on the clusters' spatial location. The resulting prototype volume then contains 5 clusters, where 3 of them can be found in all input data objects. The most important task for the user is the exploration stage (Chapter 4), where it is possible to explore the results from the analysis computations.

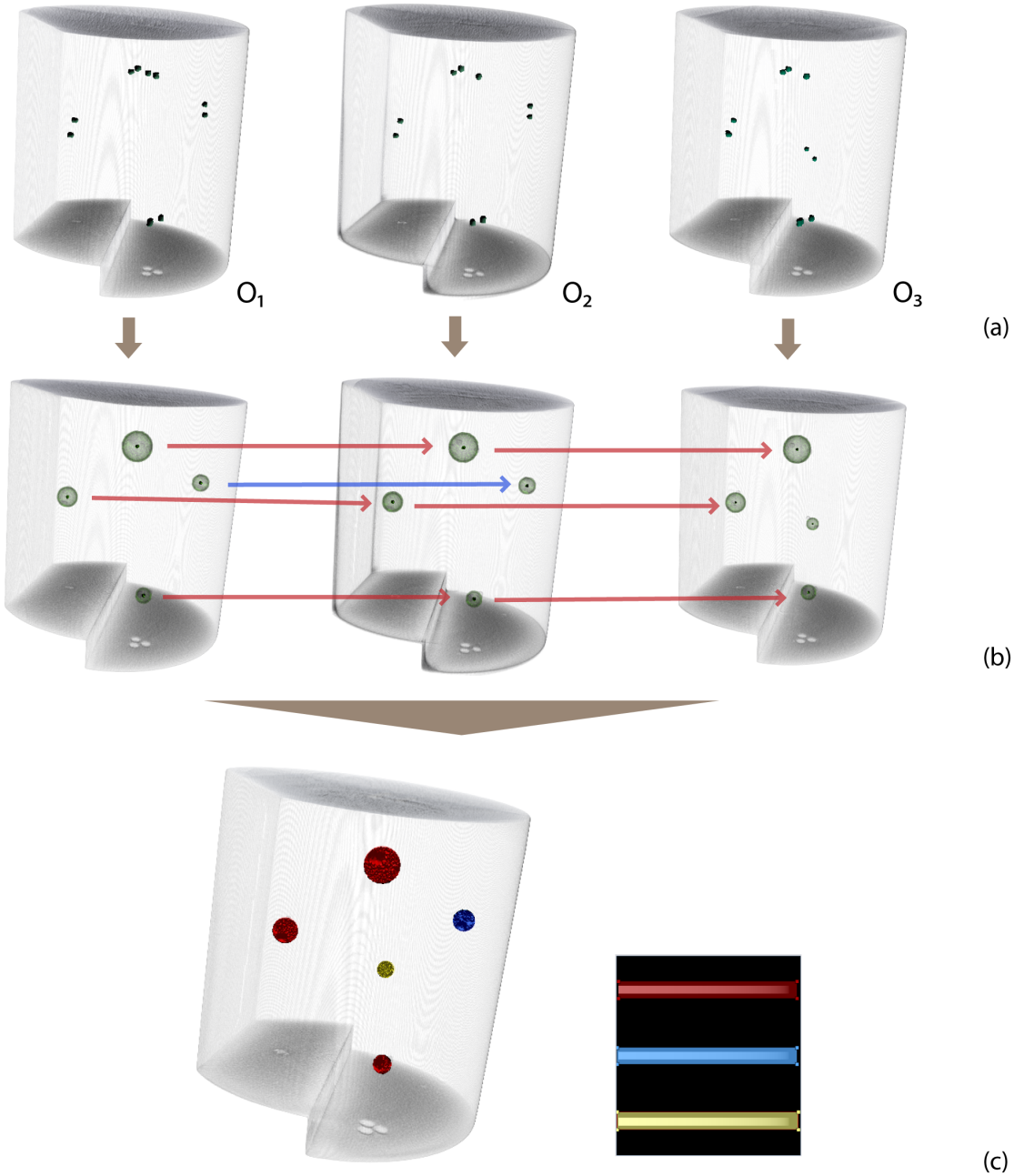


Figure 5.3: *Summary of the region-based variability analysis. The input data (a) consists of a series of volume data sets O_{1-3} and their corresponding features, which have been calculated during the defect detection step (Section 3.2). This data is used to calculate clusters, which are shown as spheres in (b). During the variability analysis, the clusters are merged based on their spatial location. This information is combined and prepared for visualisation (c). An evaluation of the results can be found in Section 5.1.2.*

The visualisation of the clusters exhibits three main parameters that are important for further evaluation steps:

- *Variability*: Clusters are assigned a variability class that defines the number of castings the cluster appears in. Clusters are grouped together based on their variability class for visualisation. The higher the variability class, the more important it is to consider this cluster during evaluation.

- *Location*: Each cluster appears at a certain place in the prototype casting. The location of a cluster is defined by its *mean* (middle point).

- *Size*: The size of each cluster defines its spreading in the casting. Since the clusters are all sphere-shaped, the size corresponds to the radius. The size is defined by the distance from the cluster mean to the outmost feature.

These three parameters are taken into account when evaluating the results. All together they comprise important information to make inferences on the variability of a foundry test run.

Evaluation of Results with Variability

Figures 5.3 and 4.4 show typical results of the region-based variability analysis. They were created using the test data set which was prepared for implementing the thesis (Section 3.1). This data set consists of three castings, therefore a maximum of three variability classes are available. Based on the visualisation of the results it is possible to draw conclusions on the *variability* and *feature distribution* within the given data set:

The number of available variability classes in the block chart inside the transfer function view depicts how many different variability classes are available. The more castings are available within the foundry test run, the more variability classes can be possibly created. For evaluating the results it is important to know whether clusters exists with a variability class value greater than *1*. In Figure 4.4 for example, three clusters with a variability class of *3* exists (painted in red) and one with a variability class of *2*. Such clusters demand special attention since they represent regions of high feature concentration that appear in more than one test castings.

Another important information is the location and size of the clusters with a high variability class. If they were located at areas where the future casting will have to withstand large forces (e.g. high pressure) in use later on, it will not be possible to accept the mould to produce errors in these regions. It will be necessary to evaluate the productions process in such cases. The size of clusters defines how far the features are spread out in these regions. In Figure 4.4 the clusters are quite small compared to the size of the casting. They are at most 9% of the object's sizes, which means that the mould produces errors in very narrow regions only. The foundry engineers may decide to accept errors if they occur in defined object regions only. In cases were clusters have a rather large size, resulting from features that are spread out over a large region, it may be necessary to adapt the production parameters to reduce these errors.

The results allow drawing conclusions about the variability of a foundry trial, but they also depict the *feature distribution* inside the prototype. The location and size of the clusters shows

were the casting process is likely to produce errors. Furthermore, the size of the clusters provides information on how far the features are spread out over the object. As noted above, the clusters in Figure 4.4 are quite small, which means features only occur in narrow regions. The results coincides with the given test data (Section 3.1), were boreholes were used to simulate the occurrence of features. These boreholes as well as the resulting features are not scattered over the whole object. When analysing the feature distribution it has to be taken into account that the variability analysis computations are done on a user-defined feature selection (Section 3.2). Therefore, the results may represent the analysis of a subgroup of features only. This way it is possible to for example make inferences on the distribution of cracks in the prototype (see also Figure 3.2).

Evaluation of Results without Variability

It may happen that the variability analysis approach locates several clusters, but is not able to find any varying variability classes. In this case the variability class of *1* is available only, and all clusters exhibit the same variability class. This means that regions of high feature concentration can be located, but they all appear in one casting only. Figure 5.4 shows how the visualisation of such results looks like: The block charts only contains one bar (one variability class), and it is only possible to either select or deselect all available clusters.

A result containing only the variability class of 1 means that it is not possible to retrieve any of the existing clusters within other objects of the test run. This in turn indicates that no region of higher feature concentration exists in more than one object. In fact this is a positive result when evaluating the serial-production readiness of a newly created mould. It means that no region can be spotted were the production process tends to produce errors more than once. Such a result will be expected after adapting production process parameters and having eliminated all possible sources of errors. Nevertheless, the results can still be used to draw conclusions on the feature distribution within the test run.

5.2 Feature-based Variability Analysis

The feature-based variability analysis finds overlaps of features of different objects within a foundry trial based on the feature voxel data (Section 3.3.2). This analysis type is mainly used to locate large features or holes that appear in more than one object. The resulting feature voxels are visualised based on their variability class, which is the main visualisation parameter. As described in Section 4.3.2, the user can select certain variability classes of interest and skip others. Figure 4.7 shows a typical result of a feature-based variability analysis.

At the one hand, the results show overlaps between features of different objects of the test run. This may concern complete features, or just parts of them. At the other hand, the feature-based approach (like the region-based approach) can be used to analyse the *feature distribution* inside the prototype. Through the visualisation regions can be identified where the production process tends to produce errors in the prototype. Furthermore, it can be seen which types of features are produced, since the shape of the features is not lost in this process. It is still possible to identify the feature classes based on the feature shapes (e.g. round, lenghty, large).

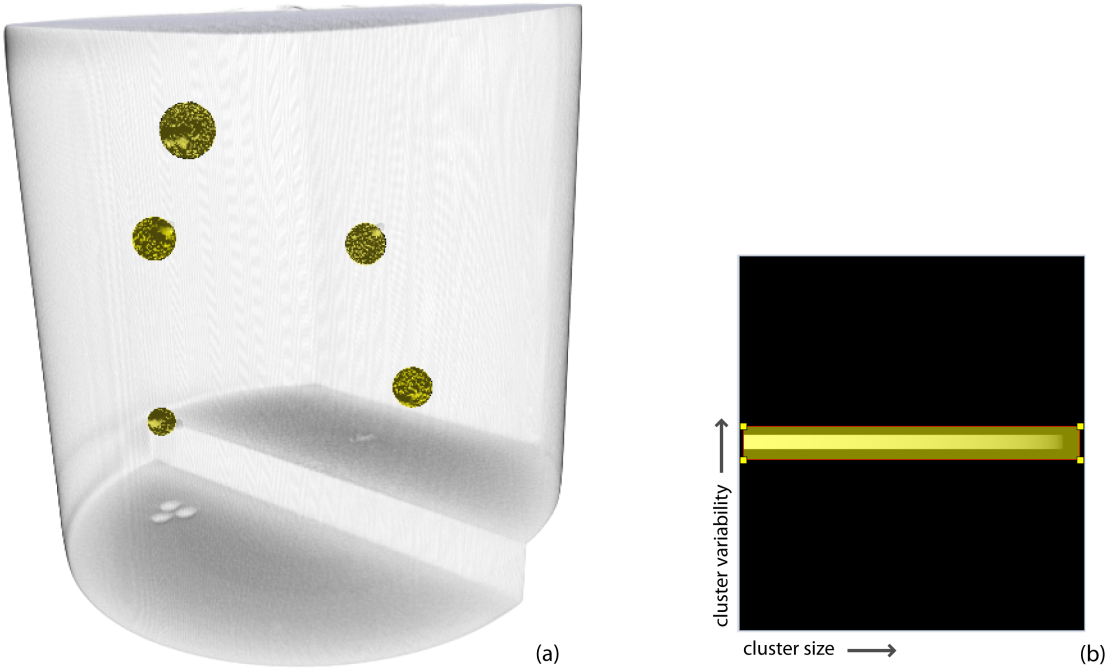


Figure 5.4: *This figure shows the result of the region-based variability analysis where one variability class is created only. The results were created by using the test data set which was prepared for implementing this thesis (Section 3.1) together with some adaptations to simulate the absence of variability. The clustering of the objects was done in a way that clusters will not have any correspondants in other objects across the series. The block chart in (b) only contains one bar for one available variability class, and therefore all clusters are painted in the same colour in (a). Such results mean that no regions can be identified where the production process tends to produce errors more than once.*

However, the voxel-based nature of this analysis approach has some major drawbacks, which has to be taken into account when evaluating the results. The analysis relies on the fact that feature voxels appear at nearly exactly the same place as their correspondents in other objects. Whereas this is an acceptable fact when analysing large features like holes, it may lead to false results when analysing small features. For large features, it is very likely that overlaps are produced, even if small shifts exist in the feature voxel data. This, in contrast, is not always the case for small features. There it may happen that features exist at the same region of interest within the objects, but they do not overlap due to shifts in the feature voxel data. Then the feature-based approach will assign a very low variability class to these feature voxels. However, the region-based approach will identify the overlapping regions and therefore will assign a higher variability class (Figure 5.5). In this case, the results of the region-based approach are practically more useful, since they make general assumptions over the whole region of interest.

(a) (b)

Figure 5.5: *Comparison of the feature- and the region-based approach. The feature-based approach assigns a very low variability class (1) to the feature voxels in (a), since they do not completely overlap. However, they are all located in the same region of interest within the objects, which can be identified by using the region-based variability analysis. This approach in turn assigns a very high variability class (3) to the computed region, since it appears in all available objects.*

5.3 Performance and Memory Usage

The following tables give an overview of the pre-computation times as well as the memory requirements for some example datasets. The performance has been measured using the following system:

- Single processor PC, AMD Athlon 64 2.4GHz, 4GB
- GeForce GTX 285, 1GB
- Windows XP64

5.3.1 Region-based Variability Analysis

The region-based variability analysis consists of two main stages: the feature clustering and the variability analysis step. During the clustering stage it is important to provide fast feedback to the user, since it is necessary to try out different parameter settings then. Afterwards it is possible to extend the run-time of the variability analysis stage according to the needs of the algorithm, since the user does not expect immediate feedback at this stage.

Clustering

For this thesis, the two well-known approaches of mean-shift and k-means clustering have been implemented. The run-time of both algorithms is fast enough to provide immediate feedback to the user. Therefore, it is possible to try out different parameter settings until the most appropriate clustering result has been found. The quality of a clustering result is evaluated by the user using

the exploration techniques described in Section 4.2.1. Table 5.1 shows some quantitative results of the run-time of the different clustering algorithms. It shows that clustering results for common volume data sets of castings can be produced in less than half a minute. The run-time mainly depends on how many iteration steps have to be performed, which does not necessarily correlate with the number of available features. It rather depends on where the clustering process has been started (initial sample points) and how easy it is to find the final position (structure of the underlying data function).

Data Set	**Features**	**mean-shift**		**k-means**	
		Clusters	**Run-time**	**Clusters**	**run-time**
Test Object	11	4	1.87 s	4	2.47 s
Rotax Small	259	11	1.19 s	11	1.16 s
Rotax	2563	35	17.58 s	35	18.21 s

Table 5.1: *Run-times of the clustering algorithms. Both clustering algorithms (mean-shift and k-means) have been used to create clustering results on single volume data sets, and the run-times of the two algorithms are compared in the table. Similar results can be produced with both algorithms at comparable run-times.*

After the final clustering has been found, temporary data structures are discarded. The final results comprise the final clusters, the list of features they cover and the clusters' histories. Since these are all float or integer values, the memory requirements for storing the clustering results are negligibly small. For the cluster exploration (Section 4.2) a bar chart of the size of the transfer function histogram given by the HVR renderer ($256x256$ pixels) is used together with a feature colour lookup table. The size of the lookup table corresponds to 16-bit per available feature.

Variability Analysis

The run-time of the variability analysis part depends on two sub-steps: the creation of the prototype volume (Section 3.4.1) and the variability analysis itself (Section 3.3.1). The prototype creation calculates the mean density value of all voxels within the objects; therefore computation linearly depends on the number of available objects and on the data set size. The run-time of the variability analysis sub-step is influenced by the number of available clusters in the objects. In summary, the run-time of the complete variability analysis step depends on:

- *Data set size*: For the prototype creation it is necessary to process all voxels of all objects to calculate the mean density value. Therefore, the run-time linearly depends on the data set size.
- *Number of objects*: The size of the test run linearly influences the run-time of the algorithm. This is because for the prototype creation voxels of all given objects have to be processed.
- *Number of clusters*: During the variability analysis, the size and location of the present clusters are compared against clusters of other objects to find overlaps. The comparison

based on the location is the most expensive task within this step. The run-time linearly depends on the number of clusters, and furthermore on the variability. For example, it may happen that all clusters are located in one object only. Then the run-time would be fast, even if a high number of clusters is given, since the clusters will not be compared against clusters from other objects.

The test run used for this thesis consists of three objects (Section 3.1) with a size of $512x512x512$ voxels respectively. For every object, four clusters are created, which leads to a total number of 12 clusters. Results for this test data set are produced within 1.2 minutes.

The final results of the region-based variability analysis comprise a prototype volume and a list of clusters. The prototype volume is of the same size as the objects within the test run ($512x512x512$ voxels for the test data set used). The final cluster information comprises float and integer values, therefore the memory requirements for storing them are negligibly small. For exploration (Section 4.3.1) a block chart of the size of the transfer function histogram given by the HVR renderer ($256x256$ pixels) is used together with a 3D volume texture with 16-bit per voxel.

5.3.2 Feature-based Variability Analysis

The run-time of the analysis depends on two sub-steps: the creation of the prototype volume (Section 3.4.1) and the variability analysis itself (Section 3.3.2). Both sub-steps linearly depend on the number of available objects and on the data set size. Like for the region-based variability analysis (Section 5.3.1), the prototype creation calculates the mean density value of all voxels in the objects. Therefore, it linearly depends on the number of available objects and on the data set size. The feature-based variability analysis needs to process the available data sets too, to locate feature voxels and calculate overlaps between feature voxels of other objects. The run-time is completely independent of the number of available features, since the variability analysis is a voxel-based approach. Results for this test data set are produced within 2.4 minutes. In summary, the run-time of the complete variability analysis depends on:

- *Data set size*: It is necessary to process all voxels of all objects twice - firstly to create the prototype volume, and secondly to locate all feature voxels within them. Therefore, the run-time linearly depends on the data set size.
- *Number of objects*: The size of the test run linearly influences the run-time of the algorithm. This is again because the volume data sets need to be processed twice during the variability analysis.

The final results of the region-based variability analysis comprise a prototype volume and a 3D volume texture with 16-bit per voxel storing the variability information. Additionally, a block chart of the specified transfer function histogram size (for this thesis: $256x256$ pixels) and a lookup table of the size of the available variability classes is used for exploration (Section 4.3.2).

CHAPTER 6

Conclusion and Future Work

This chapter draws the conclusions from the main contributions of this master's thesis. In Section 6.1 the results of the implemented variability analysis and the practical use are discussed. In Section 6.2 suggestions for future work are presented. For example, possibilities can be explored to improve the visualisation techniques to include quantificational results, or to cope with a larger number of variability classes. Another subject of future work is the involvement of statistical methods in the variability analysis computations to improve the tracking of corresponding regions and features.

6.1 Conclusion

This master's thesis presents a variability analysis pipeline for analysing and evaluating the results of a foundry trial. Whenever a new type of cast mould is created, it has to be accurately tested to avoid errors in future casting work pieces. Therefore it is very important to know whether regions within the mould exist where the production process has a certain probability to produce defects. These regions can either be large failures like misruns, but can also be represented by entrapment of smaller errors. The pipeline presented in this thesis analyses a series of castings to identify such regions in the new pattern.

Two different analysis approaches have been implemented: a region-based and a feature-based variability analysis. The region-based analysis aims at identifying regions of high error concentration and then tracks them through all castings of a foundry trial. Therefore, this approach consists of two steps. Firstly, regions of high error concentration are identified by clustering. Secondly, correspondents for all regions are searched in all castings, and their number defines the variability classes of the regions. The feature-based approach aims at tracking features across the castings of a foundry trial by inspecting the corresponding feature voxels.

The results of both implemented variability analysis approaches are visualised in a 3D rendering environment. This allows the user to explore the results and adjust the visualisation according to enhance certain parts. The complexity of the variability analysis computations are

hidden to a large extent. For the user this is the most important part, since the exploration builds the basis for evaluating the results.

The variability analysis pipeline presented in this thesis works with 3D CT scans of casting work pieces. It was implemented based on a test data set provided by the Austrian Research Foundry Institute (*Österreichisches Gießerei-Institut - ÖGI*). They created a foundry trial consisting of three objects produced by metal casting using the same mould. Additionally, boreholes were created within the objects to simulate the occurrence of spherical errors.

The results of the variability analysis are intended to assist the foundry engineer in the decision process when designing a new mould. The interpretation of the results is left to the foundry engineer, since other parameters like the casting's material, the intended usage and knowledge about weak parts within the object have to be taken into account. However, we are planning to extend the possibilities for quantification specifically with respect to analysing the feature distribution or the object's stability and usefulness. We believe that the concept presented in this thesis is very powerful, but it is also only one step towards driving first article inspection by visualisation. There are many possibilities that can be explored in the future; some of them are introduced below.

6.2 Future Work

The pipeline approach presented in this thesis constitutes the basis for enhancements in many directions. On the one hand, the cluster and feature voxel tracking between the castings is based on spatial parameters only and could be improved by applying more sophisticated statistical approaches. On the other hand, the visualisation of the results could be improved by adding more quantitative data. For example, the density of the clusters (defined by the number of features per cluster size) has not been taken into account yet.

6.2.1 Clustering Parameters

When applying region-based variability analysis (Section 3.3.1), it is necessary to cluster the feature data first. Due to the fact that finding an appropriate clustering result is NP-hard (Section 2.3.3), the evaluation of the clustering results has been transferred to the user by providing capable exploration techniques (Section 4.2). However, it is also possible to set the clustering parameters in an automated way.

The only user-defined parameter for *k-means clustering* is the number of clusters that should be created. This parameter greatly affects the quality of the resulting clustering. Therefore, it should be set automatically to support the user. Bischof et al. [7] use a minimum description length (MDL) framework, where the description length is a measure of how well the data are fit by the model. Their algorithm starts with a large value for k and removes centres (reduces k) whenever that choice reduces the description length. Pelleg and Moore [73] propose a regularization framework for learning k, called X-means. The algorithm searches over many values of k and scores each clustering model using the so-called Bayesian Information Criterion (BIC). Then X-means chooses the model with the best BIC score on the data.

When applying *mean-shift clustering*, the critical parameter affecting the quality of the clustering result is the kernel size (i.e. bandwidth). Therefore, many approaches suggest to start with an initial bandwidth and adapt it based on local data parameters [15, 20, 88, 101]. On the other hand, Einbeck [24] suggests choosing the bandwidth by analysing the underlying data first. A bandwidth will be assumed to be favourable if a high proportion of data points falls within circles (called *hypertubes*) of a certain radius.

6.2.2 Cluster Overlaps

For the implementation of the region-based variability analysis it has been decided that clusters are allowed to overlap (Section 3.3.1). However, this may lead to problems during exploration when evaluating the clustering results (Section 4.2). Therefore, the clustering approach can be extended to *fuzzy clustering*. There data elements can belong to more than one cluster. Associated with each element is a set of membership levels. The ranking of the memberships defines how probably a data element belongs to a certain cluster. One of the most widely used fuzzy clustering algorithms is the *Fuzzy C-Means* clustering approach [5].

6.2.3 Cluster Exploration

During the region-based variability analysis the user decides whether a current clustering result is appropriate to be used for further analysis computations (Section 3.3.1). This decision is made based on a cluster visualisation that paints every feature in a user-defined colour according to its cluster membership (Section 4.2). This visualisation is based on a 1D lookup table mapping feature ids to a certain colour. Due to hardware restrictions the maximum number of features that can be represented in this table is 8192. It may happen that this limit is exceeded in case of large volume data sets with a high number of features. In these cases the texture should be split up.

6.2.4 Region-based Analysis Approach

The region-based variability analysis aims at identifying regions of high feature concentration and tracking them over all objects within a given test run (Section 3.3.1). The corresponding overlapping regions (clusters) in other objects are found by their spatial location. This concept is susceptible to small shifts in the feature data. On the one hand it is assumed that all volume data sets are aligned, but on the other hand this does not ensure that regions of high feature concentration are located at exactly the same spatial position. Therefore, it would be interesting to include additional parameters like the *cluster density* into the computations. The density of a cluster is defined by the number of features covered by the cluster in proportion to the cluster size. Then it could be defined that clusters of two different objects represent the same region in case (i) the distance of their means is within a certain range and (ii) they exhibit similar feature density values.

6.2.5 Number of Variability Classes

In the presented approach the number of objects in the test run defines the maximum number of variability classes (Section 3.4.2). This is possible since the test data used consists of three objects only. However, if a test run contained significantly more objects (e.g. up to 100), it would be necessary to apply some additional calculations here to limit the maximum number of variability classes. If too many different variability classes should be displayed, the block chart presented in Section 4.3.1 would get too confusing for the user. Since foundry engineers are mainly interested in the fact whether regions of interest tend to appear more than once in the objects, it would be possible to merge several variability classes into one new class.

6.2.6 Feature-based Analysis Approach

The feature-based variability analysis tracks features over objects within a test run based on their voxel data (Section 3.3.2). This voxel-based approach relies on the fact that the correspondents of the feature voxels in other objects are located at nearly the same spatial position. Therefore, false results may be produced if small shifts exist within the feature data (Figure 5.4). Then feature voxels are assigned a too low variability class since no overlapping feature voxels can be found. This mainly happens if features are of a very small size. These analysis computations could be improved by considering the outer boundaries of the features instead of their voxel information only. A simple solution for this is to estimate the feature size with a bounding box enclosing the feature. Another possibility would be to use morphological operations as described by Jähne [49]. Mathematical morphology operates on object shape with point sets of any dimensions.

6.2.7 Quantification

The exploration of the results of the variability analysis is based on a visualisation of the clusters as well as the feature voxels (Chapter 4). The variability class is the crucical parameter where the exploration is build upon. However, it will help the foundry engineers to display some additional analysis information:

- *Object-to-Variability-Ratio*: An interesting variable will be the ratio between the object size and the size of parts affected by a high variability class. This parameter will provide information on the stability and usefulness of a prototype, since it defines how many parts exist where the productions process tends to produce errors. If nearly the complete object interior exhibits a high variability class, obviously something goes wrong during the production process. On the other hand, if a small part inside the object exhibits a high variability only, this may be an issue that could be ignored.

- *Cluster Density*: Every cluster has a certain size and covers a certain number of features. Therefore, the ratio of cluster size to features will provide information on how dense the features are given within the cluster. Additionally the features' sizes could be taken into account to see if the clusters cover many small features or one large error part only.

- *Feature Distribution*: The results of the variability analysis provide information about the feature distribution in the prototype. On the one hand, it can be identified how many parts of the objects are covered by features. On the other hand, regions can be identified where features are more likely to appear than in others.
- *Variability*: In summary, by considering all parameters mentioned above, the analysis approach will also be able to make a suggestion on the error-proneness of the given casting production process. Input from foundry engineers will be needed to define a formula or computation steps which lead to a percentage value defining the mould's suitability for serial production.

6.2.8 Cluster Picking

The exploration of the results of the region-based variability analysis is build on the available variability classes (Section 4.3.1). Therefore, it is only possible to select or deselect a complete set of clusters. However, an interesting additional exploration technique will be to be able to pick single clusters. For example, even if several clusters exhibit a high variability class, some of them may be not important for further decisions. This may be because of their size, or because they are located in an object part that will not have to withstand high pressures later on. Then it will be possible to eliminate these clusters from further visualisations. Additionally, it will be possible to display individual parameters of the picked clusters (e.g. feature density).

Bibliography

[1] Pankaj K. Agarwal and Nabil H. Mustafa. K-means projective clustering. In *Proceedings of the 23rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '04, pages 155–165, New York, NY, USA, 2004. ACM.

[2] Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. NP-hardness of Euclidean sum-of-squares clustering. *Machine Learning*, 75:245–248, May 2009.

[3] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006.

[4] Peter R. Beeley. *Foundry technology*. Butterworths, London, 1972.

[5] James C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 1981.

[6] Suchendra M. Bhandarkar, Xingzhi Luo, Richard Daniels, and E. William Tollner. Detection of cracks in computer tomography images of logs. *Pattern Recognition Letters*, 26:2282–2294, October 2005.

[7] Horst Bischof, Ales Leonardis, and Alexander Selb. MDL principle for robust vector quantisation. *Pattern Analysis and Applications*, 2(1):59–72, 1999.

[8] Herbert Boerner and Helmut Stecker. Automated X-Ray inspection of aluminum castings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10:79–91, January 1988.

[9] Richard H. Bossi, John L. Cline, Edward G. Costello, and Benjamin W. Knutson. X-Ray computed tomography of castings. Technical report, Boeing Aerospace and Electronics CO, Seattle, WA, 1990.

[10] Stefan Bruckner. Efficient volume visualization of large medical datasets. Master's thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, May 2004.

[11] Stefan Bruckner and Eduard Gröller. Style transfer functions for illustrative volume rendering. *Computer Graphics Forum*, 26(3):715–724, Sep 2007.

[12] Min Chen, Arie Kaufman, and Roni Yagel. *Volume graphics*. Springer, 2000.

[13] Yizong Cheng. Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17:790–799, 1995.

[14] Robert T. Collins. Mean-shift blob tracking through scale space. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition 2003*, CVPR '03, pages II–234–40. IEEE Comput. Soc, 2003.

[15] Dorin Comaniciu. An algorithm for data-driven bandwidth selection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(2):281–288, 2003.

[16] Dorin Comaniciu and Peter Meer. Robust analysis of feature spaces: Color image segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition 1997*, CVPR '97, pages 750–755, Washington, DC, USA, 1997. IEEE Computer Society.

[17] Dorin Comaniciu and Peter Meer. Mean shift analysis and applications. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition 1999*, CVPR '99, page 1197, Washington, DC, USA, 1999. IEEE Computer Society.

[18] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:603–619, 2002.

[19] Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer. Real-time tracking of non-rigid objects using mean shift. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition 2000*, volume 2 of *CVPR '00*, pages 142–149, 2000.

[20] Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer. The variable bandwidth mean shift and data-driven scale selection. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition 2001*, CVPR '01, page 438, Los Alamitos, CA, USA, 2001. IEEE Computer Society.

[21] Carlos D. Correa and Kwan-Liu Ma. Visibility histograms and visibility-driven transfer functions. *IEEE Transactions on Visualization and Computer Graphics*, 17:192–204, February 2011.

[22] Patrick Cozzi and Frank Stoner. GPU ray casting of virtual globes. In *ACM SIGGRAPH 2010 Posters*, SIGGRAPH '10, page 128:1, New York, NY, USA, 2010. ACM.

[23] Aleksey V. Dolgushev and Alexander V. Kelmanov. On the algorithmic complexity of a problem in cluster analysis. *Journal of Applied and Industrial Mathematics*, 5:191–194, 2011.

[24] Jochen Einbeck. Bandwidth selection for mean-shift based unsupervised learning techniques: a unified approach via self-coverage. *Journal of Pattern Recognition Research*, 2:175–192, 2011.

[25] Charles Elkan. Using the triangle inequality to accelerate k-means. Technical report, University of California, San Diego, 2003.

[26] Gereon Frahling and Christian Sohler. A fast k-means implementation using coresets. In *Proceedings of the 22nd Annual Symposium on Computational Geometry*, SCG '06, pages 135–143, New York, NY, USA, 2006. ACM.

[27] Daniel Freedman and Pavel Kisilev. Fast mean shift by compact density representation. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition 2009*, CVPR '09, pages 1818–1825. IEEE, 2009.

[28] Steffen Frey and Thomas Ertl. Accelerating raycasting utilizing volume segmentation of industrial CT data. In *Proceedings of the EGUK Theory and Practice of Computer Graphics*, TPCG '09, pages 33–40, 2009.

[29] Laura Fritz. Interactive exploration and quantification of industrial CT data. Master's thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, 1040 Vienna, Austria, 2009.

[30] Keinosuke Fukunaga and Larry Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IT*, 21(1):32–40, 1975.

[31] Georg Geier, Joerdis Rosc, Markus Hadwiger, Laura Fritz, Daniel Habe, and Thomas Pabel. Assessing casting quality using computed tomography with advanced visualization techniques. In *Proceedings of the 3rd International Symposium of The Minerals, Metals and Materials Society*, TMS '09, pages 113–139, 2009.

[32] Bogdan Georgescu, Ilan Shimshoni, and Peter Meer. Mean shift based clustering in high dimensions: A texture classification example. In *Proceedings of the IEEE International Conference on Computer Vision 2003*, CV '03, pages 456–463, 13-16 2003.

[33] Gary E. Georgeson, Alan R. Crews, and Richard H. Bossi. X-Ray computed tomography for casting development. Technical report, Boeing Aerospace and Electronics CO, Seattle, WA, 1992.

[34] Markus Hadwiger, Laura Fritz, Christof Rezk-Salama, Thomas Höllt, Georg Geier, and Thomas Pabel. Interactive volume exploration for feature detection and quantification in industrial CT data. *IEEE Transactions on Visualization and Computer Graphics*, 14:1507–1514, November 2008.

[35] Markus Hadwiger, Joe M. Kniss, Christof Rezk-Salama, Daniel Weiskopf, and Klaus Engel. *Real-time Volume Graphics*. Ak Peters Series. A K Peters, Ltd., 2006.

[36] Markus Hadwiger, Christian Sigg, Henning Scharsach, Katja Bühler, and Markus H. Gross. Real-time ray-casting and advanced shading of discrete isosurfaces. *Computer Graphics Forum*, 24(3):303–312, 2005.

[37] John A. Hartigan and Manchek A. Wong. A k-means clustering algorithm. *Applied Statistics*, 28:100–108, 1979.

[38] Stephan Hasse. *Guß- und Gefügefehler: Erkennung, Deutung und Vermeidung von Guß- und Gefügefehlern bei der Erzeugung von gegossenen Komponenten*. Schiele & Schön, 2003.

[39] Hermann Hecker. *Ein neues Verfahren zur robusten Röntgenbildauswertung in der automatischen Gußteilprüfung*. Berlin, 1995.

[40] Christoph Heinzl. *Analysis and Visualization of Industrial CT Data*. PhD thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, 12 2009.

[41] Sergio Hernandez, Doris Saez, Domingo Mery, Ricardo da Silva, and Maria Sequeira. Automated defect detection in aluminium castings and welds using neuro-fuzzy classifiers. In *Proceedings of the 16th World Conference on Non-Destructive Testing*, WCNDT '04, Montreal, Canada, 2004.

[42] Frank Herold, Klaus Bavendiek, and Rolf-Rainer Grigat. A new analysis and classification method for automatic defect recognition in X-Ray images of castings. In *Proceedings of the 8th European Conference on Non-Destructive Testing*, ECNDT '02, June 2002.

[43] Derek L.G. Hill, Philipp G. Batchelor, Mark Holden, and David J. Hawkes. Medical image registration. *Physics in Medicine and Biology*, 46(3):R1, 2001.

[44] Jiri Hladuvka, Andreas König, and Eduard Gröller. Curvature-based transfer functions for direct volume rendering. In *Proceedings of the Spring Conference on Computer Graphics and its Applications 2000*, SCCG '00, pages 58–65, 2000.

[45] Wei Hong, Feng Qiu, and Ariel Kaufman. GPU-based object-order ray-casting for large datasets. In *Proceedings of the 4th International Workshop on Volume Graphics*, VG '05, pages 177–240, 2005.

[46] Runzhen Huang, Kwan-Liu Ma, Patrick McCormick, and William Ward. Visualizing industrial CT volume data for nondestructive testing applications. In *Proceedings of the 14th IEEE Visualization*, VIS '03, pages 72–, Washington, DC, USA, 2003. IEEE Computer Society.

[47] Xiao-Yuan Huang, Concurrent Technologies Corporation, and Brian G. Thomas. Modeling of transient flow phenomena in continuous casting of steel. *Industrial Engineering*, 37(304):197–212, 1998.

[48] Mark M. Hytros, Imad M. Jureidini, Jung-Hoon Chun, Richard C. Lanza, and Nannaji Saka. High-energy X-Ray computed tomography of the progression of the solidification front in pure aluminum. *Metallurgical and Materials Transactions*, 30(5):1403–1409, 1999.

[49] Bernd Jähne. *Digital image processing: concepts, algorithms, and scientific applications*. Springer-Verlag, 1991.

[50] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:881–892, July 2002.

[51] Leonard Kaufman and Peter J. Rousseeuw. *Finding Groups in Data An Introduction to Cluster Analysis*. Wiley Interscience, New York, 1990.

[52] Richard A. Ketcham and William D. Carlson. Acquisition, optimization and interpretation of X-Ray computed tomographic imagery: applications to the geosciences. *Computational Geosciences*, 27(4):381–400, 2001.

[53] Gordon Kindlmann and James W. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *Proceedings of the IEEE Symposium on Volume Visualization*, VVS '98, pages 79–86, 1998.

[54] Gordon Kindlmann, Ross Whitaker, Tolga Tasdizen, and Torsten Möller. Curvature-based transfer functions for direct volume rendering: Methods and applications. In *Proceedings of the 14th IEEE Visualization 2003*, VIS '03, pages 67–, Washington, DC, USA, 2003. IEEE Computer Society.

[55] Joe Kniss, Gordon Kindlmann, and Charles Hansen. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *Proceedings of the Conference on Visualization 2001*, VIS '01, pages 255–262, Washington, DC, USA, 2001. IEEE Computer Society.

[56] Jean Kor, Xiang Chen, Zhizhong Sun, and Henry Hu. Casting design through multi-objective optimization. In *Proceedings of the International Seminar on Future Information Technology and Management Engineering 2009*, FITME '09, pages 604–608, Los Alamitos, CA, USA, 2009. IEEE Computer Society.

[57] Piotr Kraj, Ashok Sharma, Nikhil Garge, Robert Podolsky, and Richard A. McIndoe. ParaKMeans: Implementation of a parallelized k-means algorithm suitable for general laboratory use. *BMC Bioinformatics*, 9:200+, Apr 2008.

[58] Jens Kruger and Rüdiger Westermann. Acceleration techniques for GPU-based volume rendering. In *Proceedings of the 14th IEEE Visualization 2003*, VIS '03, pages 38–, Washington, DC, USA, 2003. IEEE Computer Society.

[59] Sang Su Lee, Dongwoo Won, and Dennis McLeod. Discovering relationships among tags and geotags. In *Proceedings of the International AAAI Conference on Weblogs and Social Media 2008*, ICWSM '08, 2008.

[60] Marc Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8:29–37, May 1988.

[61] Fajie Li and Reinhard Klette. A variant of adaptive mean shift-based clustering. In *Advances in Neuro-Information Processing*, volume 5506 of *Lecture Notes in Computer Science*, pages 1002–1009. Springer Berlin / Heidelberg, 2009.

[62] Barthold Lichtenbelt, Randy Crane, and Shaz Naqvi. *Introduction to volume rendering*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1998.

[63] Stuart P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28:129–137, 1982.

[64] James B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[65] Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. The planar k-means problem is NP-hard. In *Proceedings of the 3rd International Workshop on Algorithms and Computation*, WALCOM '09, pages 274–285, Berlin, Heidelberg, 2009. Springer-Verlag.

[66] Stephan Mantler and Stefan Jeschke. Interactive landscape visualization using GPU ray casting. In *Proceedings of the 4th International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia*, GRAPHITE '06, pages 117–126, New York, NY, USA, 2006. ACM.

[67] Nelson Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1:99–108, June 1995.

[68] Ketan Mehta and Thomas J. Jankun-Kelly. Detection and visualization of defects in 3D unstructured models of nematic liquid crystals. *IEEE Transactions on Visualization and Computer Graphics*, 12:1045–1052, September 2006.

[69] Domingo Mery and Dieter Filbert. Verfolgung von Gußfehlern in einer digitalen Röntgenbildsequenz - Eine neue Methode zur Automatisierung der Qualitätskontrolle von Gußteilen. *tm - Technisches Messen*, 67(4):160–165, 2000.

[70] Domingo Mery and Dieter Filbert. Automated flaw detection in aluminum castings based on the tracking of potential defects in a radioscopic image sequence. *IEEE Transactions on Robotics*, 18(6):890–901, 2002.

[71] Thomas Pabel, Georg F. Geier, Daniel Habe, Joerdis Rosc, Peter Schumacher, and Tose Petkov. *Correlation of Porosity Detected by Computed Tomography and Fatigue Strength of Aluminium Alloys*, pages 803–805. John Wiley & Sons, Inc., 2011.

[72] Sungchan Park, Youngmin Ha, and Hong Jeong. A parallel and memory-efficient mean shift filter on a regular graph. In *Proceedings of the International Conference on Intelligent Pervasive Computing 2007*, IPC' 07, pages 254–259, 2007.

[73] Dan Pelleg and Andrew Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *Proceedings of the 7th International Conference on Machine Learning*, ICML '00, pages 727–734, San Francisco, 2000. Morgan Kaufmann.

[74] Ning Song Peng, Jie Yang, and Zhi Liu. Mean shift blob tracking with kernel histogram filtering and hypothesis testing. *Pattern Recognition Letters*, 26(5):605–614, 2005.

[75] Graeme P. Penney, Jürgen Weese, John A. Little, Paul Desmedt, Derek L.G. Hill, and David J. Hawkes. A comparison of similarity measures for use in 2D-3D medical image registration. *IEEE Transactions on Medical Imaging*, 17(4):586–595, 1998.

[76] Hanspeter Pfister, Bill Lorensen, Chandrajit Bajaj, Gordon Kindlmann, Will Schroeder, Lisa Avila Sobierajski, Ken Martin, Raghu Machiraju, and Jinho Lee. The transfer function bake-off. *IEEE Computer Graphics and Applications*, 21:16–22, May 2001.

[77] Huy Le Phat, Toan Nguyen Dinh, Sang-Cheol Park, and Gueesang Lee. Text localization in natural scene images by mean-shift clustering and parallel edge feature. In *Proceedings of the 5th International Conference on Ubiquitous Information Management and Communication*, ICUIMC '11, page 116, 2011.

[78] Matthew Profit, Sivakumar Kulasegaram, Javier Bonet, and Roland W. Lewis. Modelling of high pressure die casting using the CSPH method. In *Proceedings of the 3rd International Conference on Engineering Computational Technology*, ECT '02, pages 103–104, Edinburgh, UK, 2002.

[79] Posinasetti Nageswara Rao. *Manufacturing technology: foundry, forming and welding*. Tata McGraw-Hill Education, 2 edition, 2001.

[80] Christof Reinhart, Christoph Poliwoda, Thomas Günther, Wolfgang Roemer, Stefan Maass, and Christian Gosch. Modern voxel based data and geometry analysis software tools for industrial CT. In *Proceedings of the 16th World Conference on Non-Destructive Testing*, WCNDT '04, Montreal, Canada, 2004.

[81] Christof Rezk-Salama, Maik Keller, and Peter Kohlmann. High-level user interfaces for transfer function design with semantics. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1021–1028, 2006.

[82] Stefan Roettger, Stefan Guthe, Daniel Weiskopf, Thomas Ertl, and Wolfgang Strasser. Smart hardware-accelerated volume rendering. In *Proceedings of the Symposium on Data Visualisation 2003*, VISSYM '03, pages 231–238, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[83] Henning Scharsach. Advanced GPU raycasting. In *Proceedings of the Central European Seminar on Computer Graphics 2005*, CESCG '05, pages 69–76, 2005.

[84] Henning Scharsach, Markus Hadwiger, André Neubauer, Katja Bühler, and Stefan Wolfsberger. Perspective isosurface and direct volume rendering for virtual endoscopy applications. *Proceedings of EUROVIS*, pages 315 – 322, 2006.

[85] Dave Shreiner, Mason Woo, Jackie Neider, and Tom Davis. *OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL(R), Version 2 (5th Edition)*. Addison-Wesley Professional, aug 2005.

[86] Christian Sigg, Tim Weyrich, Mario Botsch, and Markus H. Gross. GPU-based ray-casting of quadratic surfaces. In *Proceedings of the Symposium on Point Based Graphics 2006*, SPBG '06, pages 59–65. Eurographics Association, 2006.

[87] Martin Simon and Christoph Sauerwein. Quality control of light metal castings by 3D computed tomography. In *Proceedings of the 15th World Conference on Non-Destructive Testing*, WCNDT '05, Rome, Italy, 2005.

[88] Maneesh Singh and Narendra Ahuja. Regression based bandwidth selection for segmentation using parzen windows. In *Proceedings of the IEEE International Conference on Computer Vision 2003*, ACCV '03, page 2, Los Alamitos, CA, USA, 2003. IEEE Computer Society.

[89] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image Processing, Analysis, and Machine Vision*. Chapman & Hall, 2. edition, 1998.

[90] Andreas Tappenbeck, Bernhard Preim, and Volker Dicken. Distanzabhängige Transferfunktionen für die medizinische Volumenvisualisierung. In *Proceedings of Bildverarbeitung für die Medizin 2005*, pages 307–311, 2005.

[91] Matthias Teßmann, Stephan Mohr, Svitlana Gayetskyy, Ulf Haßler, Randolf Hanke, and Günther Greiner. Automatic determination of fiber-length distribution in composite material using 3D CT data. *EURASIP Journal on Advances in Signal Processing*, 2010:1:1–1:9, February 2010.

[92] Brian G. Thomas and Lifeng Zhang. Mathematical modeling of fluid flow in continuous casting. *ISIJ International*, 41(10):1181–1193, 2001.

[93] Edward J. Vinarcik. *High integrity die casting processes*. Number Bd. 1 in Materials Science: Engineering. Wiley, 2003.

[94] Jue Wang, Bo Thiesson, Yingqing Xu, and Michael Cohen. Image and video segmentation by anisotropic kernel mean shift. In *Computer Vision - ECCV 2004*, pages 238–249. Springer, 2004.

[95] Wikipedia. k-means clustering | Wikipedia. The Free Encyclopedia. http://en.wikipedia.org/wiki/K-means_clustering. Last accessed on 2011-09-11.

[96] Wikipedia. Volume ray casting | Wikipedia. The Free Encyclopedia. http://en.wikipedia.org/wiki/Volume_ray_casting. Last accessed on 2011-09-10.

[97] Gregory A. Wilkin and Xiuzhen Huang. K-means clustering algorithms: Implementation and comparison. In *Proceedings of the International Multi-Symposiums on Computer and Computational Sciences 2007*, IMSCCS '07, pages 133–136. IEEE, 2007.

[98] Brett Wilson, Kwan-Liu Ma, and Patrick S. McCormick. A hardware-assisted hybrid rendering technique for interactive volume visualization. *IEEE Symposium on Volume Visualization and Graphics*, pages 123–130, 2002.

[99] Chunxia Xiao and Meng Liu. Efficient mean-shift clustering using gaussian KD-tree. *Computer Graphics Forum*, 29(7):2065–2073, 2010.

[100] Junqiao Xiong. Cutting display of industrial CT volume data based on GPU. *Advanced Materials Research*, 271-273:271–273, 2011.

[101] Qi Zhao, Zhi Yang, Hai Tao, and Wentai Liu. Evolving mean shift with adaptive bandwidth: A fast and noise robust approach. In *Proceedings of the 9th Asian Conference on Computer Vision*, ACCV '09, pages 258–268, 2009.

[102] Weizhong Zhao, Huifang Ma, and Qing He. Parallel k-means clustering based on MapReduce. In *Cloud Computing*, volume 5931, chapter 71, pages 674–679. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[103] Fangfang Zhou, Ying Zhao, and Kwan-Liu Ma. Parallel mean shift for interactive volume segmentation. In *Proceedings of the 1st international conference on Machine learning in Medical Imaging*, MLMI '10, pages 67–75, Berlin, Heidelberg, 2010. Springer-Verlag.

[104] Jianlong Zhou, Andreas Döring, and Klaus D. Tönnies. Distance transfer function based rendering. Technical report, Institute for Simulation and Graphics, University of Magdeburg, Germany, 2004.