MASTERARBEIT

Interactive Exploration and Quantification of Industrial CT Data

ausgeführt am Institut für Computergraphik und Algorithmen
der Technischen Universität Wien
in Kooperation mit dem
VRVis, Zentrum für Virtual Reality und Visualisierung

unter Anleitung von
Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
in Kooperation mit
Dipl.-Ing. Dr.techn. Markus Hadwiger

durch
Laura Fritz
Matr. Nr.: 9826587
A - 8042 Graz, Peterstalstrasse 129

Wien, im Jänner 2009

Abstract

Non-destructive testing (NDT) is a key aspect of present day engineering and development which examines the internal structures of industrial components such as machine parts, pipes and ropes without destroying them. Industrial pieces require critical inspection before they are assembled into a finished product in order to ensure safety, stability, and usefulness of the finished object. Therefore, the goal of this thesis is to explore industrial Computed Tomography (CT) volumes, with the goal to facilitate the whole quantification approach of the components at hand by bridging the gap between visualization on the one hand, and interactive quantification of features or defects on the other one.

The standard approach for defect detection in industrial CT builds on region growing, which requires manually tuning parameters such as target ranges for density and size, variance, as well as sometimes also the specification of seed points. To circumvent repeating the whole process if the region growing results are not satisfactory, the method presented in this thesis allows interactive exploration of the parameter space. The exploration process is completely separated from region growing in an unattended pre-processing stage where the seeds are set automatically. The pre-computation results in a *feature volume* that tracks a *feature size curve* for each voxel over *time*, which is identified with the main region growing parameter such as variance. Additionally, a novel 3D transfer function domain over *(density, feature_size, time)* is presented which allows for interactive exploration of feature classes. Features and feature size curves can also be explored individually, which helps with transfer function specification and allows coloring individual features and disabling features resulting from CT artifacts. Based on the classification obtained through exploration, the classified features can be quantified immediately.
The visualization and quantification results of this thesis are demonstrated on different real-world industrial CT data sets.

Kurzfassung

Zerstörungsfreie Werkstoffprüfung (ZfP) ist heutzutage ein Schlüsselaspekt in der Maschinenbau- und Bauindustrie, und dient zur Inspizierung der internen Strukturen von, zum Beispiel Maschinenteilen, Rohren und Drahtseilen, ohne dabei ihre Struktur zu zerstören. Werkstücke, welche in der Industrie angefertigt werden, müssen einer strengen Kontrolle unterzogen werden, um die Stabilität und generelle Verwendbarkeit zu garantieren, bevor sie zur Produktion zugelassen werden. Diese Diplomarbeit wurde speziell zur Untersuchung von industriellen Computer Tomographischen (CT) Daten durchgeführt. Das Ziel, den gesamten Quantifizierungsablauf eines vorliegenden Werkstücks zu erleichtern, wird durch interaktive Quantifizierung auf visueller Ebene ermöglicht.

Der üblicherweise verwendete Anstatz zur Erkennung von Ungänzen in Bauteilen basiert auf Region-Growing-Methoden, welche sowohl das Setzen von Parametern wie Varianz-, Dichte- und Featuregrössenintervall verlangen, als auch manchmal ein manuelles Setzen von Seeds voraussetzt. Um zu vermeiden, dass der gesamte Region-Growing-Prozess bei nichtzufriedenstellenden Ergebnissen wiederholt werden muss, erlaubt die in dieser Diplomarbeit vorgelegte Methode eine interaktive Erforschung des Parameterraums, welche sich komplett von dem zu Grunde liegenden Region-Growing-Prozess abhebt. Auch das Setzen von Seeds passiert automatisch. Das zuvor berechnete *Feature-Volumen* verfolgt für jedes Voxel eine eigene *Featuregrössen-Kurve* über die Zeit, welche durch den wichtigsten Region-Growing-Parameter (in unserem Fall die Varianz) bestimmt wird. Zusätzlich wird eine neue 3D Transfer Funktion über *(Dichte, Featuregrössen, Zeit)* vorgestellt, welche zur interaktiven Untersuchung der verschiedenen Klassen von Ungänzen herangezogen werden kann. Ungänzen- und Featuregrössenkurven können auch individuell betrachtet werden, was die Einstellung der Transfer Funktion erleichtern soll. Die einzelnen Ungänzen können je nach Bedarf farblich hervorgehoben, oder von der

Quantifizierung ausgeschlossen werden (z.B. durch die CT Aufnahme entstandene Artefakte). Die Quantifizierung der erhaltenen Featureklassen erfolgt zeitgleich mit der Exploration.
Die Visualisierungs- und Quantifizierungs-Ergebnissse werden anhand verschiedener, Daten aus der Industrie gezeigt.

Contents

1 Introduction **1**
1.1 Motivation . . . 2
1.2 Problem Statement and Objectives . . . 3
1.3 Organization and Pipeline Overview . . . 5

2 Fundamentals and State of the Art **8**
2.1 Industrial CT . . . 8
2.2 GPU-Based Direct Volume Rendering . . . 11
2.3 Region Growing . . . 15
2.4 Multi-Dimensional Transfer Functions . . . 19
2.5 3D Visualization of Industrial CT Data . . . 24

3 Pre-Computation **27**
3.1 Parameter-Settings . . . 27
3.2 Feature-Size Curves . . . 28
3.3 Feature Volume . . . 30
3.4 Feature-Growth Table . . . 31
3.4.1 Storing the Feature-Growth Table . . . 32
3.4.2 Merging Features . . . 33
3.5 Multi-pass Region Growing and Seed Selection . . . 34
3.5.1 Culling . . . 35
3.5.2 Region Growing . . . 37
3.6 Region Growing Criteria . . . 41
3.6.1 Region growing method A . . . 42
3.6.2 Region growing method B . . . 42
3.7 Statistical Feature Properties . . . 43

4 Exploration **45**
4.1 Exploring Feature Classes . . . 46
4.1.1 Feature Histograms . . . 47
4.1.2 3D Transfer Functions and 2.5D Widgets . . . 48
4.2 Exploring Individual Features . . . 49
4.2.1 Feature Picking . . . 51
4.2.2 Feature Color Coding . . . 52
4.2.3 Removal of Artifacts . . . 53
4.2.4 Feature Table . . . 53
4.2.5 Optional Slice Plane . . . 55
4.3 Volume Rendering . . . 56
4.3.1 Brick Caching . . . 56
4.3.2 Rendering . . . 58

5 Quantification and Results **61**
5.1 Feature Quantification . . . 61
5.2 Results . . . 62
5.2.1 Reduced-Pressure-Test Sample . . . 63
5.2.2 Asphalt Core . . . 64
5.2.3 Golf Ball . . . 66
5.2.4 Laser Build-up Welding . . . 67
5.2.5 Refractory Material . . . 69
5.2.6 Isolation Material . . . 70
5.2.7 Aluminum Tensile Test Sample . . . 71
5.2.8 Cast Housing I . . . 72
5.2.9 Cast Housing II . . . 74
5.3 Performance and Memory Usage . . . 75

6 Conclusion and Future Work **79**
6.1 Conclusion . . . 79
6.2 Future Work . . . 80

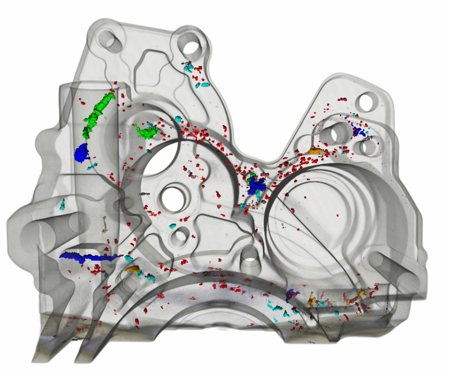
Acknowledgments **82**

Bibliography **iii**

Chapter 1

Introduction

This work presents the development of a novel method for interactive exploration of industrial CT volumes such as cast metal parts or even more complex components composed of completely different materials, like minerals. The goal is to interactively detect, classify, and quantify features using a visualization-driven approach which allows exploration of all feature classes irrespective of specific density, size, or variance characteristics, e.g., cracks and holes in casting parts (Figure 1.1(a)) or


(a) Cast Housing.

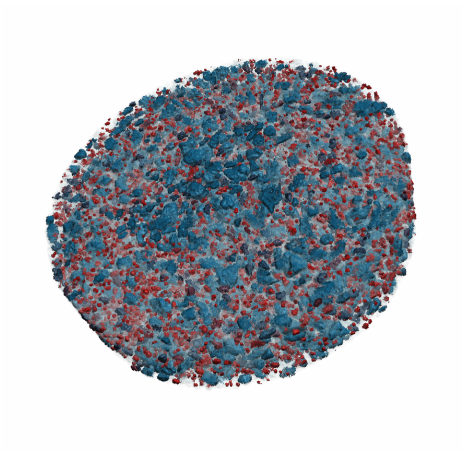

(b) Refractory Material.

Figure 1.1: *Features can be explored and quantified interactively according to their size and density. Therefore, a 3D transfer function is used to identify and colorize the different feature classes. (a) Defect detection in a Cast Housing. (b) Identification of different mineral phases in a refractory material.*

different material components (Figure 1.1(b)).

This thesis has been carried out in the scope of a research cooperation between the VRVis and the Austrian Foundry Research Institute (ÖGI) [2008], which has already resulted in several publications (Hadwiger et al. [2008], Geier et al. [2008a] and Geier et al. [2008b]), as well as the study thesis of Höllt [2007].

1.1 Motivation

Non-destructive testing (NDT) is a scientific discipline which examines the internal structures of industrial components such as machine parts, pipes and ropes without destroying them. It is an essential tool in construction engineering and manufacturing, especially in the automotive industry, heavy machinery industry and plant industry. In cast metal parts, for example, the processes during solidification may cause shrinkage cavities, pores, cracks, or inhomogeneities to appear inside the structure, which are not visible from the outside.

This work was developed in co-operation with the Austrian Foundry Research Institute and especially created to explore industrial Computed Tomography (CT) volumes, with the goal to facilitate the whole quantification approach of the components at hand. A common class of features in this context are defects such as pores and shrinkage cavities in varying cast parts, as well as more complex materials with a broad spectrum of different inclusions, like minerals or asphalt.

The CT-data recorded by the ÖGI originate on one hand from various industries which need material testing and on the other hand from their own pilot foundry. In the pilot foundry all cast-materials such as cast irons, cast steels, copper alloys, Al- and Mg-alloys can be cast and afterward tested in the mechanical testing laboratory. In this case NDT facilitates the evaluation of the behavior of such pilot castings at each stage of the testing process as well as the assessment of material defects which arise throughout the manufacturing process or during use which is simulated in the mechanical testing laboratory. Furthermore, NDT is nowadays not only used for inspecting metal parts, but for a variety of different materials such as plastics, wood, or concrete, as well as minerals in general.

1.2 Problem Statement and Objectives

In recent years, 3D CT has become common in NDT, which has created powerful new possibilities, but also new challenges for the inspection and testing process. Industrial CT volumes are generally quite large, with voxels commonly stored with 16 bits of precision, which leads to several hundred MB to one or more GB of raw data per scan. Real-time volume rendering has become an essential tool for visualizing these volumes, usually using bricking strategies as described in Engel et al. [2006] to cope with the large data sizes. However, for NDT practitioners visualization is just one part of the workflow, which includes a variety of processing tasks such as defect detection and quantification, computing statistical measures and properties such as material porosity, performing accurate measurements and comparisons, and many more.

The goal of this work is to help bridge the gap between visualization on the one hand, and quantification of features or defects on the other one. In the NDT context, feature detection is usually performed via some kind of segmentation, which most commonly builds on region growing and filtering operations such as morphological operators. Segmentation results in one or several static segmentation masks, which can be visualized as part of the 3D volume and also form the basis of quantification. However, the segmentation cannot be modified without re-computation. This decouples the detection of features from visualization and prevents working in a fully interactive manner. Most of all, it hampers interactively exploring the volume for all different kinds of contained features without already knowing what is contained in the volume. Segmentation parameters are specified, the segmentation is computed, and when the results are not satisfactory the user has to change the parameters and the entire segmentation has to be computed all over again. This is often time-consuming and tedious. In Figure 1.2 three different values for the segmentation parameters are used to estimate an appropriate threshold for region growing. This example clarifies, that in many cases it is impossible to get a satisfying result using only a single value. For some inclusions the selected parameter is too small, which causes that the region is only partially grown, or important features are even ignored completely (e.g., marked by the blue circle in Figure 1.2(a) and 1.2(b)). Whereas, a larger parameter causes oversegmentation in other regions (marked by the yellow circle in Figure 1.2(a) and 1.2(b)). In contrast to this standard approach, this thesis proposes a *visualization-driven* method for feature detection,

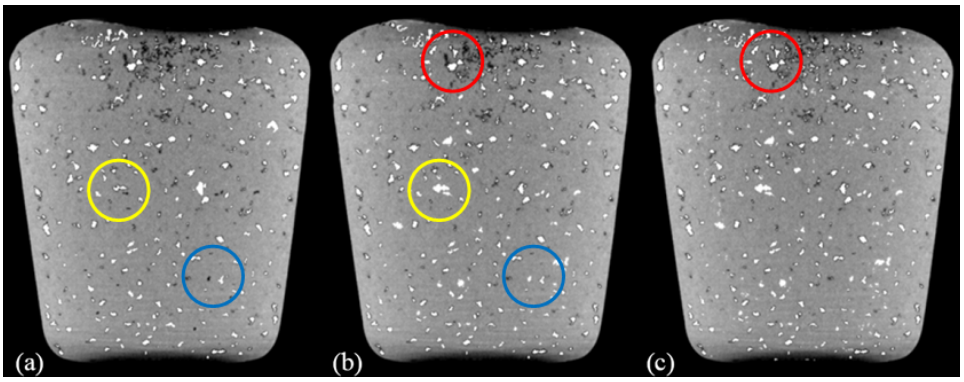


Figure 1.2: *Three results for different segmentation parameters in a conventional region growing approach. A too small threshold causes that inclusions are not or not completely segmented (marked by the blue circle in Figure (a) and (b), and the red circle in Figure (b) and (c)). However a too large threshold includes too much of the surrounding material into the feature (yellow circle in Figure (a) and (b)).*

where features in the volume can be explored interactively without re-computing segmentation information. The basis for this is an unattended pre-computation stage that computes a *feature volume* and some additional data structures, which contains the result of feature detection over *parameter domains* instead of fixed parameters. This pre-computation has to be performed only once for a given data set and forms the basis of interactively exploring all contained features. Moreover, in contrast to detection of a single type or class of features, such as "feature class *defect*," we allow the user to explore all feature classes and decide interactively which classes and features are of interest, instead of specifying this information beforehand. This is especially useful in the context of compound parts or complex materials such as minerals, where a broad range of different features with different densities and sizes emerge. In the traditional workflow, in order to detect defects in cast metal parts, for example, the parameters that need to be set for the segmentation are such that the density of defects must be below a certain threshold (assuming that air or gas comprises the interior of defects), their size must be larger than a given minimum (features that are too small are noise), or smaller than a given maximum (features that are too big are no features anymore but, e.g., intended holes in a cast part). Moreover, further parameters must be set for the region growing process, for example a maximum density variance in the region, or maximum standard deviation from the neighborhood of a seed voxel. The system might also require the user to manually specify seed voxels or set parameters for automatic seed determination. In contrast, our system computes and records the result of region growing for the entire density domain, all different sizes of features, and the entire domain of the most important region growing parameter (given a specific region growing algorithm), such as maximum variance. Therefore, it is sufficient to specify just a wide range over which the computation takes place. For generality, this parameter is referred to as the “time” parameter t throughout this thesis. Together, these three 1D parameter ranges comprise the 3D *(density, feature_size, time)* domain, which is explored by the user via 3D transfer functions. In order to make transfer function specification tractable, a 2.5D metaphor is employed, which still provides the necessary flexibility.

To summarize, the major contributions of this thesis are:

- A fully interactive workflow for exploring features in industrial CT scans for detection, classification, and quantification.
- Relocation of the underlying region growing process to an unattended pre-computation stage, while still allowing the corresponding parameter space to be explored later on.
- A new 3D transfer function domain for exploring the parameter space of feature detection to determine *feature classes*.
- Quantification of only those feature classes found to be of interest during exploration, which empowers the domain expert to interactively control the final result.

1.3 Organization and Pipeline Overview

This thesis is organized as follows: Chapter 2 summarizes the current state of the art and fundamentals of the main categories used in this thesis. It starts with a brief description of the basics concerning industrial CT data acquisition and reconstruction. The second part of this chapter focuses on the idea and concept behind direct volume rendering of GPU-based ray-casting. Further, some region growing techniques are mentioned and explained, especially those used in this thesis. In recent years multi-dimensional transfer functions have become more and more popular and contribute significantly to the image results in volume visualization. A state of the art of the most established transfer functions is given, focusing on 2D

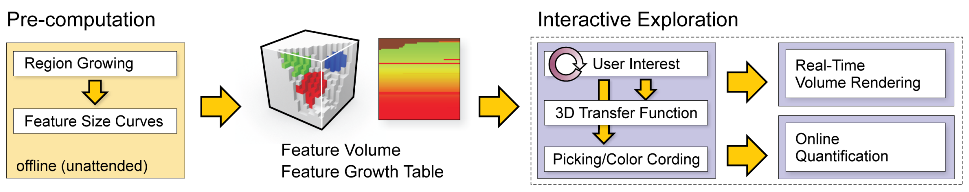


Figure 1.3: *Overview of the workflow pipeline presented in this thesis. The pre-computation stage computes feature size curves via multi-pass region growing, which are stored in a feature volume and a feature growth table. These are the basis for the subsequent interactive exploration stage (Hadwiger et al. [2008]).*

transfer functions (TFs). Finally, some already existing visualization applications for industrial CT data are introduced.

The main part of this thesis includes the different stages of the pipeline illustrated in Figure 1.3. As a pre-requisite, for a given CT volume additional information must be pre-computed, as explained in Chapter 3, which constitutes the basis of interactive feature exploration. A multi-pass region growing approach (Section 3.5) is employed that conceptually computes a *feature size curve* over "time" t (which corresponds to the main region growing parameter) for each voxel in the volume (Section 3.2). For memory efficiency, these curves are stored split up into a 3D *feature volume* (Section 3.3), and a corresponding 2D *feature growth table* (Section 3.4).

When the feature volume and growth table are available, the data set can be explored interactively for features of interest in the *exploration* stage, described in Chapter 4. Figure 1.1 shows example images from the exploration of a cast metal part and a mineral. Feature exploration builds on the specification of a 3D transfer function in the *(density, feature_size, time)* domain (Section 4.1.2), which is constituted by the CT density volume, the feature volume, and the feature growth table. Transfer function specification is not only the means by which the user determines the visualization, but also how features to be quantified are selected. Features can also be explored individually using a graphical feature view or picking in orthogonal slice views and the feature tables (Section 4.2). They are also used to remove specific features from rendering and quantification that are artifacts from the CT acquisition process. During exploration, the current feature classification is displayed using real-time volume rendering (Section 4.3). From the feature classification specified by the user during the interactive exploration phase, the *quantification* stage, explained in Section 5.1, automatically computes statistical measures such as feature count, volume, and surface area for features that have been selected in the exploration stage. That is, quantification is performed in a visualization-driven manner, where everything that is selected for feature visualization is included in the quantification. Both feature exploration and quantification can be performed as often as desired without requiring additional pre-computation. Section 5.2 discusses the results that were achieved with this system for real-world industrial CT-data. Image quality and performance are compared in Section 5.3 as well as the advantages and shortcomings for the different algorithms of the underlying region growing process, expenditure of time and memory usage. Chapter 6 finally concludes by summarizing the results and gives an outlook into possible future work.

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. Section 2.1 starts with a brief abstract of Industrial Computed Tomography where the main differences between medical and industrial applications are shown. Section 2.2 explains the basics of direct volume rendering with the main focus on GPU-based approaches. A current state of the art is given and the VRVis Hardware Volume Renderer is introduced, where the work of this thesis has been embedded. Region growing is used for segmentation of the inclusions occurring in the data. After explaining the basic concepts, the current state of the art of region growing is given in Section 2.3. In Section 2.4 the application of transfer functions is explained which is the part mostly related to the user and also a summary of the current technology of 2D and multidimensional TFs is given. Finally, Section 2.5 shows some existing visualization applications for industrial CT data.

2.1 Industrial CT

Industrial CT is a completely nondestructive technique for visualizing features in the interior of opaque solid objects, and for obtaining digital information on their 3D geometries and properties, which is useful for a wide range of materials. First developed for widespread use in medicine for the imaging of soft tissue and bone, X-ray CT was subsequently extended and adapted to a wide variety of industrial tasks. Because industrial CT systems image only non-organic, non-alive objects, they can be designed to take advantage of the fact that the items being studied do not move and are not harmed by X-rays. Therefore it is possible to scan objects in high resolution. Seeing that, Ketcham and Carlson [2001] quote the following optimizations:

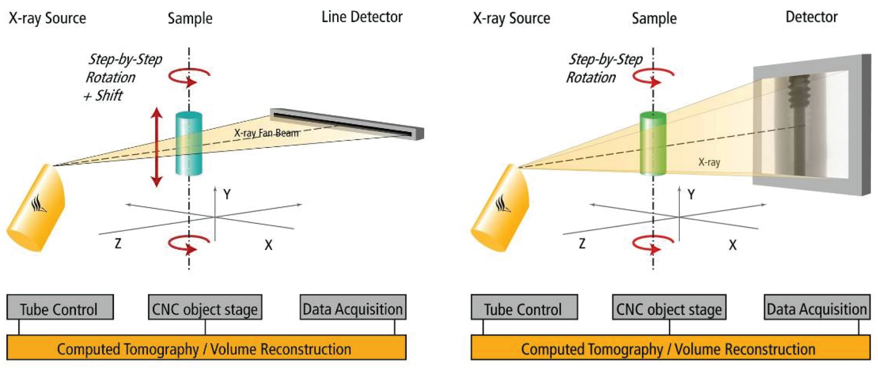
1. Use of higher-energy X-rays, which are more effective at penetrating dense materials;
2. Use of smaller X-ray focal spots, providing increased resolution at a cost in X-ray output;
3. Use of finer, more densely packed X-ray detectors, which also increases resolution at a cost in detection efficiency;
4. Use of longer exposure times, increasing the signal-to-noise ratio to compensate for the loss in signal from the diminished output and efficiency of the source and detectors.

Beside sample preparation and machine calibration two main steps are required to generate a 3D data volume:

Data Acquisition

The gray levels in a CT slice correspond to X-ray attenuation, which reflects the proportion of X-rays scattered or absorbed as they pass through each voxel. X-ray attenuation is primarily a function of X-ray energy and the density and atomic number of the material being imaged. A CT image is created by directing X-rays from multiple orientations through the subject and measuring their resultant decrease in intensity. By acquiring a stacked, contiguous series of CT images, data describing an entire volume can be obtained. Generally we can differentiate between 2D- and 3D data acquisition (see Figure 2.1).

As illustrated in Figure 2.1(a) for 2D data acquisition a scan plane is imaged where the fan beam and detector series are wide enough to encompass the entire object, and thus only rotation and vertical movement of the object is required. As the object



(a) Data acquisition with 2D industrial computed tomography.

(b) Data acquisition with 3D industrial computed tomography.

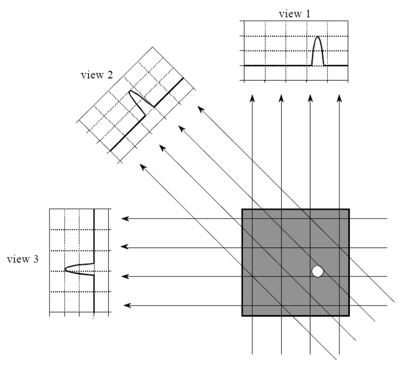
Figure 2.1: *(a) Stepwise line acquisition by rotating the workpart* 360° *(stepsize* $\leq 1^\circ$*) and stepwise switching of the workpart. (b) Stepwise projection acquisition by rotating the workpart* 360° *(stepsize* $\leq 1^\circ$*). Images by courtesy of phoenix|x-ray Systems [2003].*

rotates, it passes through the fan beam for the current vertical position. This is done for each slice position (vertical position), which permits reconstruction of a complete image.

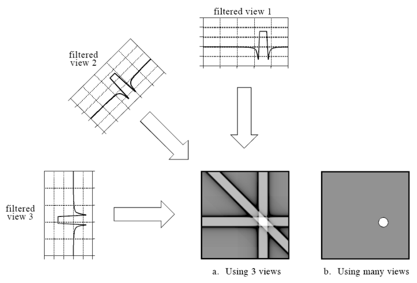
In volume CT (Figure 2.1(b)), a cone beam or highly-collimated, thick, parallel beam is used rather than a fan beam, and a planar grid replaces the linear series of detectors. This allows for much faster data acquisition, as the data required for multiple slices can be acquired in one rotation.

Data Reconstruction

Reconstruction is the mathematical process of converting the distribution of X-ray attenuation in the slice plane into two-dimensional slice images. The most widespread reconstruction technique is called filtered backprojection, in which the data are first convolved with a filter and each view is successively superimposed over a square grid at an angle corresponding to its acquisition angle (Figure 2.2). This removes the blurring which occurs in simple backprojection, and results in a mathematically exact reconstruction of the image.



(a) CT data acquisition.



(b) Data reconstruction with filtered backprojection.

Figure 2.2: *(a) A set of views is taken from many directions where the sum of the image values along each ray is built. This sample is then used to reconstruct the corresponding image. A typical CT scan uses hundreds of views at slightly different angles. (b) Filtered backprojection is used to reconstruct the image. In each projection step the respective view is smeared back along the path it was originally acquired. To prevent blurring, each view is filtered before backprojection. Images by courtesy of Smith [1997].*

2.2 GPU-Based Direct Volume Rendering

Generally, direct volume rendering (DVR) is a technique for rendering a 2D image from a sampled 3D scalar field without first fitting geometric primitives to the samples as in indirect methods. Indirect methods, like the marching cubes of Lorensen and Cline [1987] or isosurface based methods by Thévenaz and Unser [2003], need a preprocessing step to extract surfaces which are then rendered with the traditional polygon based method.

DVR operates directly on the volume data with the goal of simulating the interaction between light and a volume, most commonly given as a regularly sampled grid of densities. Therefore an optical model is used to simplify the light transport equation of Engel et al. [2006]:

$$\frac{dI(s)}{ds} = -\kappa(s)I(s) + g(s), \tag{2.1}$$

where I denotes the radiance, κ denotes the attenuation caused by absorption and out-scattering, and the term $g(s)$ is composed of *true emission* and emission caused by *in-scattering*. The survey paper of Max [1995] reviews several different optical models for light interaction with volume densities including all kinds of combinations of absorption, emission and scattering with and without shadows. A

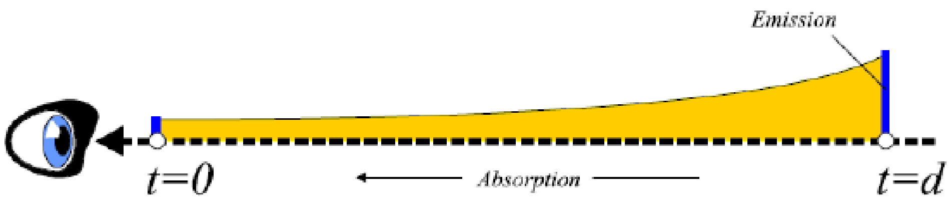


Figure 2.3: *In the emission-absorption model the amount of light reaching the eye after it has been absorbed by the volume is calculated. Image by courtesy of Engel et al. [2004].*

common emission-absorption model proposed by Blinn [1982] is based on a low albedo situation where scattering and higher order lighting effects can be ignored. This means that any element of the participating medium emits and absorbs light at the same time. The common theme, however, is to evaluate an approximation of the volume rendering integral for each pixel where the one most commonly used in the literature is given as:

$$I = \int_{s_0}^{s_n} c(t) e^{-\tau(t,s_n)} dt, \tag{2.2}$$

where c denotes the emission coefficient and the absorption is given by the optical depth τ. Hence, it is integrated over absorption and emission along a ray illustrated in Figure 2.3, which means that color and opacity values are *composited* using an appropriate blending function. With the *Front-to-Back* compositing algorithm for example, acceleration techniques, like the early-ray termination by Krüger and Westermann [2003], can be integrated:

$$Col = Col + (1 - \alpha_{akk}) C_i \alpha_l \tag{2.3}$$

$$\alpha_{akk} = \alpha_{akk} + (1 - \alpha_{akk}) \alpha_i, \tag{2.4}$$

where Col and α_{akk} indicate the accumulated color and opacity taken at the sample points (C_i and α_i) along a ray. According to the model, the mapping of volume densities to optical properties like color and opacity is done through a transfer function T, which in the simplest case is a lookup table (a 1D texture for a GPU based implementation):

$$T(f(x,y,z)) = \{R,G,B,\alpha\}, \tag{2.5}$$

where $f(x,y,z)$ denotes a scalar function over $\mathbb{R}^3$ that returns a density value at position (x,y,z) which is mapped to a color $\{r,g,b\}$ and opacity α value. The order of interpolation and application of the transfer function is differentiated by Engel et al. [2001] into *pre-* and *post-classification*, depending on which occurs first. Post-classification is characterized by applying the transfer function *after* interpolation and pre-classification vice versa. This classification process is described in more detail in Section 2.4.

In the context of DVR, the most common solutions can be separated into *object-* and *image-order* approaches. Object-order approaches are for example cell projection, shear-warp, splatting or texture-based algorithms. The most popular image-order approach for DVR is *ray-casting*, first proposed by Levoy [1988], where for every pixel a ray is cast through the volume, compositing the resulting values from sample points along its way using Equation 2.3. The first GPU-based implementations for volume rendering by Roettger et al. [2003] and Krüger and Westermann [2003] used multi-pass ray-casting where the ray traversal process is initiated by the CPU. Single-pass ray-casting requires support of loops to iterate along a ray entirely in the fragment shader and for dynamic branching to stop the traversal, which was not possible with older graphics hardware. Since the implementation of shader model 3.0 in 2004, loops are supported. This makes the implementation of single-pass approaches possible.

For the visualization of 3D scalar fields, many successful hardware based techniques have been proposed in the past, including 2D- (by Rezk-Salama et al. [2000]) and 3D-texture mapping (by Wilson et al. [1994]). Generally, in GPU-based volume rendering the data is stored as multiple 2D or one 3D texture, depending on the specific implementation and GPU, and is loaded onto the graphics board. Rezk-Salama et al. [2000] use the multi-texturing and multi-stage rasterization capabilities of consumer graphics boards to enable interactive high quality volume visualization of the 2D-texture based approach. To enable real trilinear interpolation, which overcomes the usual visual artifacts caused by stacks of 2D slices, intermediate slices are computed on the fly. The third interpolation step is then performed within the rasterization hardware using multitextures for blending the two texels resulting from bilinear interpolation. A rectilinear dataset is converted into a 3D texture map containing color and opacity information in Wilson et al. [1994]. To render an image, the texture map is applied to a stack of parallel planes cutting the texture in slices which are finally composited. Engel et al. [2001] present a pre-integrated texture-based algorithm for 2D and 3D textures, respectively, which implements an additional texture look-up of pre-integrated colors and opacities stored in a 2D texture. The VolumePro system of Pfister et al. [1999] implements ray-casting with parallel slice-by-slice processing using special purpose hardware to achieve higher image quality. Effective solutions have been introduced for rendering large (by Guthe et al. [2002]) and time-varying volume data (by Lum et al. [2001]). A detailed overview of real-time volume rendering can be found in the book by Engel et al. [2006].

In hardware based ray-casting the volume is stored in 3D textures. During the main ray-casting pass, resampling is performed by fetching samples from this texture using hardware-native trilinear filtering. For each pixel, compositing of the sample locations along a ray is performed until the volume is exited or full opacity is reached. To step along the ray, the entry position as well as the direction and the length of the ray have to be computed. Therefore, a bounding box in the $[0,1]$ range of the volume is rendered with (r,g,b) color coding of each vertex position in object space and two images are rasterized. The first image determines the volume entry positions of the rays by rasterizing the front faces of the geometry, shown in Figure 2.4(a). The second image is created by rasterizing the back faces, shown in Figure 2.4(b). The direction texture is calculated by the difference of the back faces and the front faces (Figure 2.4(c)) and yields the ray direction vectors (stored in the RGB components of the texture) and the ray length (stored in the alpha channel). These textures are used by the fragment program to determine the starting sample locations and the number of sample points that have to be taken along the ray. The density values are fetched along the rays, mapped to the (r,g,b) and α values according to the transfer function (Equation 2.5) and composited (Equations 2.3 and 2.4) to approximate the volume rendering equation 2.2. As an image order

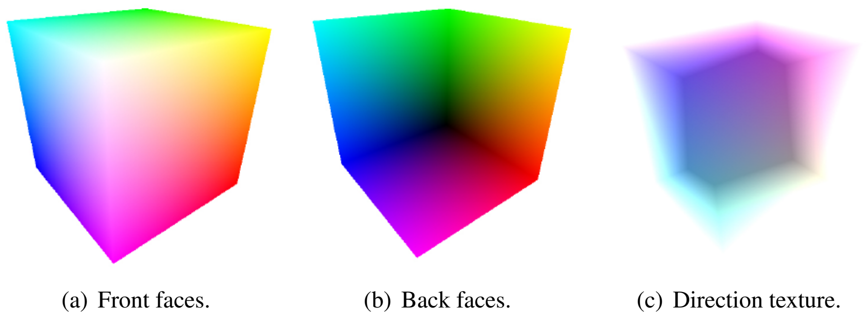

(a) Front faces. (b) Back faces. (c) Direction texture.

Figure 2.4: *The setup for the ray-casting process stores all relevant geometry information in the respective textures. Images by courtesy of Scharsach et al. [2006].*

approach, raycasting benefits from the parallel architecture of the modern GPUs as the computation of every pixel can occur in parallel without any mutual knowledge.

The *high-quality hardware volume renderer* (HVR) used in this thesis is a hardware based approach which includes several performance optimizations like *empty space skipping*, described in Scharsach et al. [2006]. Therefore, the volume is subdivided into small blocks containing a fixed number of voxels. Only non-empty blocks with an opacity greater than zero become active and form the data-dependent bounding geometry which is finally passed to the GPU. This approach is also necessary for rendering large data, to reduce memory consumption, which is described later in Section 4.3.1. A second acceleration technique, called *early ray termination*, is used to stop the ray traversal when the ray has left the volume or when the accumulated opacity value reaches a constant threshold.

2.3 Region Growing

Region growing is a fundamental image processing technique for segmentation (Zucker [1976];Sonka et al. [1993]) which assumes that neighboring voxels within the same region are homogeneous with respect to a certain prespecified criterion (e.g., similar intensity values). Generally the neighboring voxels are checked against a homogeneity criterion that determines if a voxel can be classified into the same region as its neighbor. With respect to computational complexity, region growing belongs to the class of non-uniform problems, whose run-time complexity is strongly data dependent and cannot be determined at compilation time. Simple implementations are based on aggregation starting from a specified seed point in a recursive way and can be summarized by the following steps.

- Start by choosing an arbitrary *seed pixel* (manually or automatically) and compare it with neighboring pixels.
- The region is grown from the seed pixel by adding neighboring pixels according to a homogeneity criterion, increasing the size of the region.
- When the growth of one region stops, choose another seed pixel which does not yet belong to any region and start the process again.

Seeded region growing (SRG) by Adams and Bischof [1994] is a bottom up regionbased segmentation method, that performs segmentation of an intensity image with respect to a given set of points, called *seeds*. Based on the seeds the regions are grown. Therefore, the choice of seeds is crucial as it strongly influences the resulting features of interest, and what is irrelevant in the volume. To add a new voxel x to the existing region, the voxel most similar to the region has to be found, which is estimated as follows:

$$\delta(x) = |g(x) - mean_{y \in A_{i(x)}}[g(y)]\,|, \tag{2.6}$$

where $\delta(x)$ defines a measure of how different x is from the region it adjoins, $g(x)$ is the gray value of the voxel and $A_{i(x)}$ is the region to where the new voxel should be added. This difference is estimated for all the neighbour voxels of the region and finally, that one with the smalles difference $\delta(x)$ is taken to the region. So, we take a voxel z of testet voxels T such that:

$$\delta(z) = min_{x \in T}\{\delta(x)\}, \tag{2.7}$$

and append it to $A_{i(x)}$. This process is repeated until all voxels have been allocated. Additionally, a threshold can be applied which stops the growing of a region when $\delta(x)$ exceeds a given value. To handle merging of different regions, all initial regions are marked with different IDs. If the neighboring voxel belongs to a different area it is flagged as a boundary voxel and added to a boundary set. The procedure is iterative: The most suitable adjacent voxel is added until the whole volume is classified. In our application we use a variation of this approach which automatically chooses seeds and iteratively increases the threshold for Equation 2.7, explained in detail in Section 3.6.1. This region growing method suffers from several issues, e.g. concerning the choice of seeds or absence of performance improvements. Therefore, over the years many approaches have been developed which varied from the one above, to overcome certain challenges for 2D as well as for 3D applications. Some of them are described below.

Fiorentini et al. [2003] describes a FIFO structure which is used to consider all neighbors of the seedvoxel and all consecutive accepted region voxels, for the growing process. For each region the initial seed voxel is put into the queue where its six neighbors are tested against the homogeneity criterion. Every voxel which satisfies the criterion is also added to the queue and successively used for testing its neighbors in turn, until the queue is empty. This process ensures connectivity between the segmented voxels and is one of the basics of our region growing approach.

A more efficient implementation by Horowitz and Pavlidis [1974] uses a split-and-merge strategy. Different research groups have suggested parallel implementations of region growing, like Willebeek-LeMair and Reeves [1990], Khan and Gillies [1992], or Copty et al. [1994]. Segmentation using traditional low-level image processing techniques, like region growing, often requires considerable amounts of interactive expert guidance, where the specification of the seed points and the homogeneity metric is essential for region growing techniques to work properly. Many research groups have suggested methods to simplify and automatize their specification. In the technique described by Adams and Bischof [1994], the homogeneity criterion is omitted by the use of several different seed points which are evaluated simultaneously. The volume seedlings approach described by Cohen et al. [1992] represents an interactive technique for specifying seed points to select regions of interest for volume visualization. However, since this approach is working in screen space, it is restricted to a static viewpoint. In Leila et al. [2008] seed selection is optimized with *particle swarm optimization* (PSO). The agents are placed on random locations in the image and choose the pixel with the smallest gradient as seed to ensure location within the region. During the growing process the local threshold is adjusted according to the region's structure.

Other techniques derive the homogeneity criterion from statistical information about the local neighborhood of the seed point, mainly the mean value and the standard deviation. In 3D, such techniques are closely related to automatic techniques for iso-surface extraction. Tenginakai et al. [2001] propose a method for detection of salient iso-surfaces based on the evaluation of higher-order statistical moments, such as skewness and kurtosis. Techniques such as the contour trees by van Kreveld et al. [1997] and the contour spectrum by Bajaj et al. [1997] analytically evaluate information about the topology, the surface area, and volume of iso-surfaces. This information is then used for feature classification. Zhu and Yuille [1996] present a technique called *region competition*, using a combination of snake/balloon models and the statistical techniques of region growing to minimize a generalized Bayes/Minimum Description Length (MDL) criterion. Pohle and Toennies [2001] present a region growing method which estimates the homogeneity model from the image itself to overcome the partial volume effect (PVE). Therefore, two runs of region growing are necessary. In the first run the homogeneity parameters are estimated, including mean and two different standard deviations from the grayvalues of the region. The standard deviations build a range for the grayvalues of the homogeneity function. This range may be adapted by a relaxation function during the learning process for estimating the lower and upper threshold. In the second step the regions are extracted.

Huang and Ma [2003] integrate a region growing technique into their volume visualization systems. Besides full segmentation, their technique may perform region growing on a partial data range in order to define a 2D transfer function for volume rendering. Their method starts with a kd tree partitioning of the histogram region defined from region growing. For each leaf node only the data points with values over a threshold are kept, to remove noise. Two classifiers merge when their common area is similar or less than the sum of both. An additional technique which uses the result of the region growing process to systematically construct a boundary surface, is also introduced in this paper, where a boundary tracking algorithm is used to identify all the boundary faces which enclose the extracted voxels. Such a visualization, however, will not be exact compared to the full segmentation. Although the visualization is fast and effective, modifying the seed points at runtime will also require re-computation. Huang et al. [2003] also demonstrate an application of their region growing technique for non-destructive testing of CT data, described below in Section 2.5, which is effective, but underlies the same limitations for interactive exploration and quantification.

Selle et al. [2002] use threshold based seeded region growing for vessel segmentation to generate a 3D model for analyzing the patient's intrahepatic vascular system from medical CT data. To avoid repeating the region growing process with different thresholds until an appropriate result is found, the primary threshold is refined stepwise to automatically find the optimal one. Therefore, an initial seed is set interactively in the portal vein. Next, the 26 neighbor voxels are accumulated iteratively when the intensity is greater or equal to the starting threshold of the seed voxel. This process is continued by using the neighbor voxels as new seeds and decreasing the threshold until a given end-threshold, which creates just voxels outside the vessel system, is reached. For each step a list with the appropriate voxels, for the current threshold, is generated for interactively estimating the segmentation result.

Unseeded region growing by Lin et al. [2001] is a fully automatic segmentation method, which is based on SRG. The need of placing the seeds manually, however, is overcome and therefore no pre-knowledge of the data is required. The algorithm starts with one arbitrary pixel in the image and grows the first region with this pixel as seed. If the differece measure of Equation 2.7 is valid for an adjacent pixel it is added to the region. Otherwise, the most similar region is chosen where the homogeneity criterion is valid. If neither of the two possibilities applies, a new region with the current voxel as seed is grown. This is an iterative process, which is repeated until all pixels belong to a region. In our application, we use a similar iterative approach for automatic seed selection (see Section 3.5.2), but with the difference that our segmentation result for one time step not necessarily needs to include all voxels of the entire volume. This depends on what is defined as potential feature in the parameter settings.

There exists a huge number of different region growing applications where the approaches mentioned above are just a small part of them. The reason for choosing the two methods described in Section 3.6 for this application, was more or less arbitrary, but they turned out to work well and were easy to implement as a proof-of-concept. To expand the pre-computation stage with further improved region growing options will be a task for future work, which will provide an interesting and potential improvement for this application.

2.4 Multi-Dimensional Transfer Functions

In general, a transfer function (TF) is used in direct volume rendering to facilitate volume exploration by assigning optical properties to values of a dataset, which are then composited along the viewing direction into an image. Therefore, the user can express the visualization goals with the help of a TF. Pfister et al. [2001] define a transfer function as follows:

> “A transfer function assigns values for optical properties, such as color and opacity, to original values of the dataset being visualized.”

Hence Šereda [2004] defines the TF as a mapping from the data property domain D to the domain of optical properties O:

$$D_1 \times D_2 \times \ldots \times D_n \rightarrow O_1 \times O_2 \times \ldots \times O_n. \tag{2.8}$$

In other words, the TF is a function which defines how different parts of the volume will be displayed in the volume renderer. Thus, the result and the amount of information we can see from the final 2D image, highly depends on the definition of an appropriate transfer function. However, defining the visualization goal through a TF is something not very natural, and often needs pre-knowledge of the underlying data. Therefore, the goal of current research is to make the TF design more intuitive as well as to provide a process which automatically proposes a TF for a given dataset, to support the user in finding the optimal solution.

Accordingly, we start with the following premises:

- The datasets we are concerned with are industrial CT-data.
- A regular grid delivers the mapping $D : G \rightarrow \mathbb{R}$ for a grid sample. To get the values between the grid samples, an interpolation function is used which extends the area to the whole space of $\mathbb{R}$. These values are for example the density, or the gradient magnitude but also many other values like the curvature can be calculated. Therefore, for each application the question arises which properties are suitable for the needs and the underlying data.

  In this thesis, we introduce a 3D transfer function applying the *density-, feature size-, time*-domain. The density consists of the single scalar from the CT-data where the meaning of the value is the transmittance of the tissue towards the X-rays. The feature size defines the different sizes of the inclusions in the dataset which are segmented in the underlying region growing process, described in Section 3.5.2. Tracking this region growing process over several iterations results in the third dimension – the time. This turned out to provide good visualization possibilities for the segmented features despite the large feature-size range. The details are further discussed in Section 4.1.
- A transfer function is a mapping of data properties to optical properties. In our case this includes the color-transfer function $c(x) := (r, g, b)$, which is a mapping to the RGB-colorspace and the opacity-transfer function $o(x) := a$, which assigns the opacity of a value where 0 means completely transparent

and 1 completely opaque. These are the most common used optical properties first introduced by Levoy [1988]. But there exist also other approaches using optical properties like the Phong coefficients [1975].

In recent years, usability aspects have become more and more prominent in volume rendering systems. Since a 1D transfer function is often not sufficient to separate different materials of the same density, the use of 2D, 3D, or even multi-dimensional transfer functions is becoming more and more popular. The first 2D transfer function was proposed by Levoy [1988], who added the gradient as the second dimension in order to classify the boundaries of different classes of objects. Since then, a lot of applications have been developed using different kinds of properties for the additional dimensions. In their seminal paper, Kindlmann and Durkin [1998] describe a semi-automatic transfer function design which can be regarded as the basis for multi-dimensional classification. Although their transfer function was still one-dimensional, the gradient magnitude and the second order directional derivative of the scalar field were taken into account. True multi-dimensional transfer functions were introduced by Kniss et al. [2001]. Here the gradient magnitude and the second order derivative were pre-computed for each voxel and a 3D transfer function was applied for classification. The authors also proposed a user-interface for multi-

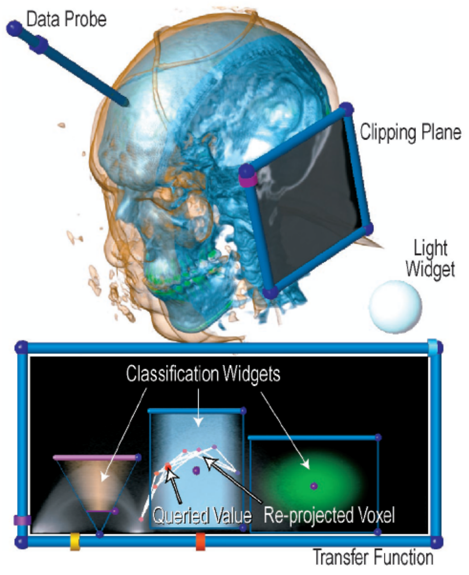


Figure 2.5: *Manipulation widgets can be used in the 2D TF and for better visualization in the volume. Image by courtesy of Kniss et al. [2001].*

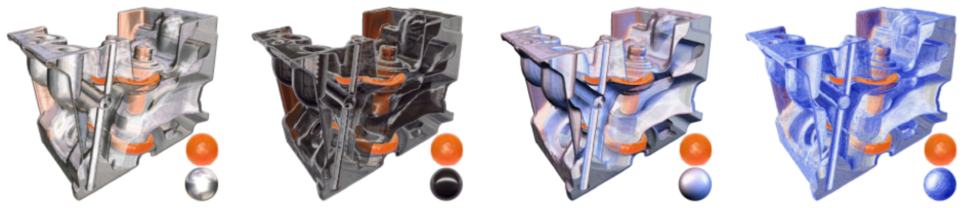


Figure 2.6: *Different non-photorealistic rendering approaches can be combined with the style transfer function. The lighting models are shown at the bottom right respectively. Images by courtesy of Bruckner and Gröller [2007].*

dimensional classification based on interaction in both the spatial domain and the feature space of the transfer function (dual domain interaction; see Figure 2.5). Because the accuracy of the second derivative is highly susceptible to noise, Lum and Ma [2004] provide a combination of a 1D TF for the assignment of color and opacity and a 2D lighting TF for the assignment of ambient, specular and diffuse coefficients. Therefore, for each sample two additional scalar values are taken into account, one in the gradient and one in the opposite direction. These values provide an indication whether a material boundary occurs at a given location and which kind of material exists on each side of the boundary for better extraction. Bruckner and Gröller [2007] extend this approach by additionally providing a variety of non-photorealistic shading styles. A new concept of style transfer functions is presented, where not only color and opacity are assigned, but additional lighting models are used to combine a multitude of different shading styles interactively in a single transfer function (Figure 2.6). Hladuvka et al. [2000] introduce a (semi)automatic transfer function in the domain of principal curvature magnitudes with the main goal to distinguish among different shape classes. It also provides a smooth color and/or opacity transition within thick surfaces or solid objects. Kindlmann et al. [2003] advance the use of curvature information in 2D TFs by combining an implicit formulation of curvature with a convolution-based reconstruction of the field. The curvature-based TFs are used with different kinds of derivatives (Figure 2.7) and applied for non-photorealistic rendering, surface smoothing via anisotropic diffusion, and visualization of isosurface uncertainty. To overcome the problem that the intensity range of vessels is very small and often overlaps with other tissue classes, Chan et al. [2006] use a 2D transfer function including a filtering response as second dimension besides the density. For this purpose a filter for curvilinear structures and a line filter are applied to extract a clear image revealing the vessel structures which are finally used for the 2D TF. Another approach for better tissue separation is provided by Lundström et al. [2005]. They propose a fuzzy classification based on local histograms as second TF dimension. Additional partial range histograms in an automatic tissue detection scheme are provided in conjunction with adaptive trapezoids to support efficient TF design. Takahashi et al. [2004] extract the topological structure of a volume dataset resulting in a volume skeleton tree, which consists of volumetric critical points and their connectivity. The resultant graph provides critical field values whose color and opacity are accentuated in the automatic design of transfer functions for direct volume rendering. This approach constitutes a 3D extension of traditional computer-aided design, where a variety of design-oriented edges (e.g., highlights and outlines) are devised for parameterizing the design quality of surfaces.

Another task, mainly used in medical applications, incorporates distance information into the transfer function space, for example, to estimate the condition and environment of tumors. In Zhou et al. [2004] the user has to define the focal center point of interest to which the Euclidean distance of all sample positions is pre-calculated. When the focal region center changes a recalculation has to occur. This distance information is combined with the information of the original volume to finally assign optical properties. In Tappenbeck et al. [2005] the distance calculations are restricted to a pre-segmented object of reference. For this object the user can define optical properties according to different distance ranges and intensity values. The main problem for many non-expert users, as for example medical staff, is that it is very difficult and time consuming to achieve satisfactory visual results even when a pre-knowledge of the underlying data exists. To ease the use of transfer functions, Rezk-Salama et al. [2006] provide interfaces with semantic information. The semantic models are designed from reference datasets

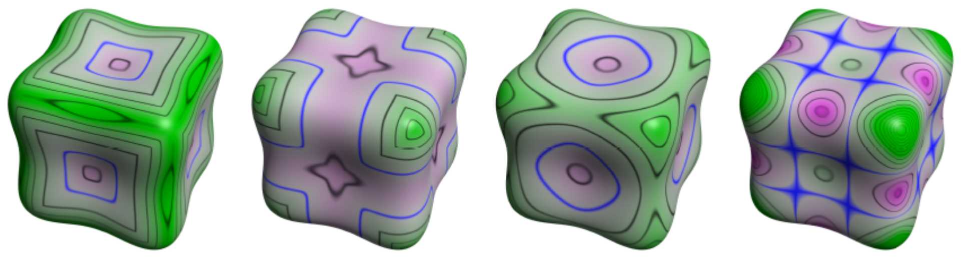


Figure 2.7: *Four different curvature measures. Left to right: first principal curvature, second principal curvature, mean curvature and Gaussian curvature. Magenta indicates negative, green positive curvature. Zero curvature is in blue, all other iso-curvature contours in black. Images by courtesy of Kindlmann et al. [2003].*

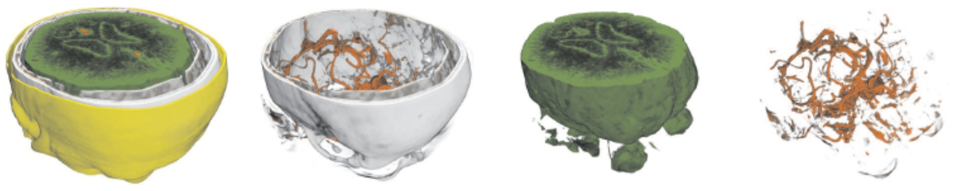


Figure 2.8: *Semantic models allow to extract the relevant structures (from left: skin, bone, brain tissue and blood vessels). Images by courtesy of Rezk-Salama et al. [2006].*

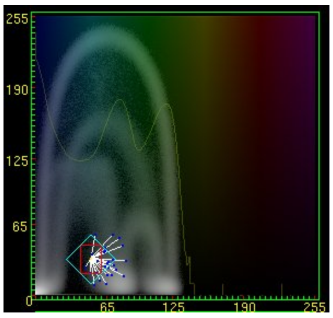
containing the relevant structures as shown in Figure 2.8, which are represented by one or more TF primitives. To create the final semantic parameters the set of vectors containing the TF information for each reference data is analyzed by using principal component analysis.

The above mentioned transfer function applications are restricted to two dimensions, but there also exist approaches trying to provide an interface to handle an almost arbitrary number of dimensions. Bordignon et al. [2006] for example, use the concept of star coordinates first proposed by Kandogan [2001] for user interface design to realize an n-dimensional transfer function by painting and scaling the axis on a two dimensional circle. A completely interactive multi-dimensional classification approach is proposed by Tzeng et al. [2003] allowing the user to define the regions of interest by painting directly on sample slices of the volume. The user applies different colors to estimate every part of the volume she or he wants to extract, and another color to disable the regions she does not want to see. This information is used by artificial neural networks which uses the painted data as training data, where the user can iteratively refine the results.

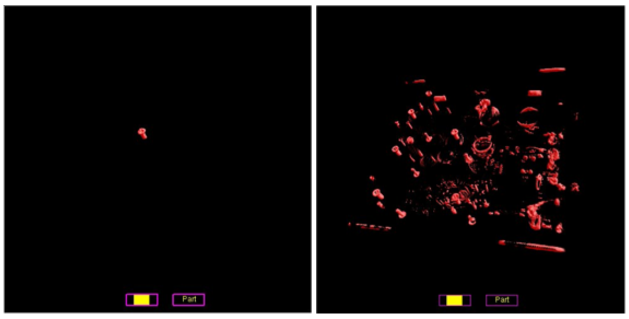
2.5 3D Visualization of Industrial CT Data

Non-destructive testing of objects is a key aspect of present day engineering and development, which leaves the structural integrity of the object being tested intact. Items such as engines, minerals, and other construction and industrial pieces, require critical inspection before they are assembled into a finished product in order to ensure safety, stability, and usefulness of the final object. Although it is extremely important to know the status and structural integrity of elements before they are assembled into a finished product, research towards visualization and quantification of industrial CT data is mainly restricted to business applications. A widely-used NDT application is VGStudio Max [2008]. The primary objective of this application is to process the huge amount of data resulting from CT scans in reasonable amounts of time, described in Reinhard et al. [2004]. It offers a broad range of different functionalities, however, with suboptimal volume rendering qualities.

Huang et al. [2003] have developed a set of visualization techniques for feature extraction and modeling, generated from high resolution industrial CT data for NDT. This semi-automatic approach is based on a region growing algorithm which uses the extracted features to mark other features with the same properties in a 2D transfer function. So the main idea is that full region growing is generally not needed. The user has to place a few seeds into a representative region. These seeds grow the region as long as the defined cost functions are satisfied. Additionally, information about the seed voxels and its surrounding voxels are presented in the transfer function. When a voxel is selected, it is highlighted together with its 26 neighbors. So the corresponding density value (x-axis) and gradient magnitude (y-axis) can be easily identified. A red rectangle shows the standard deviation of the density values and the gradient magnitude as shown in Figure 2.9(a). The blue box shows the standard deviation of the density values and gradient magnitudes. If there are multiple regions of interest to visualize, a 2D transfer function can be derived based on the partial results of the feature extraction. In Figure 2.9(b; left) a screw is extracted using seeded region growing and the derived transfer function is then



(a) Additional information displayed in the transfer function.



(b) Region growing results.

Figure 2.9: *(a) Additional information is displayed on the 2D histogram to facilitate seed selection. (b) The screw extracted using partial region growing (left) is used to derive a transfer function for capturing the other screws (right). Images by courtesy of Huang et al. [2003].*

applied to the whole volume (Figure 2.9(b; right)).

Other techniques like in Heinzl et al. [2007] use the different X-Ray modalities from a dual energy CT to extract different features of high density materials within the case of low density materials. With this approach features of different densities can be extracted and afterwards registered to each other to finally construct a surface model of the fused dataset.

Chapter 3

Pre-Computation

Although exploration seems to be the conceptually most important part of the pipeline, the basis for interactivity during exploration is a complex pre-computation stage, whose components are described in this chapter. Just minimal user-input is required for this stage. Still, it is technically complex and time consuming but completely decoupled from the exploration. The main goal for pre-computing additional information is to enable exploration of different classes of features with different parameters in such a way that, e.g., the main parameter used to steer region growing (e.g., maximum variance) can be changed interactively after actual region growing has been performed. In order to allow this, region growing is performed in multiple passes and the progress is tracked for each voxel, which is recorded in *feature-size curves*.

3.1 Parameter-Settings

To allow exploration over the parameter space, a few parameters (which are used for the pre-computation stage) need to be specified beforehand . This includes the range for the minimum- and maximum feature size described in Section 3.5.2, as well as the range for the minimum- and maximum variance value and the number of desired calculation steps, which together define the step size for each region growing pass, described in Section 3.2. Additionally, the user can select between two different region growing methods which, each on their part, allow different combination possibilities (e.g., with or without boundary growing or the number of neighbor voxels used), described in detail in Section 3.6. To improve segmentation results for some datasets or for performance reasons an additional windowing function can be used described in Section 3.5.1.

These parameter settings are cached together with the data emerging from the region growing process described below, in order to reconstruct the different stages without recalculation.

3.2 Feature-Size Curves

In order to allow interactive exploration of region growing with different parameter settings, which would not be possible to change interactively (e.g., different variance thresholds), the result of region growing is tracked and recorded along the parameter axis in the pre-computation stage. That is, instead of using a single parameter value, we track features over an entire *parameter range*. In order to make the following description more general, we denote this parameter as "time" t, which is stepped from a start time t_0 to a maximum time t_{max} in a specified number of steps b. In this work parameter t is used for the entire variance range, from the minimum interesting variance t_0 to the maximum interesting variance t_{max}. So, t tracks the region growing parameter ε or k depending on the method used, described in detail in Section 3.6, and is increased progressively by passing from one iteration step to the next. That is, the time (parameter) axis is sampled into b bins, e.g., $b = 256$, which allows the tracked curves to be stored in arrays with b entries, resulting in the third dimension of the 3D TF described in Section 4.1.2. For each voxel, the size of the feature (region) it belongs to is recorded, resulting in a *feature-size curve* for each voxel $\mathbf{x}$ along the time axis t: $f_s(\mathbf{x},t)$.

Figure 3.1 illustrates the scheme of emerging feature size curves on the basis of four different example voxels. Voxel 0 (red) is the seed voxel of a feature that starts at time t_0 with a size of 30 voxels and grows in size several times as t increases. Voxel 3 (purple) is another voxel in this feature, but only becomes a part of it at time t_2. This happens when the actual variance parameter is compatible with the density value for this voxel. Voxel 2 (green) emerges at time step t_1 and belongs to a separate feature, growing several times before it merges with the feature containing

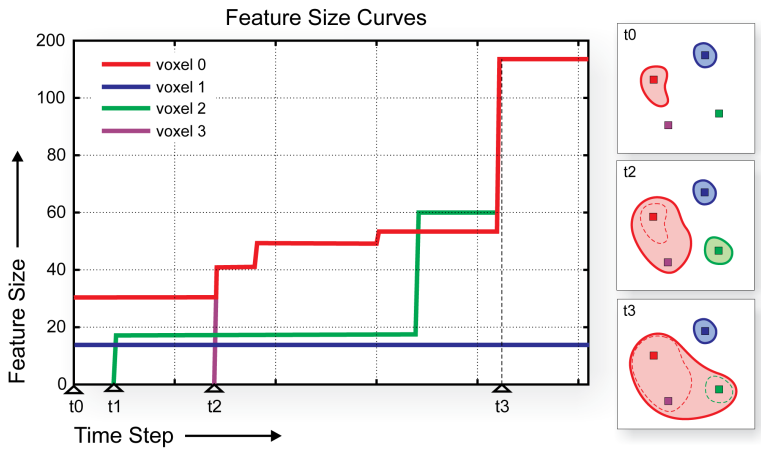


Figure 3.1: *Feature size curves of four example voxels. Seed voxels belong to a feature from the timestep where it is created (voxels 0 and 1; t_0 and voxel 2; t_1), whereas other voxels may join a feature at a later time (voxel 3; t_2). Features may merge over time (t_3), which implies that the feature size curves of all contained voxels are the same after the merge (voxels 0, 2, and 3) (Hadwiger et al. [2008]).*

voxels 0 and 3 at time t_3. Whenever a voxel joins a feature, or two features merge, their curves are identical from then on, which allows to store them very efficiently. Voxel 1 (blue) belongs to a feature with a size of 15 voxels, which stays at this size over time, i.e., does not grow any further after the first time step. This often occurs in case of air or gas inclusions within iron or similar dense material. In this case the difference between the inclusion and the environmental material is big enough to ensure a stable boundary which in most cases stops region growing after the first iteration.

Storing Feature-Size Curves:

The major observation for storing feature size curves $f_s(\mathbf{x},t)$ with a manageable amount of memory is that all voxels in a feature exhibit the same curve from the time, at which they have joined it.

Most of the features start growing at the beginning (time t_0), and may possibly grow in further time steps. For the voxels that do not belong to any feature $f_s(\mathbf{x},t_0) = 0$ applies. Seed voxels that start a new feature at time t_i record the entire growth curve of this feature. In each pass, additional voxels may become a part of this feature, and when a voxel does so at time t_j, it from then on shares the feature-size curve with the curve of the original seed voxel. Before a voxel becomes part of a feature,

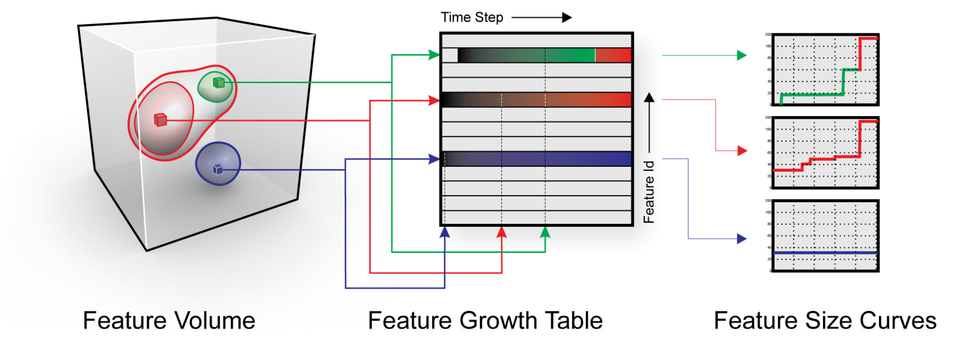


Figure 3.2: *A feature volume stores only the per-voxel information that is necessary in order to reconstruct the feature-size curve of each voxel using the per-feature growth information stored in the feature growth table, where each row corresponds to one feature (Hadwiger et al. [2008]).*

its “voxel-local” feature size is zero. That is:

$$f_s(\mathbf{x},t) = \begin{cases} 0 & t < t_j \\ f_s(\mathbf{x}_s,t) & t \geq t_j \end{cases} \tag{3.1}$$

for a voxel $\mathbf{x}$ that at time t_j becomes a part of a feature whose original seed voxel is $\mathbf{x}_s$. This fact makes it possible to store only a single feature-size curve *per feature* in full. However, for each voxel, the feature ID that it will become a part of at time t_j must be stored, as well as storing t_j itself. This *per voxel* information is stored in a 3D *feature volume* (Section 3.3), whereas the curves themselves are stored in a 2D *feature-growth table* (Section 3.4). Figure 3.2 illustrates the relationship between the feature volume, the feature-growth table, and feature-size curves, which is described in detail below. The ID_{birth} from the feature volume determines the appropriate row in the feature-growth table where the whole feature curve is stored. To get the characteristics of this feature at a specific moment j in time, t_j is used to index the column. There the current feature ID (which may change over time because of merging) and the current feature size at time j are stored. Consequently the whole curve for that voxel can be found in row ID_{birth} from column t_{birth} until t_{max}.

3.3 Feature Volume

The purpose of the feature volume is to store all per-voxel information that is needed to reconstruct full feature-size curves at run-time. As explained in the previous chapter, it is sufficient to store only two values per voxel $\mathbf{x}$: $(ID_{birth}(\mathbf{x}), t_{birth}(\mathbf{x}))$. The first value yields a feature ID that this voxel belongs to. However, when features merge over time, feature IDs can change. In order to avoid storing these changing IDs per voxel, the ID stored in the feature volume is the *feature birth ID* (ID_{birth}), which is the ID of the first feature this voxel belongs to. The actual ID of the voxel at a specific time step can be discovered with a simple lookup in the feature growth table described below 3.4. The second value determines the time at which this voxel becomes part of the feature with the corresponding feature birth ID, i.e., its *feature birth time* (t_{birth}).

Storing the Feature Volume:

The feature volume is stored in a 16-bit two-channel 3D texture, e.g., a Luminance-Alpha texture in OpenGL [2005]. During rendering, a single texture fetch from this 3D feature texture yields everything needed to reconstruct the voxel's feature-size curve via the feature-growth table stored in a 2D texture, as described below.

3.4 Feature-Growth Table

As described in the previous chapters, the feature volume itself does not store actual feature-size curves. Instead, we store one representative curve for each feature, which is the feature-size curve of the feature's seed voxel $\mathbf{x}_s$, in a 2D *feature-growth table*. The feature-growth table contains one row per feature and b columns, where b is the number of bins for sampling the parameter t. This table is indexed with the feature birth ID and feature-birth time, obtained from the lookup in the feature volume, as row and column indexes, respectively. Each entry in this table is a $(f_s(\mathbf{x}_s, t), ID(t))$ pair, which collectively represents the sampled growth curve of the feature over time t: $f_s(\mathbf{x}_s, t)$, as well as mapping time to *current feature ID*: $ID(t)$. The latter is necessary in order to be able to handle the merging of features over time, where feature birth ID and current feature ID are not necessarily the same, because the ID can change when a feature merges into another one. The handling of merging features is described below. The feature-size curve for a given voxel $\mathbf{x}$ can be reconstructed for any time t_i by using Equation 3.1 with $t_j = t_{birth}(\mathbf{x})$, and indexing the feature growth table in row $ID_{birth}(\mathbf{x})$ and the column corresponding to the bin of t_i, to obtain $f_s(\mathbf{x}_s, t_i)$ when $t_i \geq t_j$. This is a very simple and efficient

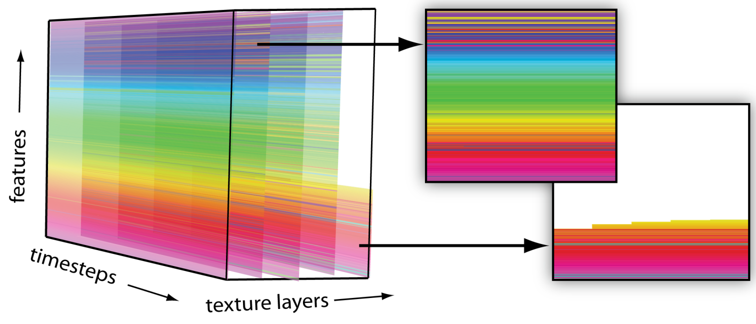


Figure 3.3: *The feature-growth table is stored in a 2D texture array where the different features are color coded for visualization. The number of texture layers (z-axis) depends on the number of features and the maximum allowed dimension of the 2D texture (y-axis). The x-axis is the number of time steps. If a feature changes its ID (because of merging) the line is continued with another color that corresponds to its new ID.*

scheme, which can be evaluated in a GPU fragment shader during ray-casting, illustrated by the pseudo code in Section 4.3.

3.4.1 Storing the Feature-Growth Table

In principle, the feature growth table is stored in a 16-bit two-channel 2D texture, e.g., a Luminance-Alpha texture in OpenGL. However, this texture has to contain one row per feature and the number of features can reach into the thousands. Therefore, hardware texture dimension constraints make it impossible in the majority of cases to use a single 2D texture for this purpose. If the hardware supports 2D texture arrays (e.g., NVIDIA GeForce 8800), we split up the feature growth table into a texture array with tex_{max} rows and $\lceil n_{features}/tex_{max} \rceil$ layers, where tex_{max} is the maximum allowed dimension of a 2D texture as reported by the hardware, and $n_{features}$ is the number of features. Figure 3.3 illustrates a feature growth table with five layers. The last layer is mostly never entirely used. If texture arrays are not supported, the feature growth table is split up in a similar way, but layers are stored in the depth dimension of a 3D texture, which is functionally almost identical. However, accessing a 3D texture is slower than accessing a 2D texture array, and it might be necessary to pad the depth dimension to a power-of-two.

3.4.2 Merging Features

Features can merge as the parameter t increases, e.g., when two small disjoint but close features grow in size over t until they finally touch while conforming to the homogeneity criterion. When this happens at time t_k, they are treated as a single feature for all $t \geq t_k$. In order to do so, we assign a new *current feature ID* to the entire merged feature, using the numerically smallest ID of the features that have merged as shown in Figure 3.4.

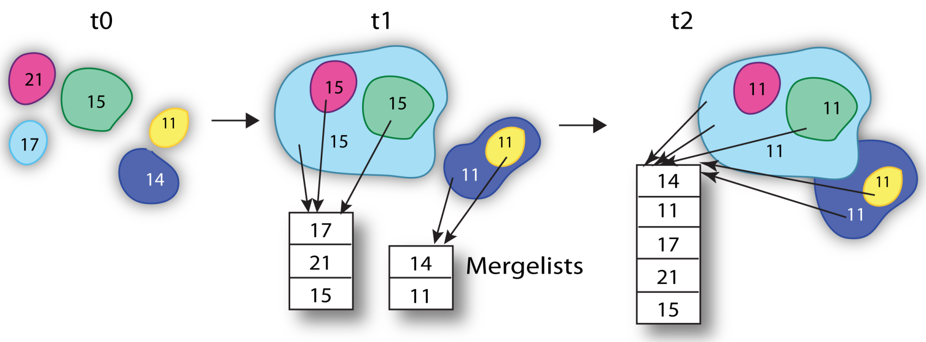


Figure 3.4: *Merged features are all assigned the smallest ID of all features included in the current merge operation. At the beginning all features hold their own ID (t_0). When some of them merge at a later timestep (t_1), they are treated as one feature with the numerically smallest ID from that time on . For each feature a mergelist exists, holding all IDs of the features involved in the new merged feature. When compound features (i.e., features that have been combined already at an earlier timestep) merge (t_2), also the corresponding mergelists need to be merged.*

Additionally a *mergelist* is tracked for each feature containing all feature IDs of which the actual feature is composed. A mergelist is attached to a feature when it merges for the first time and updated after each merge operation. For two features that merge also the mergelists have to be combined as illustrated in Figure 3.4. Tracking a mergelist for each feature is necessary to calculate the statistical feature properties described in Section 3.7. During rendering, the feature ID that is current at time t_i needs to be determined for a given voxel (sample). Thus, we also store the feature ID corresponding to each t_i ($ID(t_i)$) in the feature growth table, as explained above. Every row in this table corresponds to a *feature birth ID*, i.e., the ID that was assigned on feature creation, whereas the *current feature ID* is retrieved from the ID entry in column t_i. Thus, the IDs stored in the feature volume are feature birth IDs instead of current IDs. The big advantage of this approach is that it allows the IDs stored in the feature volume to be left untouched by merging of features. It is

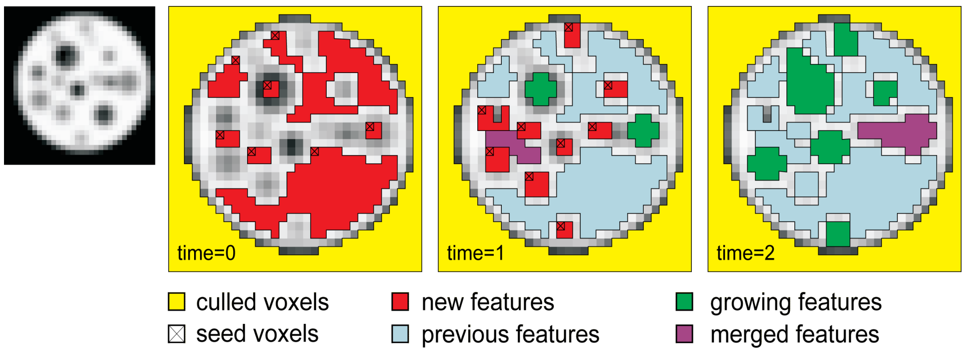


Figure 3.5: *Region growing is performed in multiple passes. In each pass, new features can be created, previously existing features may grow (except in the first pass), and features may merge. Optionally, background voxels can be culled in order to improve pre-processing performance (Hadwiger et al. [2008]).*

sufficient to know the feature birth ID for each voxel, i.e., the feature the voxel first belonged to. Everything else can be obtained from the feature growth table, i.e., for any t_i, the current ID and current size of the feature are retrieved from the feature growth table in row *feature birth ID* and column t_i.

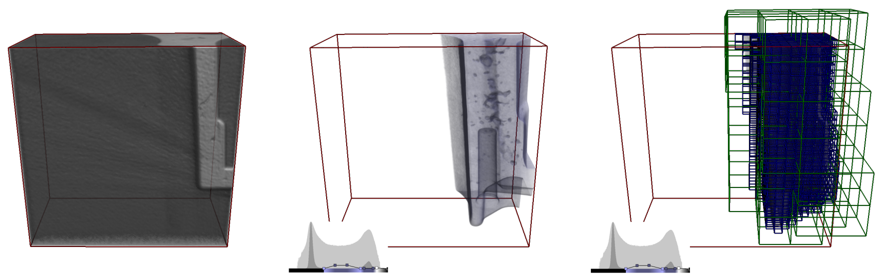
3.5 Multi-pass Region Growing and Seed Selection

This chapter describes how both the feature volume and the feature growth table are computed using region growing in multiple passes, where each pass corresponds to a specific time step t_i. Figure 3.5 illustrates the overall region growing process for three consecutive time steps. The resulting feature-size curves over time and the feature volume and feature growth table used to store them are illustrated in Figures 3.1 and 3.2, respectively.

Instead of selecting specific seed voxels either manually or automatically, we consider *every* voxel in the volume as potential seed for growing a region. This way, no seeds for potential features can be missed, and the user is not required to specify seeds at all. Nevertheless, we allow optionally culling away the background or other undesired regions as performance optimization.

3.5.1 Culling

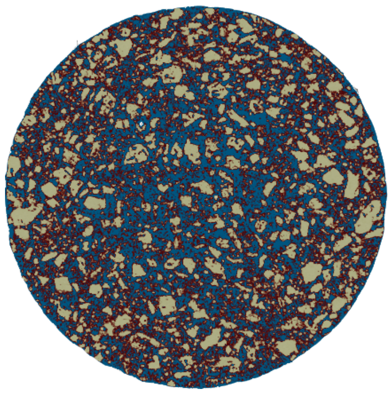
In industrial CT parts a significant number of voxels are usually part of the background, i.e., air. In order to speed up the pre-computation stage, it is worthwhile to remove these parts of the volume from the potential seed candidates. This is done by allowing the user to specify a 1D opacity transfer function, which is used to cull small sub-blocks of the volume. Figure 3.6 shows an example with a part of a cast housing. The simplest transfer function for culling is a simple windowing function. However, the default setting is a window covering the entire density range, which disables culling. This is shown in Figure 3.6(a) where the whole dataset, displayed with DVR, is enclosed by a red rectangle. In this case obviously more than half of the dataset consists of air, which can be excluded from rendering by using a simple 1D transfer function (Figure 3.6(b)). Therefore, also the region growing process is limited to either the green (32^3) or the blue (8^3) sub-blocks shown in Figure 3.6(c). When the analysis of just a part of a component is desired, clipping planes can be used to cull sub-blocks along the x-, y-, and/or z-axis respectively. An example for such a region of interest (ROI) is shown in Figure 4.8, where region growing is performed only for the remaining part of the volume.



(a) The whole dataset. (b) Using a 1D opacity TF. (c) Sub-blocks.

Figure 3.6: *Culling is used to exclude unwanted parts, i.e., air from region growing. (a) Shaded DVR of the whole dataset including the surrounding air. (b) Shaded DVR using a 1D opacity transfer function (bottom left) which excludes the air from rendering. (c) Only the green(32^3) or the blue (8^3) sub-blocks are used for region growing respectively.*

An additional method allows not only to disable whole blocks of the volume but also groups of voxels according to their density. Therefore the 1D transfer function is used as a simple windowing function to *"turn off"* voxels by setting their opacity to zero. This can be useful to extract features of a specific density if there are many different structures of different materials, like in minerals. Figure 3.7 presents a typical case where such an application might be useful. Here the windowing function (Figure 3.8) is used to extract just the phases of different sizes from the surrounding sand material in the industrial ceramics. The black areas inside the data in Figure 3.8(a) are finally used as seeds for region growing. The results (Figure 3.9(a)) are distinguished using the 3D transfer function shown in Figure 3.9(b) which is described in detail in Section 4.1.2.


(a) Slice view of an industrial ceramic.

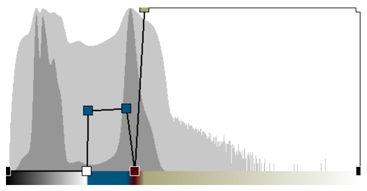
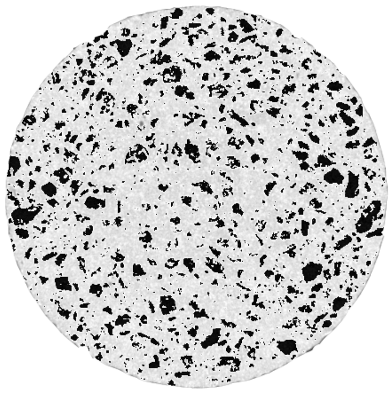

(b) A 1D opacity transfer function for material visualization.

Figure 3.7: *An industrial ceramic (a), using a 1D opacity transfer function to get an impression of different materials in the dataset (b).*


(a) Slice view of an industrial ceramic.

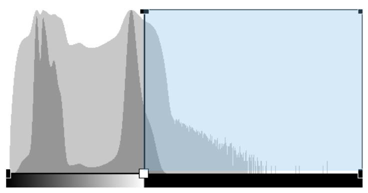
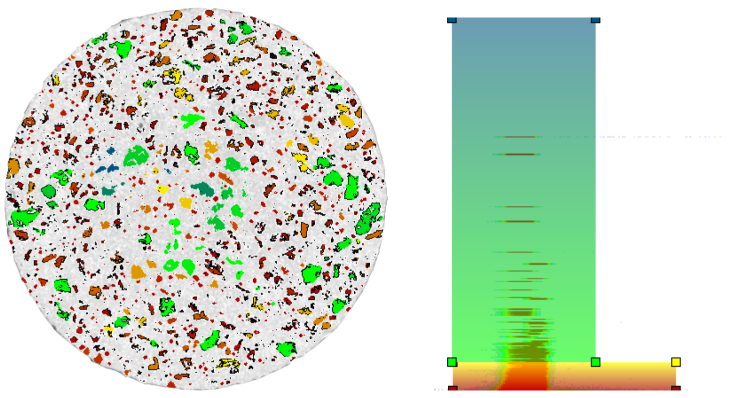

(b) The windowing Function.

Figure 3.8: *An industrial ceramic (a), where with the windowing function in (b) the opacity of the sand material is set to zero and therefore excluded from region growing. This means, only the voxels having density values in the light blue part of the TF are used as seeds for region growing.*

Generally, culling determines an active block list, and the voxels contained in active blocks are considered as potential seeds. If additionally the option for the alpha



(a) Slice view of an industrial ceramic. (b) The resulting 3D TF is used to extract different phase features.

Figure 3.9: *After region growing the phase features can be explored using the 3D TF (density, featuresize, time) shown in (b).*

values is used, the density value of the seed is checked against the transfer function and only further processed if alpha is greater than zero.

3.5.2 Region Growing

Given the active block list determined by culling (which might contain the entire volume), selection of seed candidates for region growing proceeds by processing block after block, considering each contained voxel in turn and optionally checking against the 1D transfer function, as already described in the previous Section 3.5.1.

In each pass, for each seed candidate, a region is grown as far as possible given the current parameter t_i. The pseudo code in Figure 3.10 gives an overview of the different stages in the growing process which are described in detail below.

In order for a region to become a feature, its size must be both larger than a given minimum (*min_featuresize*) and smaller than a given maximum (*max_featuresize*). These restrictions are added to avoid spurious features of only a few voxels due to noise, and turning entire structures such as intended holes, in e.g. a casting part, into features. If a region satisfies these two criteria, a new feature is created from it. All voxels comprising this feature will subsequently not be considered as seed candidates. Region growing then continues by considering the next seed candidate. Furthermore, in all passes after the first one, in addition to starting completely new

```
generateFeatureSizeVolumeAndHistogram()
{
 clear feature volume and checked mask;
 grow the encapsulating air;

 for each timestep t_i {

   clear visited mask;
   calculate variance value depending on t_i;

   first pass:  grow homogeneous areas;
   second pass: grow boundaries;
   calculate geometrical properties (surface,...);
   calculate histogram;
 }
}
```

Figure 3.10: *Pseudo code for the whole region growing process.*

features, existing features have to be grown further if allowed by the increased region growing parameter t_{i+1}. The distinction between first and subsequent passes, as well as handling multi-pass region growing efficiently, is described below.

Before growing of the features starts, the air which corresponds to the dataset and encloses the scanned component is grown. This is necessary for the statistical calculations described in the next chapter. In the first pass, $i = 0$, with a corresponding region growing parameter $t_i = t_0$, every voxel that is not yet part of any feature is considered as seed candidate for starting a completely new feature. The pseudo code in Figure 3.11 demonstrates the process of how a potential seed voxel may grow a feature. Note that since seed candidates are considered sequentially, even in the first pass many voxels may already have been assigned to features when a given candidate is processed. In the next pass, $i+1$, and all subsequent passes, there are three possibilities for a potential seed-voxel candidate:

1. Unassigned voxels are considered as new seed candidates, possibly starting a new feature at time t_{i+1}.
2. Already existing features are grown further from their boundary voxels, if possible, given the new region growing parameter t_{i+1}. If an existing feature grows further at time t_{i+1}, its feature size increases, i.e., $f_s(\mathbf{x}, t_{i+1}) > f_s(\mathbf{x}, t_i)$, see Figures 3.1 and 3.2.

3. Already existing features are grown further from their boundary voxels and merge with other features as already described in Section 3.4.2.

```
growNewRegion()
{
 push seedvoxel into region_queue;

 while region_queue is not empty {

    for the six neighbor voxels {

      check if valid position in volume;
      check against visited- and checked mask;
      optional: check against alpha value in 1D TF;
      check the homogeneity criterion;

      //when voxel is still valid...
      if region < MAX_FEATURESIZE {
        push voxel into region_queue;
        tag visited mask;
        add voxel to region;
        add ID and t_i to feature volume;
      }
      else {
        mark voxel as TOOBIG in the feature volume;
        tag checked mask;
      }
    }
  }
  if new feature > MIN_FEATURESIZE
     add to feature growth table;
}
```

Figure 3.11: *Pseudo code for growing a homogeneous region from a new seed voxel.*

This is illustrated in the pseudo code in Figure 3.12. The following approach allows handling all cases efficiently and treating the first and all subsequent passes as uniformly as possible.

We maintain two bit masks with one bit per voxel for tagging and thus removing voxels from further consideration for either growing a new feature or extending an existing feature. A voxel is considered tagged when its bit is set in either one of the two masks (or both). The goal of the *checked mask* is to remove voxels from consideration for growing a new feature (where it would be a seed voxel), or extending an existing feature (where it would be a voxel in the feature's boundary), both in the current and all subsequent passes. This mask is cleared only before the first pass and then updated from pass to pass. A bit in the checked mask is set when either:

a) the voxel is *inside* a feature, i.e., all its neighboring voxels also belong to the feature (e.g., using the 26-neighborhood); or

b) because the voxel is part of a region that became bigger than the *max_featuresize*.

```
growAndMergeFeatures()
{
 push actual voxel into queue;

 while queue is not empty {

    for the six neighbor voxels {

      check if valid position in volume;
      check against visited- and checked mask;

      //when voxel is still valid there are three
      //possibilities:
      //FIRST POSSIBILITY:
      if voxel belongs not yet to a region {
        do all steps for growing a new region;
      }
      else if voxel belongs already to a region {

        //SECOND POSSIBILITY:
        if voxel belongs to current region {
          check for completely enclosed voxels to tag bit
          in checked mask;
        }
        //THIRD POSSIBILITY:
        else voxel belongs to another feature {
          check the homogeneity criterion for both regions;
          merge the mergelists;
          merge the regions;
          update feature growth table for current timestep;
        }
      }
      tag visited mask;
    }
  }
}
```

Figure 3.12: *Pseudo code for the second and all subsequent region growing passes.*

In contrast, the goal of the *visited mask* is to avoid considering the same voxel twice in the same region growing pass (for either growing a new feature or extending an existing feature). As such, it is cleared before each pass. A bit in the visited mask is set when:

a) a voxel is added to a feature in the current pass, irrespective of whether to a new or already existing feature; or

b) it has already been a seed candidate in this pass, irrespective of whether a feature has actually been grown from it or not.

During region growing, a voxel is a candidate for either growing a new feature or growing an already existing feature further when neither its bit in the checked mask nor in the visited mask is set. The distinction between these two cases is done according to the corresponding entry in the feature volume. This entry either contains the birth feature ID of the voxel if it already belongs to a feature (i.e., it is a candidate for growing the feature further), or an invalid ID and thus does not yet belong to a feature (i.e., it is a seed candidate for growing a completely new feature). Whereas setting a bit in the visited mask excludes a voxel just for the current growing pass, setting a bit in the checked mask excludes a voxel from the entire region growing process until the end. Therefore, the voxels checked in the visited mask have the chance to grow further in a following pass.

In addition to the feature volume and feature growth table, which are converted to textures for rendering, further information is stored for quantification and statistical views which are described in Section 3.7 below.

3.6 Region Growing Criteria

Our approach is mostly independent of the actual region growing method that is used, and works well as long as a single parameter t suffices to characterize the main variation of the growing process. The two region growing methods outlined below are used as a proof-of-concept for this interactive approach. However, other region growing or feature detection methods could be adapted as well to work in the context of this framework.

3.6.1 Region growing method A

This is a variant of seeded region growing described in Adams and Bischof [1994] that is also able to include a region's boundary. Region growing is performed in two distinct phases:

1. *Grow the homogeneous "core" of a region.* A voxel is added to the region when the difference of its density to the average density of the whole region is below a given threshold ε:

   $$|v - v_r| < \varepsilon,$$

   where v is a voxel's density value, and v_r is the current region's average density. After a new voxel is added, v_r is updated accordingly.

2. *Expand the region by including its boundary.* For every voxel adjacent to the region, we either check a gradient magnitude criterion in order to decide whether this voxel should be included in the region, or not. This step is optional but enabled by default.

The time parameter t determines homogeneous growing: $t = \varepsilon$. Further, it is possible to select whether the feature should grow with the boundary or without.

3.6.2 Region growing method B

Huang et al. [2003] are using a combination of region growing based on density variance and gradient magnitude standard deviation. They determine both variances for a fixed neighborhood of a seed voxel. The main parameter is a scale factor $k > 0$:

$$f_{ca} = \frac{|v - v_s|}{k\sigma_v}, \tag{3.2}$$

$$f_{cb} = \frac{|g - g_s|}{k\sigma_g}, \tag{3.3}$$

$$f_{cc}(p) = p f_{ca} + (1-p) f_{cb}, \tag{3.4}$$

where v is a voxel's density value, v_s the density of the seed, g a voxel's gradient magnitude value, g_s the gradient magnitude of the seed, and σ_v and σ_g the corresponding variances in the seed neighborhood, respectively. The size of the neighborhood can be selected between six, eighteen or twenty-six voxels. The factor p can be set to a constant value but is set by default to $p = \frac{\sigma_g}{\sigma_v + \sigma_g}$. We employ these region growing criteria in our framework by setting $t = k$ for tracking the main parameter k. The possibilities to apply these equations are the following:

1. Only f_{ca} to just grow the homogeneous core of a region,
2. f_{ca} in combination with f_{cb} such that f_{cb} is additionally used for boundary growing and
3. f_{cc} where also a combination of the other two methods is used together with the weighting parameter p.

3.7 Statistical Feature Properties

The calculation of the following statistical properties have to be executed after each growing pass because features may change over time and therefore also the statistical parameters have to be adapted.

The columns within the statistics table are:

- **Feature ID:** The ID of the feature which indicates the growing order and merging events.
- **Volume in voxels:** The volume of the feature in voxels.
- **Volume in mm^3:** The volume of the feature in mm^3 considering the resolution of the dataset. The size of the feature voxels in percent to the whole volume of the component (excluding the air surrounding the object), can be estimated using the 2.5D widgets explained in Section 4.1.2.
- **x, y, z:** This is the position of the original seed-voxel from where the region starts its propagation, shown in Figure 3.13(a). This entry does not change over time.

- **Surface:** The number of voxels on the surface of the feature.
- **sx, sy, sz:** The size of the smallest possible bounding box encapsulating the whole feature, as shown in Figure 3.13(b).
- **px, py, pz:** The surface area of the feature's shape projected onto the x-, y- and z-plane, illustrated in Figure 3.13(c).

All values (except the position of the seed voxel) are constrained to the currently given timestep which is adjustable in the 3D TF as explained in Section 4.1.2. Storing the minimum and maximum density of a feature allows efficient highlighting of the picked feature in the transfer function domain.

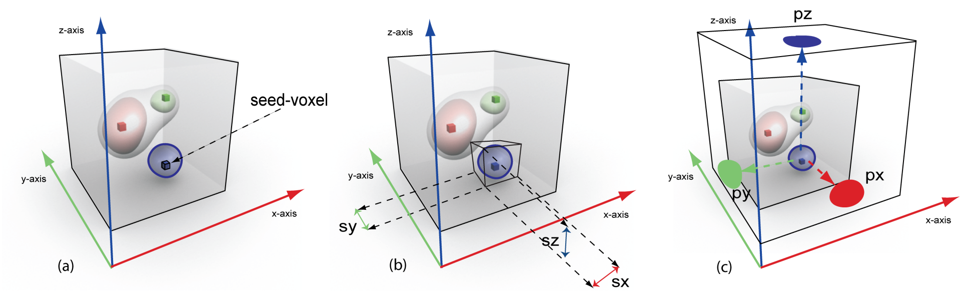


Figure 3.13: *(a) The x-, y-, and z position of the seed-voxel in the data volume. (b) The dimensions of the smallest possible bounding box enclosing the feature. (c) The surface area of the feature's shape projected onto the three planes of the volume.*

Chapter 4

Exploration

This chapter describes the feature exploration stage of the pipeline. For the user, this stage is the most important and visible one. The complexities from the previous pre-computation stage are hidden to a large extent. In order to explore and classify features and feature classes, the corresponding regions in the volume can be mapped to color and opacity using one of two different means:

1. via a 3D transfer function in the *(density, feature_size, time)* domain, described in Section 4.1 (Figure 4.2(a)), which maps entire feature classes; or

2. directly via picking individual features in either one of three orthogonal slice views, in the *feature table*, (Figure 4.9(a)) or in the *graphical feature view* (Figure 4.6), both described in Section 4.2.

The graphical feature view allows the user to pick features, inspect their feature size curves, and set their color and opacity individually. It can also be used for disabling features that stem from CT artifacts by setting their opacity to zero. Also some attributes like ID, size and picked time step are shown immediately. When a location is picked in a slice view, the corresponding feature (if any) is also selected in the graphical feature view. During exploration, the current classification is shown in real-time in a 3D volume view (Section 4.3; Figure 4.6)) and three slice views (Figure 4.7). Picked features are immediately highlighted in all of these views.

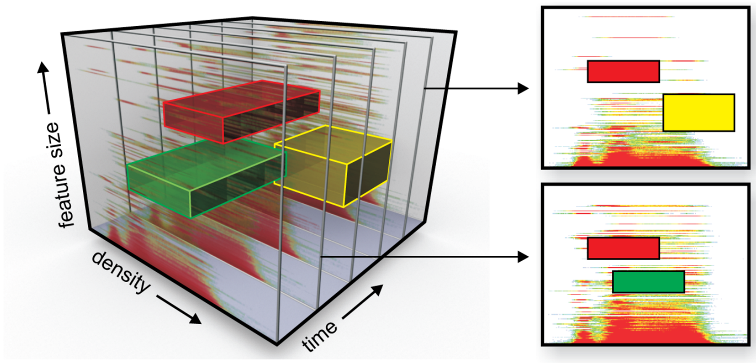
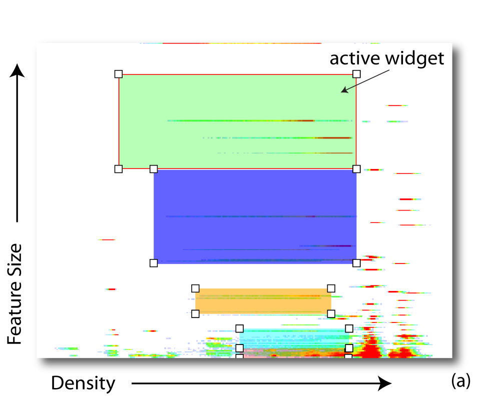


Figure 4.1: *Transfer function with 2.5D widgets in the 3D* (density, feature_size, time) *domain. A stack of 2D* (density, feature_size) *histograms, one for each time step, helps with transfer function specification. The widgets are valid over a user-specified timerange (Hadwiger et al. [2008]).*

An important concept during exploration is the handling of the time axis t. Showing all time steps simultaneously is only supported by the graphical feature view. It allows to inspect feature size curves along the time axis in a function plot, where the dense color-coding of feature IDs shows their evolution over time (due to creation and merging of features). The feature size curves constitute the main part of the view (Figure 4.5(b)). All other views, i.e., the 3D volume view, the three orthogonal 2D slice views, the feature tables and also the transfer function panel depict only a single time step. This *current time step* t_{cur} is specified globally for exploration and can be modified by the user at any time via a simple slider.

4.1 Exploring Feature Classes

The main goal of classification is to explore feature classes, instead of requiring the user to inspect individual features. Features are classified by specifying a transfer function in the 3D domain of *density* (from the CT volume), *feature_size* (retrieved from feature size curves, described above in Section 3.2), and *time* (the changes of features according to the main region growing parameter). Although this is a 3D domain, we use a 2.5D metaphor to make the manual specification of transfer functions manageable. The 2D *(density, feature_size)* subdomain can be viewed in its entirety for any given time t_{cur} in the transfer function panel. This corresponds to choosing a specific time of interest via a slider and then exploring



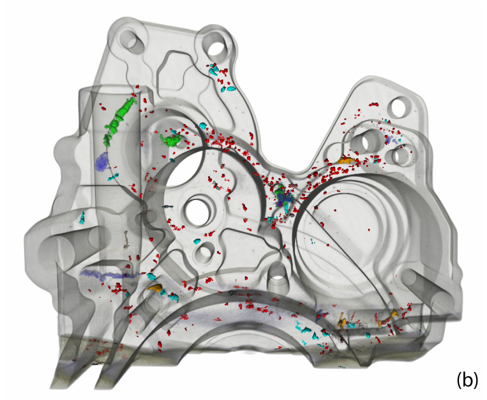


Figure 4.2: *Apart from inspecting individual features, the feature classification space can be explored through a stack of 2D histograms spanning the 3D domain of* (density, feature_size, time). *(a) 2D slice through the domain with histogram and transfer function widgets which define the different feature classes (x axis:* density*, y axis:* feature_size*). See also Table 5.9. (b) Volume view generated with GPU-based real-time ray-casting;*

features according to their size and density distribution. Figure 4.1 illustrates the 3D TF domain, highlighting two selected 2D subdomains and the widgets intersecting them.

4.1.1 Feature Histograms

Figure 4.2(a) shows a 2D histogram plotting voxel density (x-axis) against feature size (y-axis) (Figure 4.5(a) shows such a histogram without classification widgets). The number of voxels with a given *(density, feature_size)* combination is color-coded (red corresponding to a large number of voxels). For each time step t_i, a corresponding 2D histogram is computed, which are maintained as a stack of 2D histograms that collectively span the entire 3D domain. In order to gain insight into the distribution of feature sizes, densities, and their occurrence in time, the time axis is explored using the slider for t_{cur}, which specifies the current time of interest. This enables the user to specify the appropriate time step, or even a whole time range for each feature class.

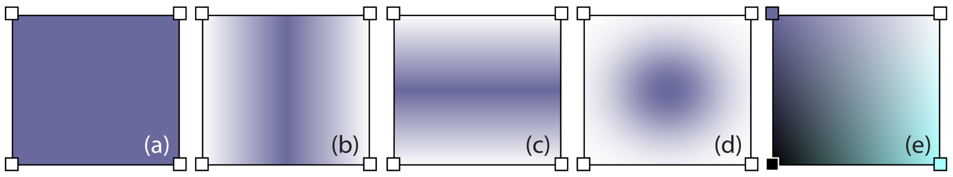


Figure 4.3: *The 2D transfer function widgets are expanded to 3D by specifying a valid time range for each. We use completely solid widgets (a), tent widgets with adjustable peak position and width (b) and (c), gauss blobs (d) and boxes where color and opacity is variable for each corner respectively (e). An application example for widget (e) is shown in Figure 4.6 where it is used to apply a color gradient from small to large features.*

4.1.2 3D Transfer Functions and 2.5D Widgets

The 3D transfer function in the *(density, feature_size, time)* domain is specified using 2.5D widgets, which result from extending some of the well-known regular 2D transfer function widgets. Figure 4.3 illustrates the widgets such as boxes, tents, or Gauss blobs as used in Kniss et al. [2001]. They are expanded into the third dimension by assigning a time range $[t_a, t_b]$ to each widget using a range slider. This range determines for which time steps this widget is active, i.e., 2D widgets are extruded into 3D from time t_a to time t_b. The actual widget shape is 2D, e.g., a Gauss blob is extruded into a cylindrical shape in 3D. The reason for this is that opacity ramps are very useful in the $(density, size)$ subdomain, but gradually changing the opacity classification over time is not meaningful because the time axis is in fact not continuous (it is not only sampled, but also corresponds to the impact of fixed increments in the main region growing parameter on the evolution of regions, not actual time). Thus, a widget is either fully present at a time t_i with $t_a \leq t_i \leq t_b$, or not present at time t_i at all. In many cases, in order to determine a specific feature class the user explores the time domain until a time step is found that depicts the features of this class well. In this case, widgets for this class are set to be active only in this particular time step, i.e., $t_a = t_b$. Example transfer functions are shown in the result Section 5.2. Widgets can also be used to get information about the selected feature classes. The user can get the following information for an active widget (Figure 4.2(a), marked by a red frame), by clicking on it:

- The feature size range of the widget (extent along the y-axis).
- The density range of the widget (extent along the x-axis).

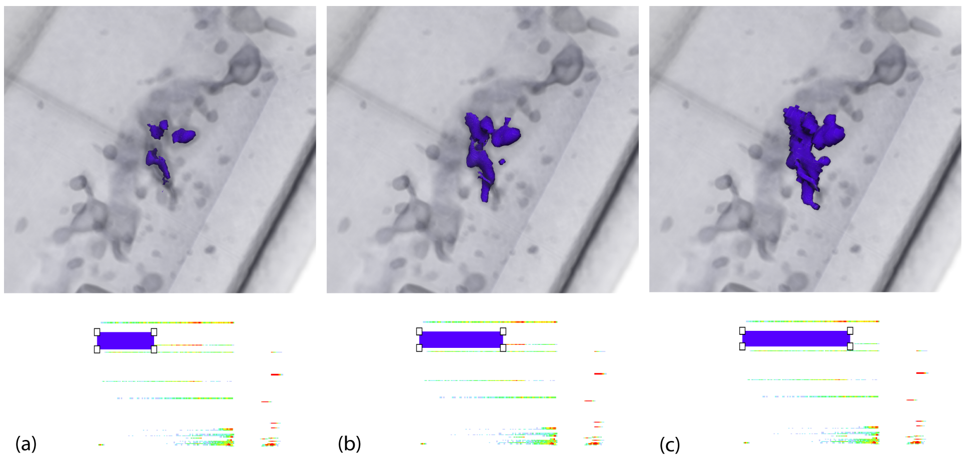


Figure 4.4: *The size of the explored features are adjustable via the length of the widgets. So the growing process of individual features can be retracked by tracking the widget's length from left to right - from the seed-voxel to the whole feature including the boundary. For example, in Figure (a) the widget covers 494 voxels of the feature, in Figure (b) 999 voxels and in Figure (c) 1862 voxels of the same feature at the same time step.*

- The number of features the widget covers.
- The number of feature voxels the widget covers.
- The size of the covered feature voxels in percent with respect to the whole volume of the object (excluding the air surrounding it).

The quantification result tables in Section 5.2 follow from this information.

Also the size of the individual features is adjustable via the TF. Because the x-axis of the histogram depicts the density range of the features, each line in the histogram reaches the voxels from the inner homogeneous part of the feature till the boundary. Therefore, by varying the length of the widget, the size of the features in the according feature class are tunable as shown in Figure 4.4. Therefore, beside the time domain, the user has a second possibility to explore the size of the features.

4.2 Exploring Individual Features

In addition to exploring whole classes of features, it is important to allow the user to also pick and inspect individual features. Features can be picked with the mouse in either:

- one of three orthogonal slice views (Figure 4.7), which retrieves the current feature ID at the picked location;
- in the feature tables according to their ID, where the individual features are listed; or
- in the graphical feature view.

The graphical feature view has two main components, a visualization of all features with their feature IDs color-coded and depicted over time (Figure 4.5(b)), and a plot of the representative feature size curve of the currently picked feature (Figure 4.5(c)). The former visualization contains one row for each feature, with the vertical coordinate corresponding to the feature ID (increasing from bottom to top). The horizontal axis is time t, where horizontal changes in color indicate the merging of features and thus a change in feature ID. The staircase pattern in Figure 3.3 (bottom right) emerges from features that are only created at later time steps, i.e., who have no feature ID before their *feature birth time*. This view shows the feature growth table described in Section 3.4, depicting a color-coding of the $ID(t)$ channel from the $(f_s(\mathbf{x}_s,t), ID(t))$ pairs stored in the table. As such, it is a visual representation of the behavior of all features over time with respect to their creation and merging with other features. When features merge, their IDs change (except for the feature that has the numerically smallest ID of the merging features, which is kept, see Section 3.4). This shows up as color changes within a row in this view. This color-coded view does not allow detailed analysis but provides a

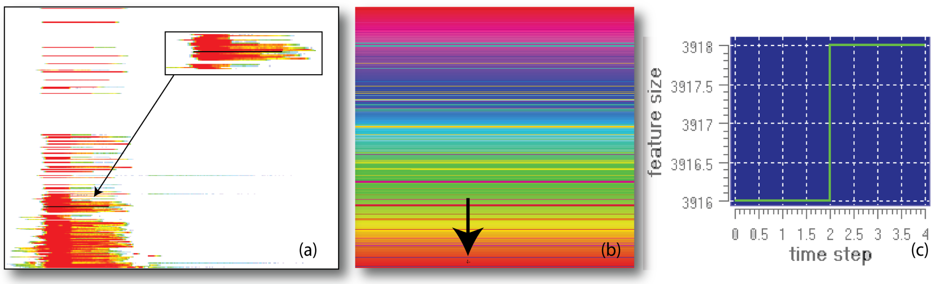
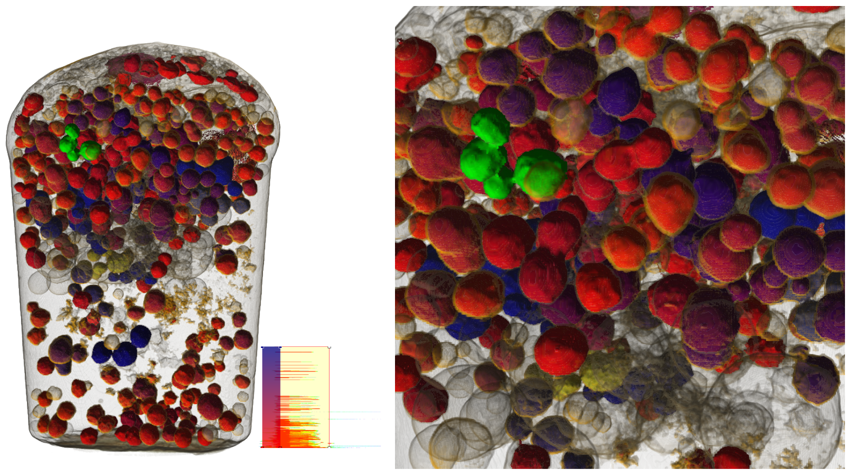


Figure 4.5: *When a position in the graphical feature view (b) is picked, the corresponding feature curve (c) shows up and the location of the picked feature in the histogram of the 2D TF is highlighted at the indicated timestep (a).*



(a) Reduced-Pressure-Test Sample (RPTS). (b) Feature 1547 (green).

Figure 4.6: *Lower densities are the interior of pores (red-blue), higher densities their boundaries (yellow). The feature size is mapped with a color gradient from red (small) to blue (large). Very large features are set to transparent. (a) The picked feature is highlighted in green. For the other features the TF at the bottom right is used. (b) Feature 1547 (green) consists of* 13997 *voxels which are* $17,145mm^3$ *at a resolution of* $0.106996mm$.

good overview at a glance whether a lot of features are merging or not, and at what time steps a lot of merges occur. Detailed inspection is then possible by picking a feature (row), and looking at the corresponding plot that shows all details of the feature's size curve. This curve exactly reveals at which timestep the feature has grown and how much. High leaps in the curve often denote that a merging of two or more features occurred. When a feature is picked in the graphical feature view, the global current time t_{cur} is updated automatically to correspond to the column where the picking has occurred. This concerns the 2D transfer function where the appropriate timestep shows up, as well as the 3D volume view, the three orthogonal 2D slice views, and the feature tables. If enabled, the related location of the feature is highlighted in the histogram of the 2D TF (Figure 4.5(a)) which makes the exploration of similar features easier.

4.2.1 Feature Picking

When a feature is picked, a specific individual color and opacity can be specified, which is then stored per feature by overriding the corresponding entry in the 1D color ramp transfer function described in the next paragraph. Picking in the slice views is implemented by looking at the value in the feature volume and mapping it to a feature ID via the feature growth table. The picked feature is also shown in the transfer function domain (see Figure 4.5(a)). In order to do this efficiently, the minimum and maximum density over all voxels in the feature are recorded for each time step during region growing. Showing the picked feature's footprint in the transfer function domain is very helpful for transfer function specification. Knowing the location of one feature belonging to a specific feature class, helps the user to find features with similar density and size properties. For rendering, the ID of the picked feature is supplied to the corresponding GPU fragment shader, which compares the picked ID with the current feature ID of the sample to be rendered. This is illustrated by the pseudo code in Section 4.3.

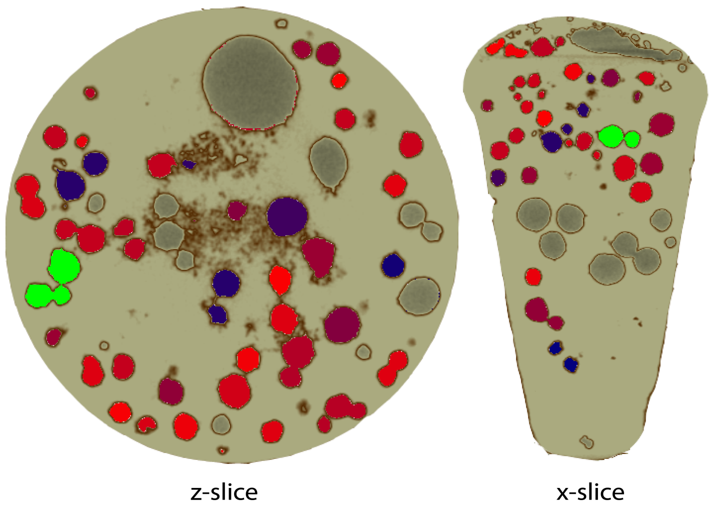


Figure 4.7: *Picking can also be achieved in the slice viewer. The picked feature (green) is highlighted in the same color as in the volume rendering of Figure 4.6.*

4.2.2 Feature Color Coding

All features can automatically be shown in different colors, by mapping feature IDs to colors and opacity via a 1D transfer function, which is filled with a color ramp by default. This is the same color ramp used in the graphical feature view (Figure 4.8 (b) and (f)). Figure 4.8 ((a) and (e)) shows this color mapping applied in the 3D volume view, which is useful to gain a quick overview before transfer functions are specified. Similarly to storing the feature growth table (Section 3.4.1), limitations in hardware texture dimensions often require this 1D table to be stored as several packed rows in a 2D texture.

4.2.3 Removal of Artifacts

Picking features is especially useful for removing erroneous features that in fact are artifacts from the scanning process, such as reconstruction/Feldkamp artifacts or center/circular artifacts. When an artifact is picked, its individual opacity can be set to zero, which removes it from both rendering and quantification. Figure 4.8 demonstrates such a process. Here just a part of the reduced-pressure-test sample (RPTS) is used for demonstration purposes (the clipping planes along the z-direction are shown in blue). After the region growing process, the opacities of single unwanted features are set to zero. Hence a single widget is sufficient to extract all desired features without paying attention to erroneous features. This avoids, that the user has to apply more widgets in the transfer function just to eliminate unwanted features from classification.

4.2.4 Feature Table

The feature tables described here are a mixture between the exploration of whole feature classes and individual features since, even though feature classes may be explored, individual information is provided. They contain all the geometrical attributes described in Section 3.7 for the current timestep t_i and can be written to a *.csv* file if desired. There are two different feature tables:

1. The feature table of all features defined in the graphical feature view.
2. The feature table of the features defined by the feature classes in the 2D transfer function.

For the first table optionally the same color coding as in the graphical feature view is used. However, features may be listed more often because of merging (all features that merge get the same ID from that time on but stay as separate entries in the table). The second feature table allows to get all attributes of the features contained in the feature classes defined via the 2D transfer function (Figure 4.9). The color

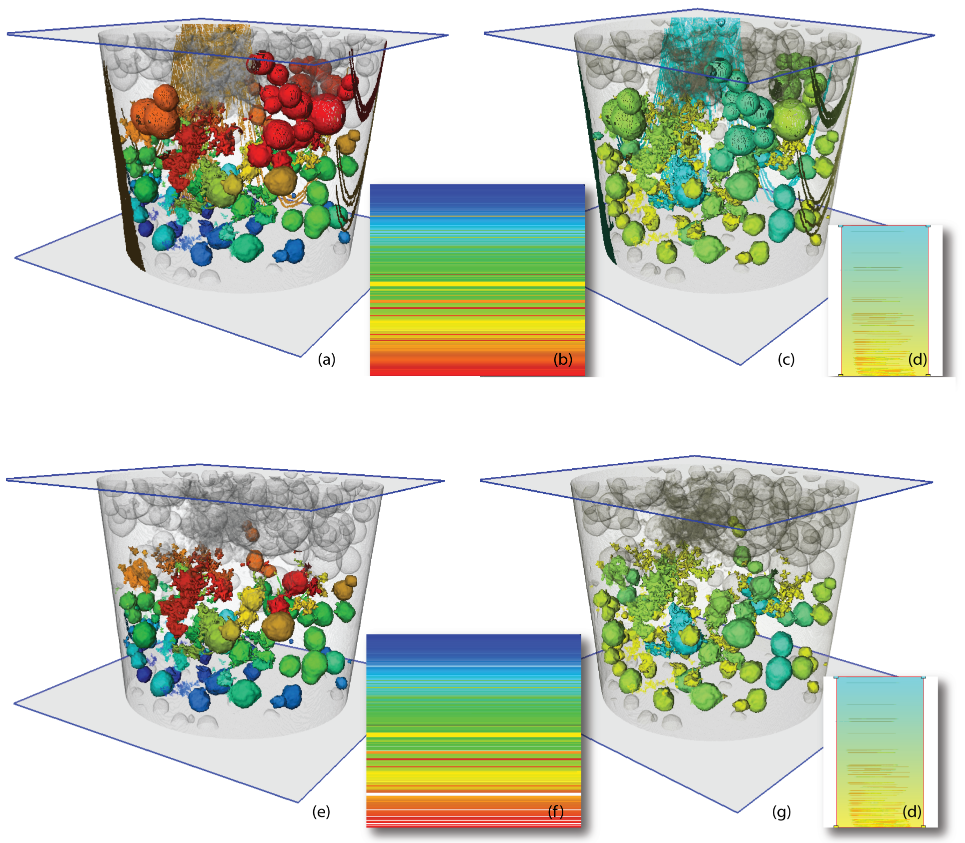
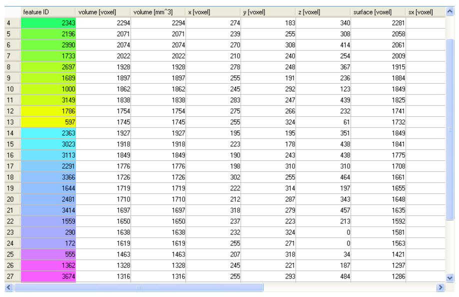
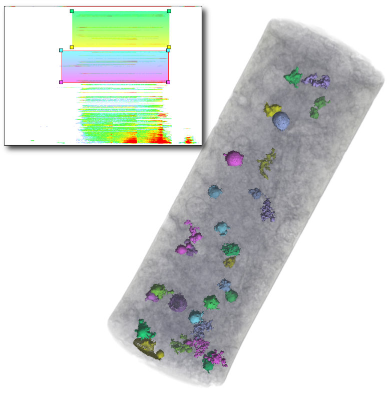


Figure 4.8: *Color coding as described in Section 4.2.2 is useful to get a first impression of all grown features (Figures (a), (b), (c) and (d)). To remove unwanted features, the opacity for the appropriate features in the color ramp are set to zero (in this case emphasized by the white lines in Figure (f)) (Figures (e), (f)). When a 2D feature size transfer function is applied additionally (Figure (d)), the affected features are excluded from rendering and evaluation (Figure (g)).*



	feature ID	volume [voxel]	volume [mm^3]	x [voxel]	y [voxel]	z [voxel]	surface [voxel]	sx [voxel]
4	2343	2294	2294	274	183	340	2281	
5	2196	2071	2071	239	255	308	2058	
6	2990	2074	2074	270	308	414	2061	
7	1733	2022	2022	210	240	254	2009	
8	2697	1928	1928	278	248	367	1915	
9	1689	1897	1897	255	191	236	1884	
10	1000	1862	1862	245	292	123	1849	
11	3149	1838	1838	283	247	439	1825	
12	1786	1754	1754	275	266	232	1741	
13	597	1745	1745	255	324	61	1732	
14	2363	1927	1927	195	195	351	1849	
15	3023	1918	1918	223	178	438	1841	
16	3113	1849	1849	190	243	438	1775	
17	2291	1776	1776	198	310	310	1708	
18	3366	1726	1726	302	255	464	1661	
19	1644	1719	1719	222	314	197	1655	
20	2481	1710	1710	212	287	343	1648	
21	3414	1697	1697	318	279	457	1635	
22	1559	1650	1650	237	223	213	1592	
23	290	1638	1638	232	324	0	1581	
24	172	1619	1619	255	271	0	1563	
25	555	1463	1463	207	318	34	1421	
26	1362	1328	1328	245	221	187	1297	
27	3674	1316	1316	255	293	484	1286	

(a) The feature table.



(b) The volume view.

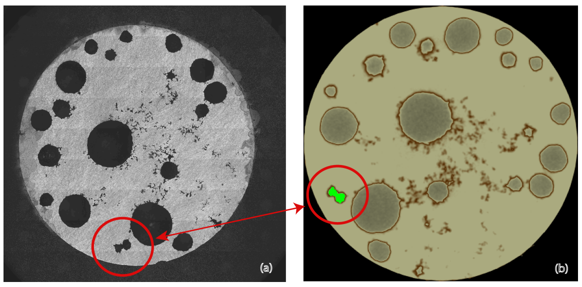
Figure 4.9: *The aluminum tensile test sample has two different types of defects: gas pores and shrinkages, similar to the RPTS (Figure 4.6). (a) The features in the feature table are defined via the classification widgets in the 2D TF (Figure 4.9(b) top left) and color coded respectively. (b) The defined features are highlighted in the volume view generated with GPU-based real-time ray-casting.*

coding in the table (Figure 4.9(a)) is equivalent to the color of the features defined in the 2D transfer function and used for rendering (Figure 4.9(b)).

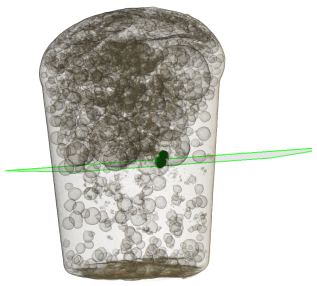
Picking in the tables is possible and linked with the renderer, slice viewer, and the graphical feature view as described in Section 4.2.1.

4.2.5 Optional Slice Plane

Figure 4.10 shows a comparison of slice images classified and rendered with our pipeline (Figure 4.10(b)) and images acquired via microscope (Olympus BX-51) from slices cut from the actual aluminum RPTS part (Figure 4.10(a)). To achieve the exact copy of the microscope image with our application it is necessary to tilt the z-plane horizontally as well as vertically for arbitrary degrees, because the electron microscope slice images do not lie exactly in the z-direction of our application. The registration occurs manually with one respective slider for each direction. Figure 4.10(c) shows the optional z-plane positioned in the 3D space.



(a) Comparison of slice images.



(b) The tilted z-plane.

Figure 4.10: *For comparing the measurements of our application with the manual measurements, we need to register the two slice planes. (a) A slice image of a RPTS pictured with an electron microscope (left) versus a slice image of our application (right). (b) It is necessary to tilt the slice plane along the z-direction to get the electron microscope's perspective.*

4.3 Volume Rendering

The size of volumetric datasets like industrial CT data are quite large and still increasing at a rapid rate. Although today's graphics hardware offers 1GB and more of onboard memory, which allows ray casting algorithms on the graphics processing unit (GPU) to achieve real-time frame rates, the general amount of texture memory is limited. Due to the additional texture memory used for the feature volume (Section 3.3) and the feature growth table (Section 3.4) in this application, more than three times the size of the density volume itself is used. Therefore bricking (described in Hadwiger et al. [2005]) is employed, for keeping only the visible subset of the entire volume in GPU memory, which is described in the following Section 4.3.1. Volume rendering is performed by using ray-casting in the fragment shader as described in Stegmaier et al. [2005], where the shader modifications for the feature rendering are explained in Section 4.3.2.

4.3.1 Brick Caching

As already mentioned, the problem at hand is the limited amount of texture memory in GPUs. Beside the 16-bit one-channel 3D texture of the original density volume itself, this application needs also a second 16-bit two-channel 3D texture for the feature volume (Section 3.3) and a two-channel 2D texture array for the feature

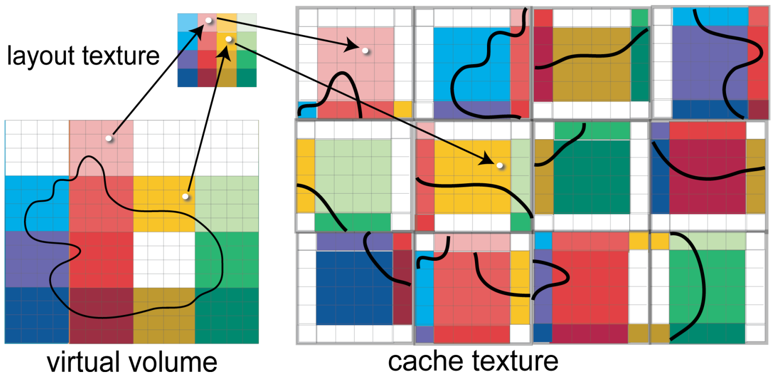


Figure 4.11: *Only non-empty bricks are stored in an arbitrary sequence in the 3D cache texture. During rendering, translation between virtual volume space and physical cache texture coordinates is performed via texture fetches in a small 3D layout texture described in Beyer et al. [2008].*

growth table, where the final size depends on the number of timesteps t and features resulting from region growing (Section 3.4.1). So it is obvious that we need a compression method for high-resolution industrial CT data to handle such amounts of data while sustaining an acceptable frame rate.

First, the user defines the final texture cache size depending on the memory capacity of the GPU and the data size, that later stores the active part of the volume. Then, the volume is subdivided into equally sized bricks of size n^3 (n has to be a power of two) in a pre-processing step. These bricks are stored in the central processing unit (CPU) memory. A brick is stored with the size $(n+2)^3$ because the voxels at brick boundaries need to be duplicated for texture filtering. During this process, the minimum and maximum density values of each brick are stored for culling at run time.

Culling (Section 3.5.1) determines a list of active bricks that are packed into a single 3D brick cache texture which is loaded into the GPU memory. This is also the volume used for the region growing process and, therefore, determines the size of the feature volume. Certainly bricks in this texture cache have to be updated whenever the transfer function changes. Especially in industrial CT data the amount of air surrounding the scanned object is very high and culling often leads to less of half the volume size as demonstrated in Figure 3.6. Hence, not only the amount of texture memory is reduced but also the computation time for region growing is reduced. Because the cache has a different size than the volume itself, the bricks are stored in an arbitrary sequence in memory (Figure 4.11). During rendering, a low resolution 3D layout texture with one texel per brick saves the offset of each position for each brick from virtual volume space to physical cache texture coordinates. This address translation is shown in Figure 4.11 where the layout texture encodes (x, y, z) address translation information in the RGB channels. Beyer et al. [2008] introduce the following formulation for the address translation of a volume space coordinate $\mathbf{x}_{x,y,z} \in [0,1]^3$ to cache texture coordinates $\mathbf{x}'_{x,y,z} \in [0,1]^3$:

$$\mathbf{x}'_{x,y,z} = \mathbf{x}_{x,y,z} \cdot \mathbf{bscale}_{x,y,z} + \mathbf{t}_{x,y,z}, \tag{4.1}$$

where $\mathbf{t}_{x,y,z}$ is the RGB-tuple from the layout texture corresponding to volume coordinate $\mathbf{x}_{x,y,z}$, and **bscale** is a constant scale factor for matching the different coordinate spaces of the volume and the cache. When filling the layout texture, the former is computed as:

$$\mathbf{t}_{x,y,z} = \left(\mathbf{b}'_{x,y,z} \cdot \mathbf{bres}'_{x,y,z} - \mathbf{o}_{x,y,z}\right) / \mathbf{csize}_{x,y,z} \tag{4.2}$$

where $\mathbf{b}'$ is the position of the brick in the cache, $\mathbf{bres}'$ is the storage resolution of the brick, and **csize** is the cache texture size in texels to produce texture coordinates in the $[0,1]$ range. The offset $\mathbf{o}_{x,y,z}$ is computed as:

$$\mathbf{o}_{x,y,z} = \mathbf{b}_{x,y,z} \cdot \mathbf{bres}_{x,y,z} \tag{4.3}$$

where **b** is the position of the brick in the volume, and **bres** is the brick resolution. The global scale factor **bscale** is computed as:

$$\mathbf{bscale}_{x,y,z} = \mathbf{vsize}_{x,y,z} / \mathbf{csize}_{x,y,z}, \tag{4.4}$$

where **vsize** is the size of the volume in voxels.

4.3.2 Rendering

For the remaining volume in the 3D cache texture, rendering is done via ray-casting in the fragment shader. During rendering, a single texture fetch from the 3D feature texture yields everything needed to reconstruct the voxels feature size curve via the feature growth table stored in a 2D texture. The pseudo-code (similar to GLSL) in Figure 4.12 illustrates the main steps that need to be done in order to determine the color and opacity (without shading) of a sample as RGBA tuple in the variable `out`, which can then be composited during ray-casting, or simply displayed in an orthogonal slice view.

```
float density = texture3D(density_volume, sample_coord3);
vec2 feat_vox = texture3D(feature_volume, sample_coord3);
float birthID   = feat_vox.x;
float birthTime = feat_vox.y;

if ((birthID == ID_NONE) || (birthTime > T_CUR)) {
  out = texture1D(tf1D, density);
} else {
  vec2 fsmap = texture2D(growth2D, vec2(T_CUR, birthID));
  float curSize = fsmap.x;
  float curID   = fsmap.y;

  if (curId == PICKED_ID) {
    out = pickingColor;
  } else {
    out = texture1D(selection1D, curID);
    if ( (out.a > 0.0) && !use_color_ramp_1D )
      out = texture3D(tf3D, vec3(density, curSize, T_CUR));
    if (out.a == 0.0)
      out = texture1D(tf1D, density);
  }
}
```

Figure 4.12: *Pseudo code for the main steps to determine the color and opacity of a feature voxel in the fragment shader. If the voxel does not belong to a feature the 1D opacity transfer function is used.*

The volume is rendered for a specific time step, i.e., the global current time t_{cur}, which is denoted as `T_CUR` in the code. For a sample with volume coordinates `sample_coord3`, the density volume (`density_volume`) and the feature volume (`feature_volume`) are sampled at that position, which yields the density, the feature birth ID (`birthID`) and the birth time (`birthTime`). If no feature is present at a specified location given the current classification, it indicates that either, no feature exists there at all (`ID_NONE`), or the feature does not exist yet at time t_{cur}, or it is mapped to zero opacity in the feature transfer function (`tf3D`). If no feature is present, the regular 1D density transfer function is applied to the sample (`tf1D`). However, it is also easily possible to use a 2D instead of a 1D transfer function, such as density/gradient-magnitude (Kniss et al. [2001]), or LH (Šereda et al. [2006]). For color-coding features or using individual colors and opacity (see Section 4.2.2), a 1D table is used (`selection1D`), which overrides the feature transfer function when `use_color_ramp_1D` is set. For picked features an additional `pickingColor` can be specified (see Section 4.2.1).

Chapter 5

Quantification and Results

In order to assess the quality of materials or the whole casting process itself, it is necessary to quantify the features contained in a data set, e.g., their number, volume, surface area, as well as global statistical measures such as average volume or area.

The quantitative information displayed by our system is separated into information that has already been computed in the pre-computation stage (Chapter 3), and additional information obtained during exploration by using the 3D transfer function (Section 4.1) and individual feature exploration (Section 4.2).

5.1 Feature Quantification

In contrast to rendering during exploration, quantification does not primarily consider individual voxels (samples), but whole features with all their voxels. Therefore, quantification does not need the feature volume, but most of all the feature growth table (Section 3.4). The feature growth table contains the representative feature size curves for every feature, as well as information computed during region growing that was not needed for rendering (i.e., lists of voxels comprising individual features). For a feature f_i, the representative feature size curve $f_s(\mathbf{x},t)$ is the feature size curve of the feature's seed voxel $f_s(\mathbf{x}_s,t)$, because this voxel belongs to the feature from the beginning, see Section 3.2. All other voxel have the same curve from the time they start joining the feature. The size of the feature in voxels at time t_i can be determined directly from this representative feature size curve, which is stored in the feature growth table. Not only the size at time t_i is available, but also all the relevant feature parameters like the position, surface area, etc. (see Section 3.7). However, this considers only the region growing process itself, not the classification done via the transfer function which can exclude features and whole feature classes from quantification. During the region growing process (Section 3.5.2), a list of contained voxels is incrementally constructed for each feature. In order to quantify a feature, these voxels have to be visited, and their density, together with the feature's size at time t_i, must be used to perform a lookup in the 3D feature transfer function. The resulting opacity determines whether this voxel should be included in the quantification or not. In addition to using the opacity, feature classes can be quantified individually (i.e., quantifying each widget's classification separately), or combining the classification of several widgets to collectively classify a single feature class.

We compute the most common measures such as the volume of features in voxels, the volume in cubic millimeters via the reference size of a single voxel in x, y, z, the surface area of a feature, the size of the bounding box encapsulating the feature, the surface area of the features shape projected onto one of the three axial planes, density average and standard deviation, as well as global statistical measures such as average feature size and standard deviation. Additional measures can be added easily. However, the focus of our system is to provide the basis for interactively specifying *what* should be quantified. Including other measured quantities is the subject of future work. Examples of quantification results from the statistical calculations described in Section 3.7 are shown in Table 5.1 (Section 5.2.1) and Table 5.10 (Section 5.2.8), resulting from the feature table described in Section 4.2.4 (Figure 4.9(a)).

5.2 Results

This chapter describes results generated with our system for real-world industrial CT data. The tables provided for each sample give quantification results for different void sizes (feature classes) contained in the respective data set, including the number of features in each class, their average volume in voxels, and the percentage of voxels in the class with respect to the number of voxels in the whole part (excluding

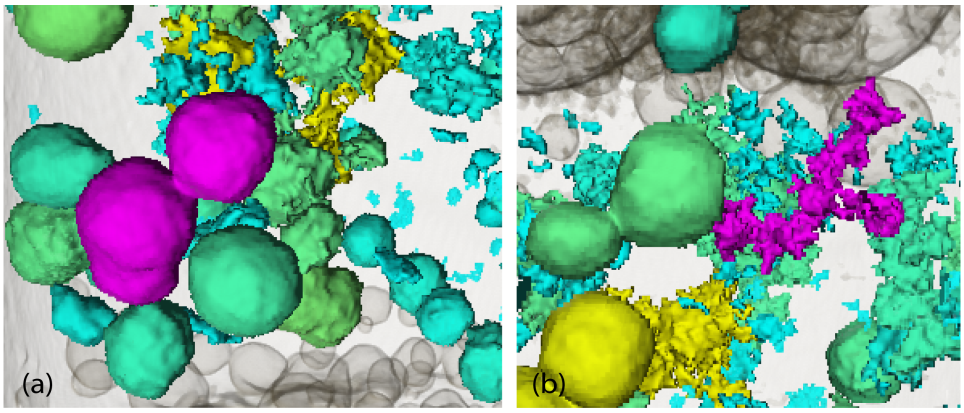


Figure 5.1: *In the RPTS two kinds of features with different origin exists: gas pores (a) and shrinkage cavities (b) both colored in purple. The measurements of both are listed in Table 5.1.*

the surrounding air). The values in the tables can be immediately obtained by clicking on the desired widget and adjusted by simply changing the size and/or the position of the widget. During widget movement the quantification results as well as the visual results in the volume renderer are kept consistent. Additionally, for better orientation in the feature size/density domain, the feature size and density range the widget covers is displayed, as already described in Section 4.1.2.

5.2.1 Reduced-Pressure-Test Sample

Figure 4.6 shows a "reduced-pressure-test sample" (RPTS), which is used in the casting industry to evaluate the gas content of an aluminum or copper melt. Therefore about $30cm^3$ ($1.83in^3$) of the melt is solidified at a pressure of 8000 Pa (1.16 psi), which causes the gas in the melt to form pores in the metallic volume. Furthermore, the shrinkage of the metal during solidification causes shrinkage cavities to be formed in the center of the upper regions of the sample. Therefore, these samples are ideal test pieces for the evaluation of feature detection, as they are virtually full of different pore sizes (Figure 5.1(a)) and shrinkage cavities (Figure 5.1(b)). The evaluation of these basic types of features is of great relevance to the casting industry as they are the most common defects in metallic cast parts beside non-metallic inclusions. Especially the distinction between these defects is of interest for the casting expert, as they are both voids but have a completely different origin, and therefore have to be treated differently in the casting process. Table 5.1 shows a quantification example at the time step t_c for the measures of the two features shown in Figure 5.1 respectively:

feature ID	size [voxels]	size [mm^3]	x/y/z pos.	surface	sx/sy/sz	px/py/pz
148 (Fig. 5.1(a))	24464	29.966	199/63/371	3723	42/35/53	1144/1222/923
35 (Fig. 5.1(b))	4441	5.440	155/254/304	3357	45/50/52	723/745/838

Table 5.1: *This table represents a quantification example for the two features from Figure 5.1, resulting from the feature table described in Section 4.2.4.*

The highlighted feature (magenta) in Figure 5.1(a) illustrates that interconnected gas pores also merge in the pre-processing step, and are therefore handled as one single feature. This is also valid for the measurements shown in Table 5.1 (Feature 148). For this application, the time step t_c in our system was chosen interactively, such that the rendered features comply with the actual position and size of the different defects. In this case, a (2D) feature transfer function for the time step t_c was sufficient in order to classify the gas pores and the shrinkage cavities according to their size and density. In cases where the separation of features of different origin is not directly possible with the 2D feature transfer function, picking can be used to quantify single features. Table 5.2 shows the quantification results according to the feature size transfer function from Figure 4.6, where the classification occurs by separating the boundary of the features and the shrinkages (orange widget). Shrinkages correspond to the high densities whereas the interior of the voids (color gradient from red to blue) correspond to the low density values. The quantification results are presented for the appropriate colors in Table 5.2:

Data set	class	feature count	avg.vol. [voxels]	% of part vol.
RPTS	🟥 left	830	6280.5	8.765
	🟧 right	2008	3879.5	13.099

Table 5.2: *The classification occurs due to the density values shown in the transfer function from Figure 4.6, and separates the boundary and shrinkage voxels (orange) from the interior of the voids (red).*

5.2.2 Asphalt Core

The asphalt core depicted in Figure 5.2 with a diameter of 100*mm* (3.9*in*) is used to characterize the quality of the asphalt. It is composed of three main phases:

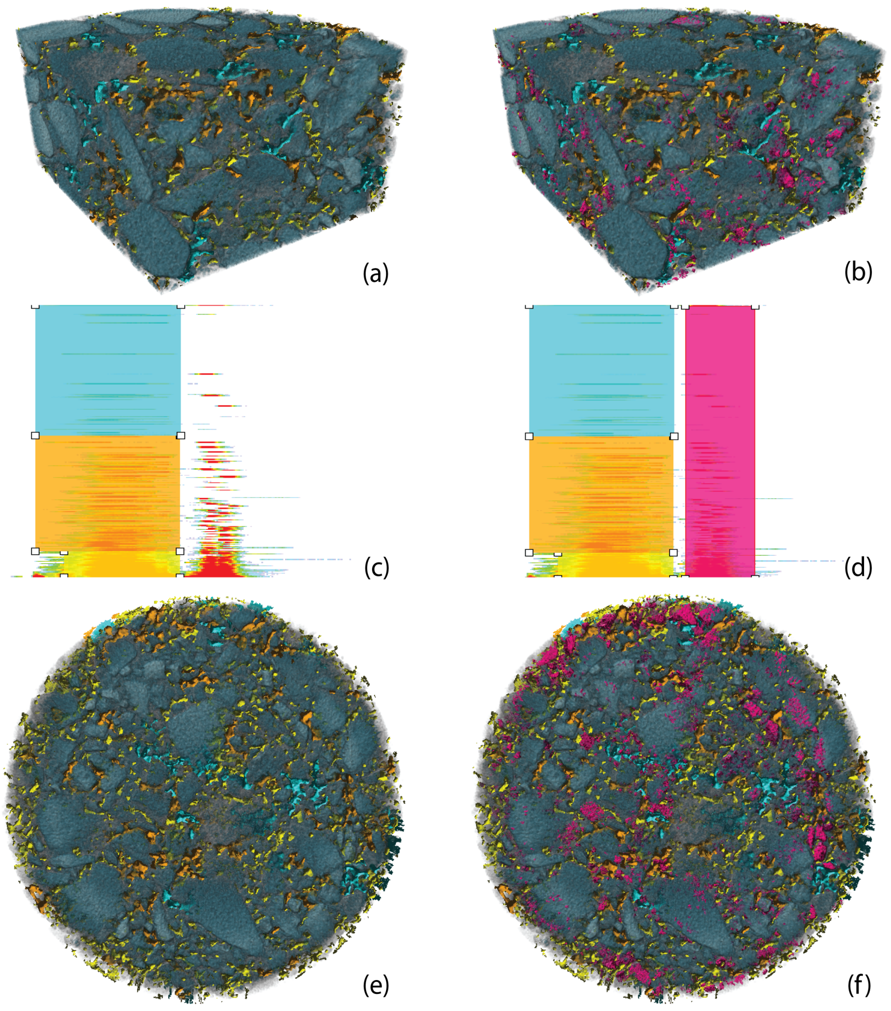


Figure 5.2: *Asphalt core with different material components. Region growing yields features of two clearly distinguishable density ranges. The higher density range (purple widget in right column (b), (d) and (f)) corresponds to mineral components of higher density that are incorporated in the coarse fraction of the asphalt, which in this case are undesired features. Mapping them to completely transparent (left column (a), (c) and (e)) removes them. The phases between the coarse mineral components of lower density can be further distinguished according to their feature size, giving different constituents of the fines: small features (yellow), medium-sized features (orange), and large features (turquoise).*

the mineral phase, the bitumen binding phase, and pores. The mineral phase can be composed of different minerals in different grain sizes. The knowledge of the composition and distribution of the single phases is of great importance for the quality of the material. For example, the total volume of the pores is limited in order to prevent ruts on the street. It is also of interest if the pores are connected to the surface, to characterize the drainage of the asphalt. This highly complex composition concerning density profile and size of the inclusions makes a reliable evaluation especially difficult. Such samples ideally show the advantages of interactive feature detection. Due to the fundamentally different behavior and composition of the employed phases, a 3D transfer function can be used to separately characterize the different phases. Figure 5.2 shows two different feature transfer functions at a specific time t_c where the quantification of the classification is shown in Table 5.3:

Data set	class	feature count	avg.vol. [voxels]	% of part vol.
Asphalt Core	small	6068	277.6	2.158
	med.	243	5735.9	1.785
	large	40	19542.4	1.001
	right	12345	170.5	2.696

Table 5.3: *Quantification results for the transfer function widgets in Figure 5.2.*

5.2.3 Golf Ball

Figure 5.3 shows a 2-piece construction golf ball. The inner piece is reinforced with dense particles. To evaluate the quality of these reinforcements, their distribution and size were checked using industrial CT. An inhomogeneous distribution of the particles may result in a deviation of the center of mass compared to the geometric center of the ball which causes a gyration during rolling of the ball. The two different time steps in Figure 5.3 clearly show the influence of the region growing parameter (here, k of region growing method B) on the quantification result at the end of the pipeline. A 3D transfer function was used to obtain an optimal result for the complex structure of the sample. Table 5.4 contrasts the quantification results of two selected time steps. The user visually determined that the earlier time step (values in parentheses) corresponded to incomplete results, whereas the later one resulted in a plausibly complete detection of features. For example, the particle indicated in Figure 5.3 corresponds to an agglomeration of smaller particles during the production of the golf ball. It also appears in the quantification as a singular particle of large size (last row of Table 5.4).

Data set	class	feature count	avg.vol. [voxels]	% of part vol.
Golf Ball	small	6131 (4375)	70.2 (83.5)	0.044 (0.037)
	med.	1875 (948)	209.1 (224.2)	0.040 (0.022)
	large	78 (26)	1152.2 (1029.8)	0.092 (0.027)
	XL	1 (1)	9472 (3441)	0.0097 (0.0035)

Table 5.4: *Quantification results for Figure 5.3. The results for the golf ball are for the time step selected by the user as the "complete" one (Figure 5.3(b)), and an earlier, "incomplete" time step in parentheses (Figure 5.3(a)).*

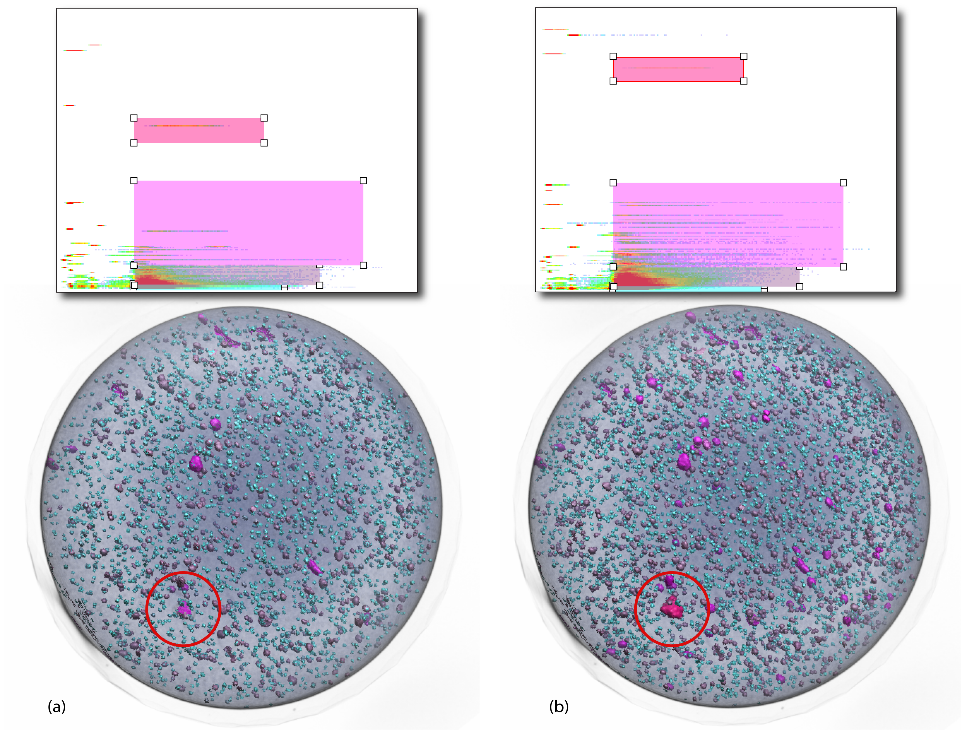


Figure 5.3: *A golf ball, reinforced with dense particles. Two different time steps and the corresponding 2D section of the 3D TF are shown. In the earlier time step (a), the indicated particle is still small because the region growing parameter has not advanced far enough yet. In the later time step (b), it has become a big feature that does not grow further since it has reached its maximum (actual) size. See also Table 5.4.*

5.2.4 Laser Build-up Welding

The welding of high alloyed steel was built-up using a high power laser. The sample exhibits pores and inclusions of higher density and is therefore an ideal test phantom for the computed tomography. The geometry with two parallel shoulders which are prone to scatter under radioscopic measurement is especially interesting for testing CT equipment and visualization. This workpart is also very interesting for our application because it shows that it is possible to distinguish the inclusions by means of their different densities. As shown in the transfer function in Figure 5.4 (bottom right), three main density classes are well defined which brings out the different kinds of inclusions. Table 5.5 shows the quantification results for these feature classes:

Data set	class	feature count	avg.vol. [voxels]	% of part vol.
Laser	left	720	348.9	0.137
Build-up	mid.	2984	240.9	0.391
Welding	right	2329	573.5	0.726

Table 5.5: *The quantification results of the three feature classes defined in the transfer function in Figure 5.4 (bottom right).*

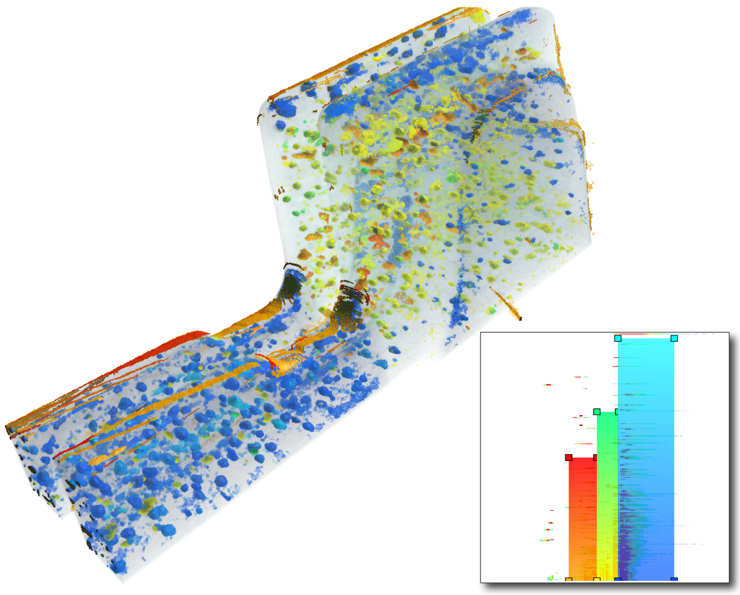


Figure 5.4: *The inclusions of the welding of alloyed steel are of different consistency and therefore can be well differentiated by their densities. The sizes of the features are additionally encoded with a color gradient along the y-direction in each widget.*

5.2.5 Refractory Material

The model shown in Figure 5.5 depicts a sample of a refractory material that is used for the lining of metallurgical aggregates, where metallic melt is processed, for isolation and protection purposes. For the quality of such materials the distribution and size of the mineral phases is important as well as the porosity contained. These properties influence the mechanical behavior of the material at high temperatures as well as the elongation under thermal load. The feature size transfer function allows to evaluate the different fractions of particles and their distribution as shown in Table 5.6:

Data set	class	feature count	avg.vol. [voxels]	% of part vol.
Refractory	small	20288	246.8	0.513
Material	med.-XL	633	7024.9	0.455

Table 5.6: *The quantification results for the refractory material sample in Figure 5.5 .*

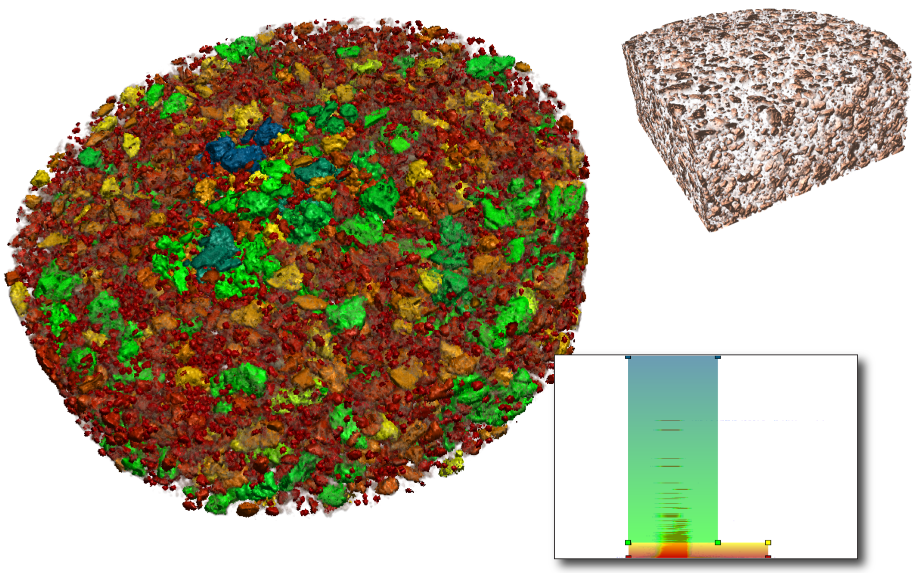


Figure 5.5: *Visualization of different mineral phases in an industrial ceramic according to their size by using the 2D feature size transfer function (bottom right). For comparison a part of the industrial ceramic is shown in the upper right, DVR shaded, without feature classification.*

The sample in Figure 5.5 was pre-computed using the previously described 1D transfer function culling option from Section 3.5.1. The slice images and the appropriate windowing and transfer functions in Figure 3.7, 3.8 and 3.9 demonstrate the transfer function settings used in this example.

5.2.6 Isolation Material

The isolation material shown in Figure 5.6 is used in the construction industry to isolate buildings where moisture may emerge. Therefore, the polystyrene component is covered by a ceramic layer. The features in Figure 5.6 colored in dark gray mark areas where the ceramic agglomerates the plastics phase. The agglomeration rate has influence on the thermal properties of the product and, therefore, has to be checked. The results in Table 5.7 show that the agglomerated features are relatively big (i.e., the medium (green) and large (gray) feature classes) and cover nearly the same part of the volume than the small (orange) features:

Data set	class	feature count	avg.vol. [voxels]	% of part vol.
Isolation	small	8911	328.3	8.573
Material	med.	321	3554.9	3.383
	large	85	12970.5	3.230

Table 5.7: *The quantification results of the isolation material shown in Figure 5.6.*

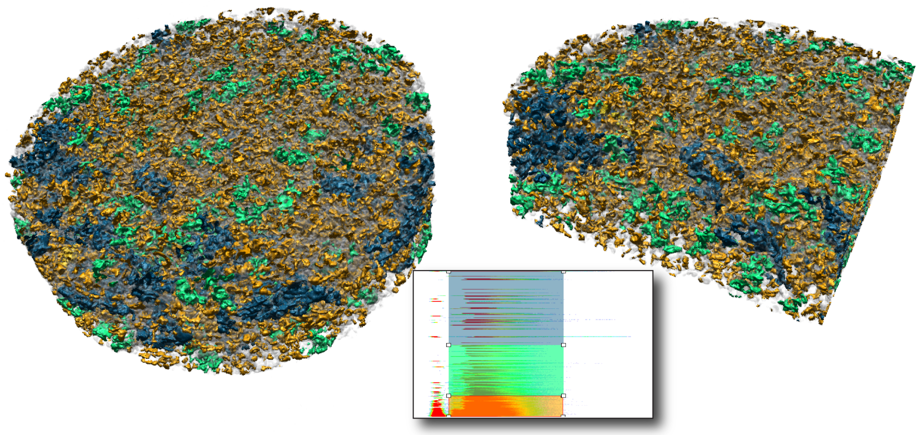


Figure 5.6: *The isolation material is used in the construction industry to avoid dampness. Because the agglomeration rate has influence on the thermal properties, visualization and quantification is of great interest.*

5.2.7 Aluminum Tensile Test Sample

The aluminum tensile test sample is used to characterize the static mechanical properties of the aluminum alloy at hand. In the tensile test, the sample is loaded with an increasing static load until the the sample fractures. The load and elongation at which the fracture occurs are important characteristics of the mechanical properties of the material. When using samples of parts coming out of production, the samples normally exhibit defects that influence the measurement results. Depending on type and size of the defect this influence may vary greatly. Therefore, it is interesting for the materials expert to characterize the defects in advance of the measurement to be able to better predict the possible disturbance.

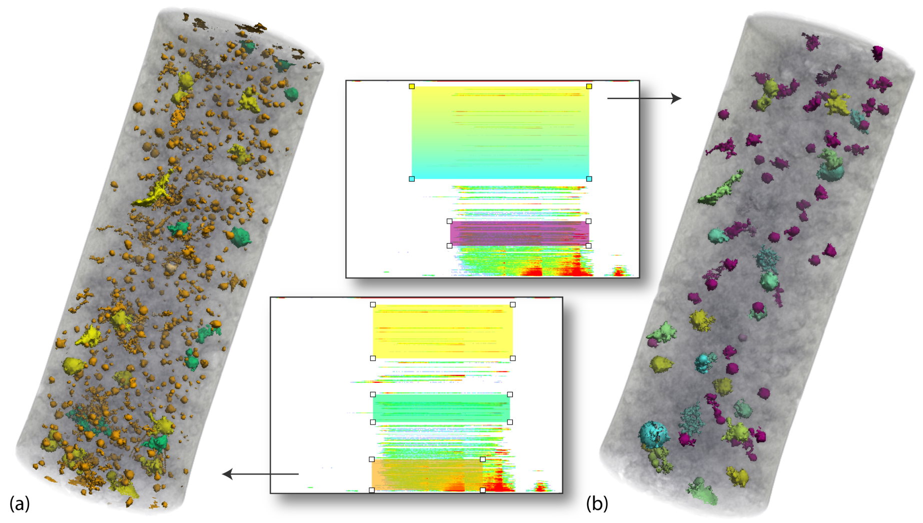


Figure 5.7: *In Figure (a) a lot of small features are shown (orange). The right Figure (b) shows some bigger features which are color coded with a color gradient from turquoise to yellow.*

The sample shown in Figure 5.7 has two different types of defects. The round gas pores have a significantly different impact on the mechanical properties compared to the mushy shrinkage defects, as already described in Section 5.2.1 and shown in Figure 5.1. The correlation of the measured properties of the sample to the defects detected is part of ongoing research in this field. Figure 5.7 shows two different exploration examples with their respective transfer functions. The related quantification is presented in Table 5.8.

Data set	class	feature count	avg.vol. [voxels]	% of part vol.
Aluminum Tensile	small	1348	202.9	0.303
Test Sample (ATTS)	med.	20	3282.4	0.073
(Fig. 5.7; left)	large	13	7318.7	0.105
ATTS	med.	70	1796.3	0.142
(Fig. 5.7; right)	large	25	6646.2	0.184

Table 5.8: *The quantification of two possible classification examples in Figure 5.7 .*

5.2.8 Cast Housing I

Figure 4.2 depicts a part of a cast housing for the automotive industry. The part is an AlSi-type alloy produced in a die-casting process. As it carries fluids during operation, impermeability of the housing is one point of specification. Therefore, a characterization of voids in the casting has to be performed. Other requirements are the mechanical properties of the casting as well as the mechanical properties of specific parts of it. Volume defects such as gas pores or shrinkage porosity have a great influence on these properties. Not only the total volume of these voids influences the part's strength but also their distribution. Small, finely distributed pores will weaken the casting less than one large shrinkage pore of the same volume. Moreover, the casting has to fulfill different quality levels at different sections. For example no defects may be allowed on a visible surface of the part, while they are irrelevant on the surfaces of inner structures. The quantification results for Figure 4.2 as well as for Figure 5.8 are shown in Table 5.9:

Data set	class	feature count	avg.vol. [voxels]	% of part vol.
Cast	small	337	158.1	0.041
Housing I	med.	16	717.2	0.088
(Fig. 4.2)	large	6	1629.7	0.075
	XL	6	5181.3	0.024
	XXL	3	10678.7	0.026
Cast	small	359	236.2	0.200
Housing I	med.	10	3671.4	0.087
(Fig. 5.8)	large	4	12933.5	0.122

Table 5.9: *Two example quantifications for the cast housing which contains five different feature classes in Figure 4.2 and just three feature classes in Figure 5.8.*

The housing in Figure 5.8 shows four big shrinkage pores (classified with the orange widget in the transfer function; bottom left). We focus on the pore between the main wall of the housing and a drilled hole (Figure 5.8(a)). For the impermeability of the housing it is important that this void builds no connection between the inner and outer part of the housing, which is the case in our example. By contrast the other

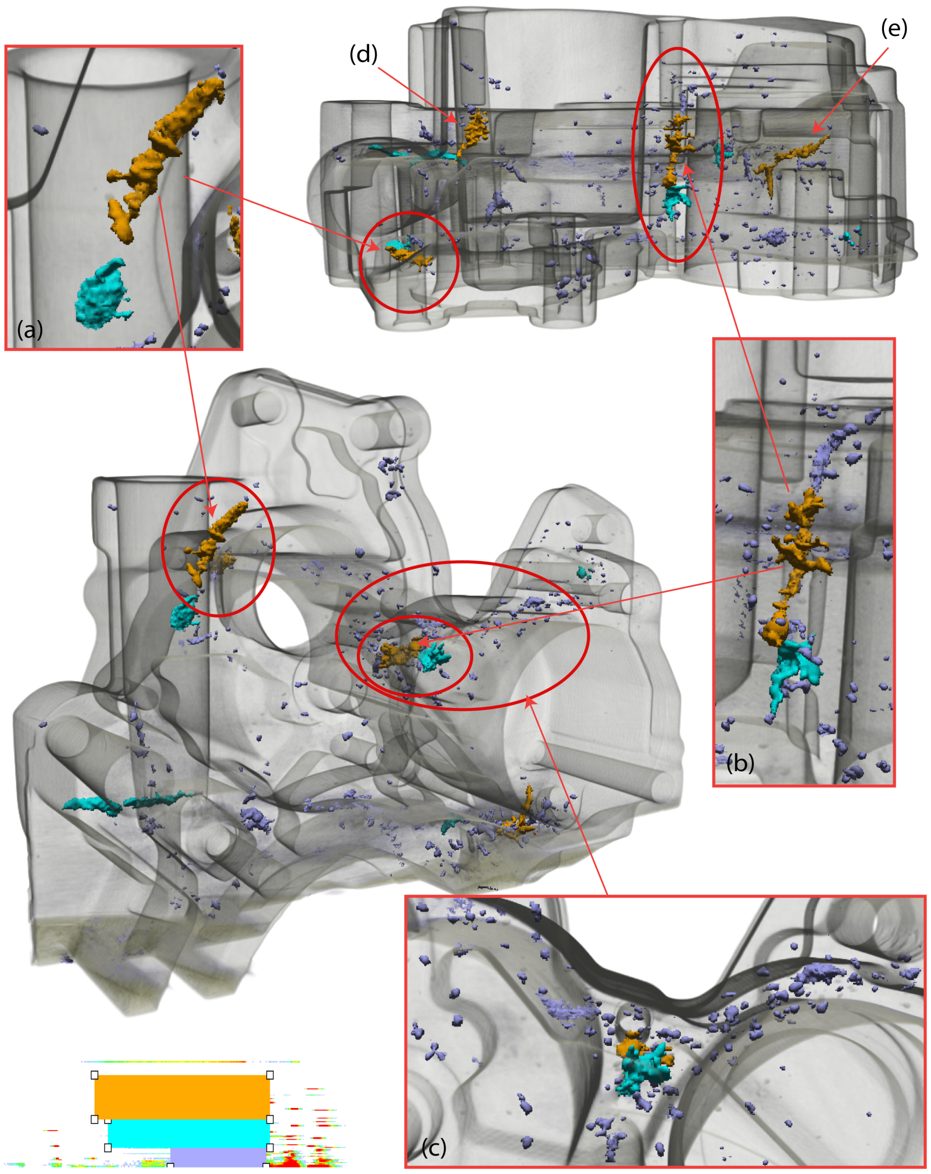


Figure 5.8: *An application example for a defect analysis of a cast housing. To ensure the closeness of different parts of the housing, it is important that voids do not connect them (a). Otherwise they must not exceed a certain size (b), (d) and (e), or frequency (c) to keep the part's strength.*

pore between the two aggregate sections in the center of the part (Figure 5.8(b)), has no relevant influence on the part's strength in this section and can be accepted.

The small finely distributed gas pores (the blue ones in Figure 5.8(c)) in that part can also be accepted, as they do not reach the surface, where the fluids flow. The measurements for the four orange features are shown in Table 5.10. These measurements result from the feature table described in Section 4.2.4 which calculates the values due to the classification in the feature size transfer function:

feature ID	size [voxels]	size [mm^3]	x/y/z pos.	surface	sx/sy/sz	px/py/pz
45 (Fig. 5.8(a))	4191	4191	159/314/149	4171	50/125/86	216/361/63
386 (Fig. 5.8(b))	4441	4441	380/214/263	4395	51/125/86	266/381/65
86 (Fig. 5.8(d))	3932	3932	223/182/175	3913	50/124/86	216/349/63
1448 (Fig. 5.8(e))	6411	6411	543/230/432	6141	53/125/86	437/935/81

Table 5.10: *The measurements for the four features classified by the orange widget in Figure 5.8 are a quantification example following from the feature table described in Section 4.2.4.*

5.2.9 Cast Housing II

Figure 5.9 illustrates a part of another cast housing. For this casting the same rules for specification must be fulfilled as already described in the previous Section 5.2.8. In contrast to the cast housing from Figure 5.8, the classified voids have no significant impact on the quality of the housing. Their quantification results are shown in

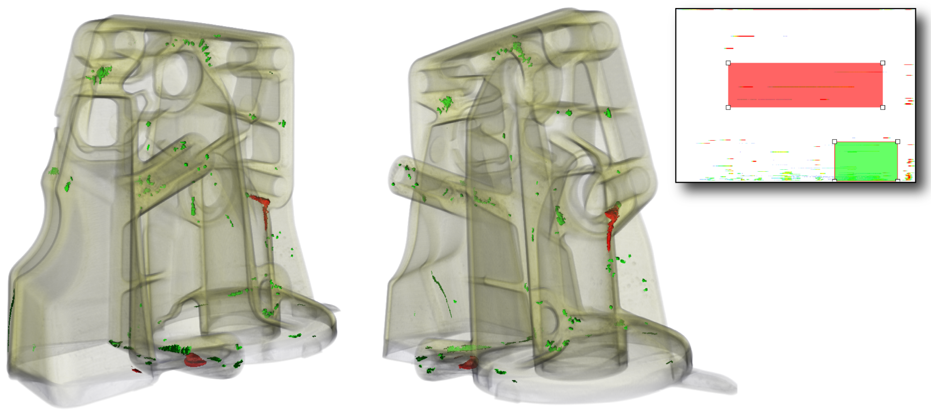


Figure 5.9: *Another part of a cast housing where two feature size classes are visualized, shown in the transfer function (top right).*

Table 5.11:

Data set	class	feature count	avg.vol. [voxels]	% of part vol.
Cast	small	103	233.5	0.019
Housing II	med.	4	9429	0.028

Table 5.11: *Quantification of the two selected feature classes of the casting in Figure 5.9.*

5.3 Performance and Memory Usage

The following tables give an overview of the pre-computation times as well as frame rates, memory size, and number of features for some example datasets described in the previous sections. The performance has been measured using the following system:

- Single processor PC, AMD Athlon 64 2.4GHz, 4GB
- GeForce 8800 GTX, 768MB
- Windows XP64

The first region growing step always consumes the most time.

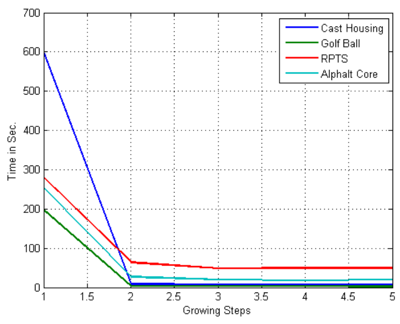
Additional data needs to be computed (e.g., the standard deviation of the seed voxel neighborhood), most voxels are considered as seeds, and regions are grown and discarded if they become bigger than *max_featuresize*. Consequently, all subsequent time steps are much faster. Table 5.12 gives typical numbers for pre-computation times, comparing the two region growing methods described in Section 3.6 for some example data sets:

The bit masks maintained during region growing, described in Section 3.5.2, ensure that many voxels are visited only once over time, which implies that the multi-pass region growing performance of our approach is not much worse than computing only a single region growing pass, even for many time steps.

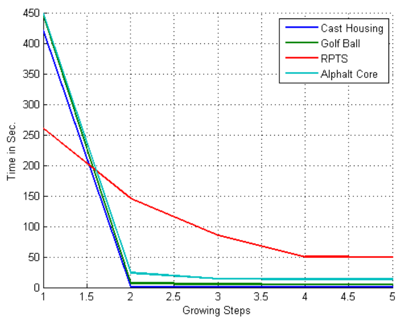
Table 5.12 shows that region growing time not only depends on the size of the data, but also on the number and character of the features grown. When comparing

Data Set	Pre-comp. (1st, 2nd, 3rd..*n*th time step, overall for 16 steps), Method A||B							
Cast Housing I	600 s	8.7 s	7-8 s	12.1 min	421 s	0.84 s	0.5-0.9 s	7.2 min
Golf Ball	198 s	3.8 s	3-4 s	4.2 min	447 s	7.2 s	5-6 s	8.9 min
RPTS	280 s	64.6 s	45-66 s	20.5 min	260 s	146 s	29-85 s	15.1 min
Asphalt Core	253 s	27.4 s	17-21 s	9.4 min	450 s	24 s	10-15 s	10.9 min

Table 5.12: *Some example datasets, with typical pre-computation times (first and second time step, range for 3rd+; and overall time for 16 time steps). The left four columns of pre-processing use region growing method A, and the right four columns method B (Section 3.6). The first step is the most expensive one; after the second step, computation times decrease rapidly.*



(a) Region Growing method A.



(b) Region Growing method B.

Figure 5.10: *Visualization of the pre-processing time for the example data sets from Table 5.12. Region Growing method A was better applicable to the Cast Housing and the RPTS. Region Growing method B led to better results for the Golf Ball and the Asphalt Core.*

the data sizes in Table 5.14 with the number of features in Table 5.13 and the pre-computation time of the 3rd+ time steps in Table 5.12 we see that the duration of the first time step strongly depends on the data size, while all following steps are related to the actual number and shape of features. On the one hand this derives from the boundary growing process which is done as a second pass at each time step for each feature (described in Section 3.5.2) -consequently the computation time increases with the number of features. On the other hand this derives from the number of bits left unset in the *checked mask* (see Section 3.5.2) which depends on the shape of the features. For example, although the cast housing dataset is bigger, the overall pre-computation process for the RPTS takes nearly twice the time. Figure 5.10 illustrates the impact of the two different region growing criteria in Section 3.6. Whereas the cast housing and the RPTS (Figure 5.10(a)) need the most pre-processing time using region growing method A, the Golf Ball and the Asphalt Core have a higher pre-processing time when using region growing method B (Figure 5.10(b)). These are also the region growing methods used for the final visualization and quantification results above. The higher pre-processing time denotes that the particular region growing criteria were able to identify more features and were therefore better suitable for the respective dataset. To handle different kinds of industry constructs, as shown in Section 5.2, it is important to have the choice between two or more different region growing algorithms. It turned out that region growing method B was better suitable for materials where inclusions of a broad range of different density values occur. The Laser Build-up Welding (described in Section 5.2.4) for example, contains different density classes among their inclusions (shown in the transfer function in Figure 5.4). Whereas the region growing method A was only able to classify features of the blue feature class, the algorithm of method B detects all inclusions. Also for the Asphalt Core and the Golf Ball, method B was much more suitable. On the other hand, region growing method A yields generally better results for castings.

Table 5.13 lists the numbers of features for the respective pre-computation steps for region growing method A (Section 3.6.1) for some example data sets.

Data Set	Num. features (1st, 2nd, 3rd, and 16th time step)			
Cast Housing I	1936	1953	1968	2646
Golf Ball	1177	1212	1243	1608
RPTS	2620	2633	2640	3238
Asphalt Core	20874	20889	20900	20914

Table 5.13: *Some example datasets, with the number of features resulting from region growing method A at each time step respectively.*

Volume rendering is fast, as only a few additional operations compared to regular volume rendering have to be executed per fragment (see the pseudo code in Section 4.3). Table 5.14 also lists the memory usage of the major additional data structures computed, i.e., the feature volume (first value) and the feature growth table for 16 time steps (second value):

Data Set	Resolution	Feat.-Mem.	Rendering
Cast Housing	667x465x512	606MB+128KB	16-22 fps
Golf Ball	512x512x256	256MB+128KB	22-28 fps
RPTS	373x377x512	275MB+192KB	20-23 fps
Asphalt Core	512x512x256	256MB+1.3MB	20-40 fps

Table 5.14: *The example data sets used above, with typical volume rendering frame rates (viewport 512x512) and the amount of memory used in the GPU.*

For rendering, only a subset of the entire feature volume needs to be in GPU memory due to texture bricking (see Section 4.3.1), whereas the feature growth table is always resident in texture memory in its entirety. The size of the feature growth table is dependent of the total number of features after the whole growing process (Table 5.13; right column).

Chapter 6

Conclusion and Future Work

This chapter draws the conclusions from the main contributions of this thesis. Furthermore it presents some suggestions for future work, like the additional employment of morphological operations, involvement of 3D measurement tools or extensions to the existing region growing methods.

6.1 Conclusion

This thesis presented an approach for interactive exploration of features in industrial CT volumes that helps to bridge the gap between visualization and feature detection. Given the complexity of feature and defect detection, and the wide variety of data and material properties, we do not claim that our approach solves all challenges in this area. However, it enables a powerful interactive workflow that tightly couples visualization and feature detection, by building on region growing, and allows for a full exploration of the volume with no or almost no beforehand parameter specification. The result of the exploration process is a classification of *all* feature classes of interest using transfer functions, which can immediately be used to quantify the corresponding features. This implies that subsequent quantification is visualization-driven as well, i.e., quantification is performed exactly on the region of interest the user has chosen to visualize. This empowers users who are experienced domain experts to decide on their own and make informed decisions for quantification, instead of relying on the result of a given set of parameters, which is the approach employed by systems currently used in practice.

Chapter 5 demonstrates that this implementation is applicable for varying data originating from industry. We believe that the concept presented in this thesis is very powerful, but it is also only one step towards driving defect and feature detection by visualization and bringing visualization methods and segmentation closer together. There are many possibilities that can be explored in the future, some of them are introduced below. We are also planning to extend the possibilities for quantification, specifically with respect to global measures such as porosity. We would like to investigate adaptive sampling schemes of the parameter (time) domain, as well as semi-automatic TF generation in our new 3D TF domain. Additionally, investigating how the quality of different region growing approaches is affected by choosing different parameters for time parameter t, would be worthwile.

6.2 Future Work

The application presented in this thesis constitutes the basis for enhancements in many directions. On one hand the feature size transfer function provides a couple of new combination possibilities, where, for example, morphological operations can be applied on the already segmented features for better exploration possibilities. On the other hand, the quantification is still in its infancy and we are planning to extend it, specifically with respect to global measures such as porosity or different measurement tools.

Morphological Operations

Mathematical morphology is based on algebra of non-linear operators, operating on object shape with point sets of any dimensions as described in Castleman [1996]. It is not a segmentation method itself but might help to classify pre-segmented objects by putting them through a sequence of set transformations. This is useful when features of the same density but, e.g., with different shape, should be quantified separately. Soroushian and Elzafraney [2005] propose a couple of morphological operations to identify cracks and voids in concrete, for 2D as well as for 3D microstructural images captured by environmental scanning electron and fluorescent microscopy. Some of these operations in combination with our feature size approach may offer new possibilities for multi-dimensional transfer functions.

Adaptive sampling of the Time Domain

As already explained in Section 3.5.2 the result of region growing is tracked and recorded along an entire parameter range. Therefore, the time parameter t is stepped from a start time t_0 to a maximum time t_{max} in a specific number of equidistant steps where the region growing is recalculated no matter how many modifications take place. Therefore, we want to employ adaptive sampling of the time domain where we first "scan" the entire parameter space with just a few time steps to obtain the "important" sections in time where many changes occur. In such sections we adaptively refine the stepsize for further pre-processing iterations whereas we can avoid nonessential calculations in ranges where no changes take place. Thus, we can improve the pre-processing time while a better exploration of feature classes is possible.

Advanced Rendering over Time

For the results shown above (Section 5.2) one specific timestep was sufficient to derive the desired classification. But with a wider parameter range, using adaptive sampling, the graduation in time becomes finer in places and features of different density may appear at completely different timesteps. Consequently, a single timestep t_x is no longer sufficient for all kind of feature classes. Therefore, we need to expand the feature rendering over the whole 3D feature size TF range. A semi-automatic TF generation would help to improve exploration.

Integration of Measurement Tools

As already explained in Section 5.2.8, the shape, size, orientation, and amount of inclusions have different influence on the quality of the castings. Therefore, especially in critical regions measurements of their, e.g., geometric behavior, distance to each other, orientation, etc., is of great interest. Preim et al. [2002] propose a number of tools for direct-manipulative measurements where special attention is paid on the optimal legibility of the parameters and providing enough depth cues for interaction in 3D. The following measurement tools are described in design and implementation for 3D volumes in combination with the 2D views of a slice viewer:

- **Distance Lines:** Distance lines are used to measure distances between objects and extents of objects. When segmented objects are available, snapping can be enabled to attract the endpoint by the surface of an object for easier positioning.

- **Interactive Rulers:** Rulers are also useful for estimating distances, but rather to roughly approximate the magnitude of structures, like a scale as used in maps.

- **Angular Measurements:** Angular measurements are not only useful in medical applications to define angles between anatomical structures but also suitable in NDT applications. To ease the assessment of the size of the angles, a portion of a circle is used as orientation aid and to communicate the extent of the angle.

- **Volume Approximation:** Volumes of interest are enclosed by a simple geometric shape, so the computation of voxels is restricted to a bounding volume. The structures inside this bounding volume which are not relevant for the size measurement are then suppressed by a special transfer function.

- **Automatic Definition of Object Extents:** To calculate the extent of an object principal component analysis (PCA) is used. Subsequently it can be visualized with three orthogonal distance lines derived from the PCA.

- **Automatic Definition of Angles between Objects:** For the automatic calculation of an angle between two elongated objects also PCA is used. Therefore, it is sufficient to know the eigenvector of the largest eigenvalue of both objects.

Aditionally, an automatic 3D measurement approach for already segmented objects is proposed, which could be applied to the result of the pre-computed region growing (see Section 3.5.2) to obtain better quantification results.

Region Growing

Our application offers two different simple region growing methods as a proof-of-concept implementation for its interactive approach. We would like to explore improved region growing options (see Section 2.3) in the future.

Acknowledgments

This work has been carried out at the VRVis Research Center for Virtual Reality and Visualization in Vienna (http://www.vrvis.at), and was funded by the FFG. Thanks to the VRVis for providing me a workplace and for the really nice working atmosphere.

I would like to thank everyone who made this work possible. First and foremost, I would like to thank my supervisor Markus Hadwiger for supporting me and for so many things I can't all mention here – so thank you for everything! Most of all I want to thank him for the possibility to continue my work by offering me a PhD position.

I want to thank Meister Eduard Gröller for his helpful advices and proof reading this thesis.

I want to thank Georg Geier and Daniel Habe, our project partners from the ÖGI, for the great cooperation, for providing such cool data sets, the interesting discussions and the productive workshops ;-). Special thanks to Georg who helped me with the description and analysis of the data.

I also want to thank Johanna Beyer and Florian Schulze for all helpful advices and especially for the nice time. Thanks Johanna for proof reading this thesis.

I want to thank my family, for their financial aid and mental support during the whole studies, and last but not least Peter for always supporting me and everything else.

So, thank you everybody and now let's go for a beer :-)!

References

ADAMS, R., AND BISCHOF, L. 1994. Seeded region growing. *IEEE Trans. Pattern Anal. Mach. Intell. 16*, 6, 641–647.

BAJAJ, C., PASCUCCI, V., AND SCHIKORE, D. 1997. The contour spectrum. In *Proceedings of IEEE Visualization*, 167–173.

BEYER, J., HADWIGER, M., MÖLLER, T., AND FRITZ, L. 2008. Smooth mixed-resolution gpu-based raycasting. In *IEEE/EG Symposium on Volume and Point-Based Graphics 2008*, 163–170.

BLINN, J. F. 1982. Light reflection functions for simulation of clouds and dusty surfaces. In *SIGGRAPH '82: Proceedings of the 9th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 21–29.

BORDIGNON, A. L., CASTRO, R., LOPES, H., LEWINER, T., AND TAVARES, G. 2006. Exploratory visualization based on multidimensional transfer functions and star coordinates. In *SIBGRAPI*, 273–280.

BRUCKNER, S., AND GRÖLLER, M. E. 2007. Style transfer functions for illustrative volume rendering. *Computer Graphics Forum 26*, 3, 715–724.

CASTLEMAN, K. R. 1996. *Digital Image Processing*. Prentice Hall Press.

CHAN, M.-Y., WU, Y., QU, H., CHUNG, A. C. S., AND WONG, W. C. K. 2006. Mip-guided vascular image visualization with multi-dimensional transfer function. In *Computer Graphics International*, Springer, T. Nishita, Q. Peng, and H.-P. Seidel, Eds., vol. 4035 of *Lecture Notes in Computer Science*, 372–384.

COHEN, M. F., PAINTER, J., MEHTA, M., AND MA, K.-L. 1992. Volume seedlings. In *Proc. ACM Symp. on Interactive 3D Graphics*, 139–145.

COPTY, N., RANKA, S., FOX, G., AND SHANKAR, R. V. 1994. A data parallel algorithm for solving the region growing problem on the connection machine. *Journal of Parallel and Distributed Computing 21*, 1, 160–168.

ENGEL, K., KRAUS, M., AND ERTL, T. 2001. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *HWWS '01: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, ACM Press, New York, NY, USA, 9–16.

ENGEL, K., HADWIGER, M., KNISS, J. M., LEFOHN, A. E., SALAMA, C. R., AND WEISKOPF, D. 2004. Real-time volume graphics. *Course Notes for Course #28 at SIGGRAPH 2004. SIGGRAPH 2004.*

ENGEL, K., HADWIGER, M., KNISS, J. M., REZK-SALAMA, C., AND WEISKOPF, D. 2006. *Real-Time Volume Graphics.* A K Peters, Wellesley, Mass.

FIORENTINI, S., LARRABIDE, I., AND VNERE, M. J. 2003. A simple 3d image segmentation technique over medical data. In *simposio de informaica y salud, Buenos Aires Argentina*, 21–26.

GEIER, G., HADWIGER, M., HÖLLT, T., FRITZ, L., AND PABEL, T. 2008. Interaktive Exploration und Quantifizierung von Ungänzen in komplexen Bauteilen. In *Proceedings of Industrielle Computertomografie (CT Tagung Wels)*, 103–108.

GEIER, G., PABEL, T., HADWIGER, M., HÖLLT, T., AND FRITZ, L. 2008. Interaktive Exploration von multiphasigen, mineralischen Werkstoffen mittels Computertomographie. In *Proceedings of DACH-Tagung Deutsche Gesellschaft fuer Zerstörungsfreie Prüfung.*

GUTHE, S., WAND, M., GONSER, J., AND STRASSER, W. 2002. Interactive Rendering of Large Volume Data Sets. In *Proceedings of IEEE Visualization*, 53–60.

HADWIGER, M., SIGG, C., SCHARSACH, H., BÜHLER, K., AND GROSS, M. 2005. Real-time ray-casting and advanced shading of discrete isosurfaces. *Computer Graphics Forum 24*, 3, 303–312.

HADWIGER, M., FRITZ, L., REZK-SALAMA, C., HÖLLT, T., GEIER, G., AND PABEL, T. 2008. Interactive volume exploration for feature detection and quantification in industrial ct data. *IEEE Transactions on Visualization and Computer Graphics 14*, 5.

HEINZL, C., KASTNER, J., AND GRÖLLER, E. 2007. Surface extraction from multi-material components for metrology using dual energy ct. *IEEE Transactions on Visualization and Computer Graphics 13*, 6, 1520–1527.

HLADUVKA, J., KONIG, A., AND GRÖLLER, E. 2000. Curvature-based transfer functions for direct volume rendering. In *Proceedings of Spring Conference on Computer Graphics and its Applications(SCCG)*, 58–65.

HÖLLT, T. 2007. GPU-Based Direct Volume Rendering of Industrial CT Data. Tech. rep., VRVis Research Center und Universität Koblenz-Landau.

HOROWITZ, S., AND PAVLIDIS, T. 1974. Picture segmentation by a directed split-and-merge procedure. In *Proc. Pattern Recognition*, 424–433.

HUANG, R., AND MA, K.-L. 2003. Rgvis: Region growing based techniques for volume visualization. In *Proceedings of Pacific Graphics 2003 Conference*, 355–363.

HUANG, R., MA, K.-L., MCCORMICK, P. S., AND WARD, W. 2003. Visualizing industrial ct volume data for nondestructive testing applications. In *Proceedings of IEEE Visualization*, 547–554.

KANDOGAN, E. 2001. Visualizing multi-dimensional clusters, trends, and outliers using star coordinates. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, 107–116.

KETCHAM, R. A., AND CARLSON, W. D. 2001. Acquisition, optimization and interpretation of x-ray computed tomographic imagery: applications to the geosciences. In *Computers and Geosciences*, 381–400.

KHAN, G. N., AND GILLIES, D. F. 1992. Parallel-hierarchical image partitioning and region extraction. In *Computer Vision and Image Processing*, 123–140.

KINDLMANN, G., AND DURKIN, J. W. 1998. Semi-automatic generation of transfer functions for direct volume rendering. In *VolVis98*, 79–86.

KINDLMANN, G., WHITAKER, R., TASDIZEN, T., AND MOLLER, T. 2003. Curvature-based transfer functions for direct volume rendering: methods and applications. In *Proceedings of IEEE Visualization*, 513–520.

KNISS, J., KINDLMANN, G., AND HANSEN, C. 2001. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *Proceedings of IEEE Visualization*, 255–262.

KRÜGER, J., AND WESTERMANN, R. 2003. Acceleration techniques for GPU-based volume rendering. In *Proceedings of IEEE Visualization*, 287–292.

LEILA, D., NACER, K., AND MOHAMED, B. 2008. Image segmentation by self-organised region growing. *Computer Information Systems and Industrial Management Applications*, 171–176.

LEVOY, M. 1988. Display of surfaces from volume data. *IEEE Computer Graphics and Applications 8*, 3, 29–37.

LIN, Z., JIN, J. S., AND TALBOT, H. 2001. Unseeded region growing for 3d image segmentation. In *Selected papers from Pan-Sydney Area Workshop on Visual Information Processing (VIP2000)*, ACS, Sydney, Australia, P. Eades and J. Jin, Eds., vol. 2 of *CRPIT*, 31–37.

LORENSEN, W. E., AND CLINE, H. E. 1987. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 163–169.

LUM, E. B., AND MA, K.-L. 2004. Lighting transfer functions using gradient aligned sampling. In *Proceedings of IEEE Visualization*, 289–296.

LUM, E. B., MA, K. L., AND CLYNE, J. 2001. Texture Hardware Assisted Rendering of Time-Varying Volume Data. In *Proceedings of IEEE Visualization*, 263–270.

LUNDSTRÖM, C., LJUNG, P., AND YNNERMAN, A. 2005. Extending and simplifying transfer function design in medical volume rendering using local histograms. In *Proceedings of EuroVis*, 263–270.

MAX, N. L. 1995. Optical Models for Direct Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics 1*, 2, 99–108.

NIER, E., AND ROTH, H. 2003. Innere Strukturen sichtbar machen. Tech. rep., Phoenix-xray.

ÖGI, 2008. The austrian foundry research institute. http://www.ogi.at/index_en.html.

PFISTER, H., HARDENBERGH, J., KNITTEL, J., LAUER, H., AND SEILER, L. 1999. The VolumePro real-time ray-casting system. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 251–260.

PFISTER, H., LORENSEN, W. E., BAJAJ, C. L., KINDLMANN, G. L., SCHROEDER, W. J., AVILA, L. S., MARTIN, K., MACHIRAJU, R., AND LEE, J.

2001. The transfer function bake-off. *IEEE Computer Graphics and Applications 21*, 3, 16–22.

PHONG, B. T. 1975. Illumination for computer generated pictures. *Commun. ACM 18*, 6, 311–317.

POHLE, R., AND TOENNIES, K. D. 2001. A new approach for model-based adaptive region growing in medical image analysis. *Lecture Notes in Computer Science 2124*, 238–246.

PREIM, B., TIETJEN, C., SPINDLER, W., AND PEITGEN, H. O. 2002. Integration of measurement tools in medical 3d visualizations. In *VIS '02: Proceedings of the conference on Visualization '02*, IEEE Computer Society, Washington, DC, USA, 21–28.

REINHARD, C., POLIWODA, C., GUENTER, T., ROEMER, W., MAASS, S., AND GOSCH, C. 2004. Modern voxel based data and geometry analysis software tools for industrial ct. Tech. rep., Volume Graphics GmbH, Heidelberg, Germany.

REZK-SALAMA, A., ENGEL, K., BAUER, M., GREINER, G., AND ERTL, T. 2000. Interactive volume rendering on standard pc graphics hardware using multi-textures and multi-stage rasterization. In *Proceedings Eurographics/SIGGRAPH Workshop on Graphics Hardware 2000 (HWWS00).* Best Paper Award.

REZK-SALAMA, C., KELLER, M., AND KOHLMANN, P. 2006. High-level user interfaces for transfer function design with semantics. *IEEE Transactions on Visualization and Computer Graphics 12*, 5, 1021–1028.

ROETTGER, S., GUTHE, S., WEISKOPF, D., ERTL, T., AND STRASSER, W., 2003. Smart hardware-accelerated volume rendering.

SCHARSACH, H., HADWIGER, M., NEUBAUER, A., WOLFSBERGER, S., AND BÜHLER, K. 2006. Perspective isosurface and direct volume rendering for virtual endoscopy applications. In *EuroVis*, 315–322.

SELLE, D., PREIM, B., SCHENK, A., AND PEITGEN, H. 2002. Analysis of vasculature for liver surgical planning. In *IEEE Transactions on Medical Imaging*, vol. 21, 1344–1357.

ŠEREDA, P., BARTROLI, A. V., SERLIE, I. W., AND GERRITSEN, F. A. 2006. Visualization of boundaries in volumetric data sets using LH histograms. *IEEE Transactions on Visualization and Computer Graphics 12*, 2, 208–218.

ŠEREDA, P., 2004. The transfer function design for volume data rendering.

SHREINER, D., WOO, M., NEIDER, J., AND DAVIS, T. 2005. *OpenGL programming guide: the official guide to learning OpenGL, version 2*, fifth ed. Addison-Wesley, Reading, MA, USA.

SMITH, S. W. 1997. *The scientist and engineer's guide to digital signal processing*. California Technical Publishing, San Diego, CA, USA.

SONKA, M., HLAVAC, V., AND BOYLE, R. 1993. *Image Processing, Analysis, and Machine Vision*. Chapman and Hall Computing.

SOROUSHIAN, P., AND ELZAFRANEY, M. 2005. Morphological operations, planar mathematical formulations, and stereological interpretations for automated image analysis of concrete microstructure. *Cement and concrete composites 27*, 7, 823–833.

STEGMAIER, S., STRENGERT, M., KLEIN, T., AND ERTL, T. 2005. A simple and flexible volume rendering framework for graphics-hardware-based raycasting. In *Volume Graphics*, 187–195.

TAKAHASHI, S., TAKESHIMA, Y., AND FUJISHIRO, I. 2004. Topological volume skeletonization and its application to transfer function design. *Graph. Models 66*, 1, 24–49.

TAPPENBECK, A., PREIM, B., AND DICKEN, V. 2005. Distanzabhängige Transferfunktionen für die medizinische Volumenvisualisierung. In *Bildverarbeitung für die Medizin*, 307–311.

TENGINAKAI, S., LEE, J., AND MACHIRAJU, R. 2001. Salient iso-surface detection with model-independent statistical signatures. In *Proceedings of IEEE Visualization*, 231–238.

THÉVENAZ, P., AND UNSER, M. 2003. Precision isosurface rendering of 3-D image data. *IEEE Transactions on Image Processing 12*, 7 (July), 764–775.

TZENG, F.-Y., LUM, E. B., AND MA, K.-L. 2003. A novel interface for higher-dimensional classification of volume data. In *Proceedings of IEEE Visualization*, 66–73.

VAN KREVELD, M., VAN OOSTRUM, R., BAJAJ, C., PASCUCCI, V., AND SCHIKORE, D. 1997. Contour trees and small seed sets for isosurface traversal. In *Proc. ACM Symp. on Computational Geometry*, 212–220.

VGSTUDIOMAX, 2008. Volume graphics–vgstudio max. `http://www.volumegraphics.com`.

WILLEBEEK-LEMAIR, M., AND REEVES, A. 1990. Solving nonuniform problems on simd computers: Case study on region growing. *Journal of Parallel Distributed Computing 8*, 135–149.

WILSON, O., GELDER, A. V., AND WILHELMS, J. 1994. Direct Volume Rendering via 3D-textures. Tech. Rep. UCSC-CRL-94-19, UCSC.

ZHOU, F. J., DÖRING, A., AND TÖNNIES, K. D. 2004. Distance transfer function based rendering. Tech. rep., International Telegraph and Telephone Consultative Committee.

ZHU, S. C., AND YUILLE, A. 1996. Region competition: Unifying snakes, region growing, and bayes/mdl for multiband image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 18*, 884–900.

ZUCKER, S. 1976. Region growing: Childhood and adolescence. *Computer Graphics and Image Processing 5*, 382–399.