FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Advanced Measurement and Quantification of Industrial CT Data

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Computergraphik & Digitale Bildverarbeitung

eingereicht von

Fritz-Michael Gschwantner
Matrikelnummer 0627004

an der
Fakultät für Informatik der Technischen Universität Wien

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

Wien, 21.10.2011

(Unterschrift Verfasser) (Unterschrift Betreuung)

Erklärung zur Verfassung der Arbeit

Fritz-Michael Gschwantner
Erlgasse 20/7, 1120 Wien

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

(Ort, Datum) (Unterschrift Verfasser)

Acknowledgements

My thanks go to all the ponies of Equestria. Fluttershy, Pinkie Pie, Twilight Sparkle, Rainbow Dash, Rarity, Applejack, to name a few, but also especially Lyra. They were always an inspiration and everypony helped me get through the rough time during the creation of this thesis.

I am also grateful for the help, patience and understanding of all the people I worked with or used to work with at the VRVis Research Center and the Vienna University of Technology, especially Laura Fritz, Markus Hadwiger and Meister Eduard Gröller, but also Johanna Schmidt, who joined the team later on for the duration of her thesis.

Abstract

Non–destructive testing (NDT) is a vital part in today's industrial production and research processes. Such testing procedures often use Computed Tomography (CT) in order to get insights of the inner parts of an object. During the analysis of different objects, certain features can be automatically segmented and quantified in the CT dataset. However, due to various effects during the acquisition of the data, the original boundaries of two materials within the objects are not accurately represented in the dataset.

This thesis describes a method to reconstruct these boundaries for automatically segmented features on a subvoxel level of the dataset. They are searched along the gradient of the data, using an edge–detection approach commonly used in image processing. The result is then represented as a distance field and further quantified through over–sampling and measuring.

For a variety of datasets it is shown that these reconstructed boundaries are indeed providing a more accurate representation of the original segmented region. Further comparisons are made with a method that simply tries to improve the visual appearance through smoothing.

Kurzfassung

In modernen industriellen Produktions– und Forschungsprozessen ist die zerstörungsfreie Werkstoffprüfung (ZfW) ein wichtiger Bestandteil. Dabei wird oft Computer Tomographie (CT) eingesetzt, um Einblicke in das Innere des Objektes zu bekommen. In solchen Datensätzen können bestimmte Merkmale automatisch erkannt und quantifiziert werden. Allerdings werden darin die ursprünglichen Grenzen zwischen zwei Materialien nicht mehr korrekt abgebildet.

In dieser Diplomarbeit wird ein Verfahren vorgestellt, womit diese Grenzen aus dem realen Objekt wieder hergestellt werden können. Diese Rekonstruktion basiert auf einem einfachen Prinzip zur Erkennung von Kanten, wie sie auch in der Bildverarbeitung eingesetzt wird. Diese Grenze wird entlang des Gradienten der Daten gesucht und das Ergebnis in einem sogenannten Distance Field abgebildet.

Aus diesen Ergebnissen wird anschließend das genaue Volumen einzelner Merkmale durch Over–Sampling berechnet. Für verschiedene Arten von Datensätzen wird gezeigt, dass die Rekonstruktion dieser Grenze auf Subvoxel–Ebene einer genaueren Darstellung entspricht, als die zuvor segmentierte Region selbst. Diese Ergebnisse werden außerdem mit einer Methode verglichen, die durch Glättung die Darstellung der ursprünglichen Region verbessert.

Contents

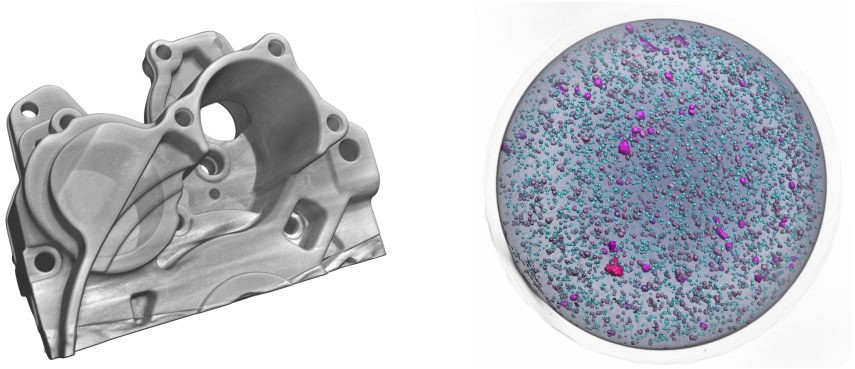
CHAPTER 1

Introduction

To improve the quality of products and determine defects within them, many different fields in the industry use testing techniques as part of their research and manufacturing process. The automotive industry, heavy machinery industry and plant industry often deal with different *metal casts* for example. Depending on the type of metal or alloy and the casting process itself, various kinds of defects in a cast body can occur. Such properties have an influence on the stability and usefulness of the object. The analysis of these defects can be used to improve the production process and the overall quality of the product. Other industrial fields are also interested in the inner structures of their products. For instance the hard rubber core of golf balls are reinforced with more dense particles and the resulting distribution of these particles during the production process are an important property during the analysis. Another example is steel fibre reinforced sprayed concrete in the building industry. The overall load–bearing capacity, durability and impact resistance of this type of sprayed concrete is mainly defined by the choice of fibres and their distribution and alignment within the concrete.

Obtaining such a view of the inner structures of any object can be done by cutting it open. However, testing an object in such a destructive manner is costly and very time consuming, since it would need to be ablated layer by layer for a full analysis. Therefore *Non-Destructive Testing* (NDT) techniques are often the method of choice for the inspection of such objects. NDT describes a variety of analysis techniques that can be used to determine certain properties of materials without actually causing damage to them. These methods include liquid penetrant inspection, ultrasonic and radiographic scanning as well as *Computed Tomography* (CT).

This work mainly deals with industrial CT data provided by the *Austrian Foundry Research Institute* (ÖGI). The objects scanned by an industrial CT scanner at the ÖGI range from products or workparts of various industries which need material testing, to cast parts created in their own pilot foundry. In this pilot foundry test castings can be created with all cast–materials such as cast irons, steels, copper alloys, Al– and Mg–alloys [44]. These casts can then be tested in a mechanical testing laboratory. Through NDT the behavior, properties and possible defects of such test castings can be evaluated at different stages of the testing process. Figure 1.1(a) shows



(a) Cast housing (b) Golf ball

Figure 1.1: Different industrial CT datasets. (a) A cast housing of an engine block part. (b) A scan of a golf ball with feature visualization. Image from Hadwiger et al. [20].

an example of a dataset of a cast housing, while Figure 1.1(b) shows the visualization of dense particles within a scanned golfball.

1.1 Problem Statement and Objectives

Exploration and quantification of three–dimensional industrial CT data is often the method of choice during NDT. The visualization of such volumes has become a large research field in the recent decades, making it possible to explore them in real–time. Apart from visualization, many different methods for automatically extracting important information for the domain expert have been devised as well. In the context of NDT, this *feature extraction* is mostly done through a preliminary segmentation step using region growing, water shedding, morphological filter operations or applying local thresholds for instance. These methods provide a binary classification of individual volume voxels into segmented regions, meaning that a whole voxel either belongs to the region of a specific feature or not.

However, in reality the boundary between two materials is almost never exactly between two adjacent voxels, nor does the actual form of the feature in the real object follow the form generated by the binary segmentation. Due to the inherently limited resolution and other side effects of a CT scan, the real object boundary cannot be represented accurately and without artifacts. The binary segmented region of such features then appears as a heavily aliased set of voxels, when used in direct volume rendering. In many cases this sort of segmentation and representation of the features is enough, in order to get a rough estimate of their volumes, their distribution and extents. In other fields a better visualization of such segmented regions is required, thus different smoothing techniques are often applied.

If more accurate quantifications of the data are required, such techniques are not suitable, since the smoothing process happens without any direct connection to the original, underlying data. To get more accurate results, it would be necessary to extract additional information of the available data, at a fractional level within a voxel, i.e. on a *subvoxel* level. By analyzing

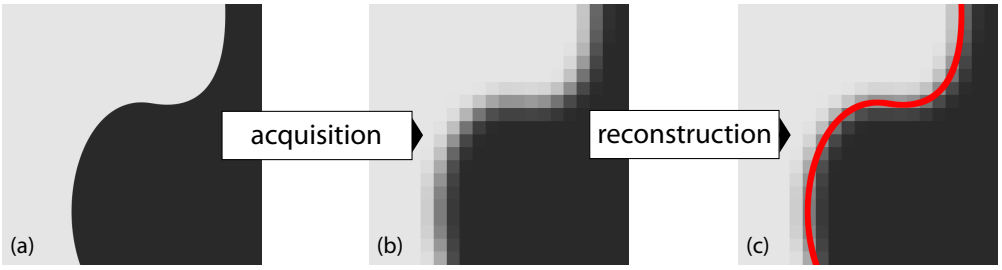


Figure 1.2: The main objective of this work: reconstructing the original subvoxel boundary, that is lost during the data acquisition and using it for advanced quantification purposes.

the acquisition process of CT data and how the original object boundary between two materials might be mapped into the final volume data, the objective is to devise a method that actually reconstructs this boundary again (Figure 1.2). This includes determining the subvoxel boundary and representing it in a suitable data format. Using these results, additional objectives are to be able to take quantifications and measurements on a subvoxel level. It should be possible to calculate the fractional voxel volume of the segmented regions and take manual measurements between these subvoxel boundaries.

1.2 Pipeline Overview

The methods presented in this thesis are implemented in the *High–Quality Hardware Volume Renderer* framework (HVR), at the *VRVis Research Center*. This framework already features an extensive segmentation and exploration method, described in more detail in Hadwiger et al. [20]. This method will be applied, to automatically segment the volume into features. After this step, the results from the feature detection can be explored, to choose the optimum segmentation result and optionally select only specific features of interest for further processing. The next step tries to find the subvoxel boundary, based on these segmentations. This method combines different techniques and principles: the boundary is searched along a path defined by the gradient vector field of the data and a *distance transform* of these subvoxel locations is stored in a *distance field volume*. This search along the gradient direction is a principle also used by Šereda et al. [58], while the actual determination of the subvoxel boundary is basically an edge–detection principle as often used in 2D image processing.

This resulting distance field can then be used for quantifying the subvoxel volume of individual features, as well as for helping with manual measurements of the data. For the latter part, simple measurement tools will be integrated into the framework. Figure 1.3 illustrates the basic work flow.

1.3 Structure of the Work

This thesis is organized as follows: Chapter 2 will first provide some basic fundamentals of the main categories used in this thesis. Since this work is mainly concerned with the analysis of

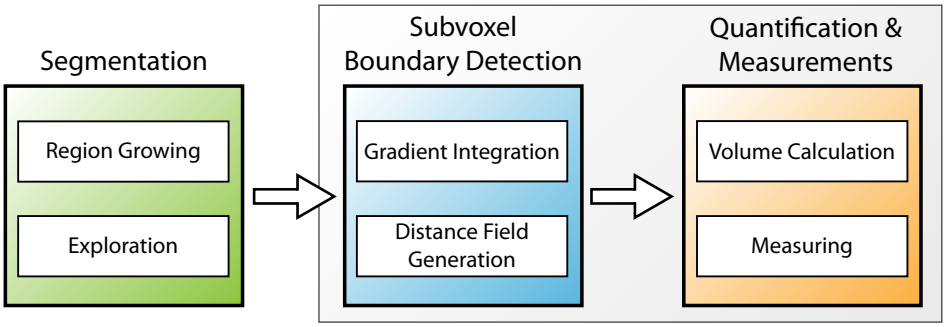


Figure 1.3: The basic work flow pipeline presented in this thesis. Based upon a preliminary segmentation of the volume data, the subvoxel boundary for these segmented features is determined. A distance field will be generated, representing this boundary, which will be used for further quantifications and measurements.

industrial CT data, Computed Tomography in general, as well as specific differences of industrial CT data will be explained. The second part of this chapter covers the method used for visualizing these datasets: GPU–based direct volume rendering. A little more details on principles and used techniques for distance fields are provided in the following section. These distance fields are reconstructed with high–quality cubic filtering on the GPU. The technique used to achieve this reconstruction is also covered in this chapter. The last part covers a basic state of the art on some quantification and processing techniques on a subvoxel level.

Chapter 3 covers the methodology for finding and storing the subvoxel boundary. Assumptions as well as the involved mathematical properties of the data are discussed. The algorithm and actual implementation is then described in more detail. Finally a mesh smoothing technique and its implementation is also described, which is used as a point of comparison later on. Chapter 4 describes the second part of the aforementioned pipeline, the quantification and measurement process. A method for determining the subvoxel volume directly from the distance field is presented, as well as the integration of measurement tools in the framework and their possible uses.

Results from the subvoxel boundary detection and volume quantification are then presented in Chapter 5. A comparison between original binarily segmented regions and the detected boundary is provided for specific features of some industrial CT datasets. Issues and limitations of the work are also discussed in this chapter. The thesis then closes with a conclusion and some possible suggestions for future work in Chapter 6.

CHAPTER 2

State of the Art

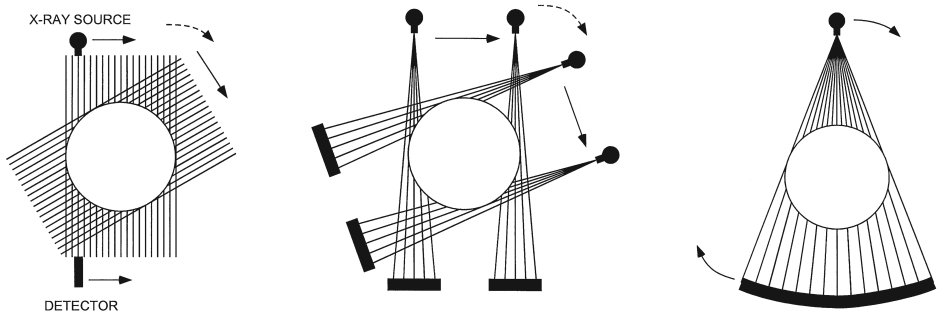
This chapter covers the basic fundamentals and current state of the art of important topics and practices that are discussed in this thesis. Section 2.1 will talk about *Computed Tomography* (CT) in general, including differences between medical and industrial CT. Section 2.2 gives an overview of visualization techniques for volumetric data and goes into detail about the technique that is mainly used in the *High–Quality Hardware Volume Renderer* (HVR), the framework for this thesis. Distance Fields are used in the volume quantification process of this thesis and are discussed in Section 2.3. Furthermore, high quality cubic filtering for reconstruction of the distance field during rendering and volume sampling (see Section 4.1 and Chapter 5) is employed. Section 2.4 describes the technique that is used for filtering on the GPU in this thesis. Section 2.5 describes some approaches to determine boundaries or volumes on a subvoxel level.

2.1 Industrial CT

In *classical tomography*[1] a 2D image is obtained of a 3D object with X-rays or other penetrating radiation. In the resulting image interior parts of the object overlap with each other. In order to obtain actual 2D slices through these objects, Computed Tomography was developed in the early 1970s and has revolutionized the X-ray field ever since [55]. CT is now an important tool for visualizing interior features of objects in a non-destructive way in medical and industrial applications as well as in geosciences for instance. The key differences between medical and industrial CT are outlined in Ketcham and Carlson [27]:

- Objects of industrial CT have no limitations on acceptable radiation dose. Thus high energy X-rays can be used at longer exposure times.
- Higher energies and longer exposure times lead to smaller detectors, which increase the resolution of the scan.

[1] *tomography* means *„a picture of a plane“*.



(a) first generation CT (b) second generation CT (c) third generation CT

Figure 2.1: Different generations of CT system configurations. (a) first generation with moving pencil beam X-ray source and moving detector. (b) second generation with moving fan-beam X-ray source and moving detector. (c) third generation with fan beam X-ray source that covers the whole object and a single detector array. Images from Ketcham and Carlson [27].

- Inanimate objects of industrial CT can be positioned more precisely for the scan.

After sample preparation and machine calibration there are two main steps in CT imaging: *data acquisition* and *data reconstruction*.

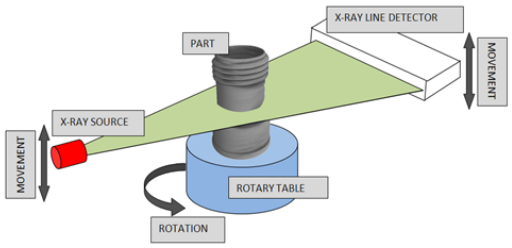
2.1.1 Data Acquisition

X-ray images are obtained through the attenuation of the ray as it passes through an object. This attenuation is mainly caused by three different physical processes: photoelectric absorption, Compton scattering and pair production. These effects depend on the initial energy of the X-ray as well as the atomic number of the material, through which the ray passes.

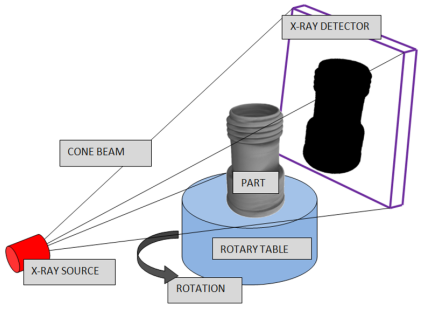
Every configuration of X-ray radiography consists of at least these three elements: an X-ray source, an object to be scanned and a series of detectors that measure the intensity of the X-rays, which have been attenuated as they pass through the object. The basic principle of CT however is to acquire multiple *views* of an object over a range of angular orientations. The number of these views can range from 600 to 3600 or more.

There exist four generations of CT system, three of which are outlined in Figure 2.1. The rotational movement of these setups is relative: either the the X-ray source and detector rotate around the object or the object itself rotates within the setup.

Industrial CT scanners are mostly custom built and thus can differ in their exact implementation. However, typically a setup like in Figure 2.2 is used, where the object is rotated and the radiation source is either a fan beam or a cone beam. In the former case (Figure 2.2(a)), a fan beam is used that is wide enough to encompass the whole object. Either the object or the radiation source and detector are moved vertically, to obtain a 2D image for each angle. In the latter case (Figure 2.2(b)) the cone beam encompasses the whole object and creates a full 2D image for one direction.

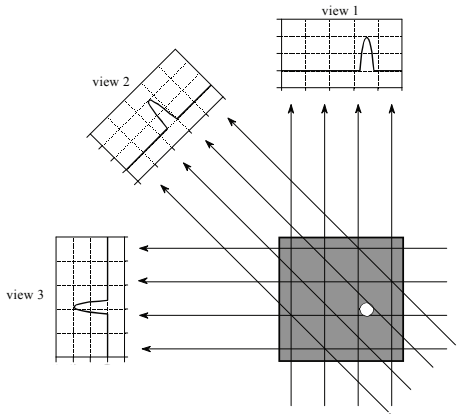


(a) Industrial CT with fan beam.

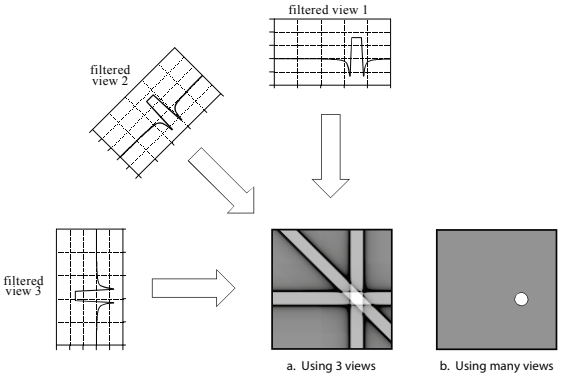


(b) Industrial CT with cone beam.

Figure 2.2: (a) Data acquisition with a fan beam by rotating the workpart 360° and moving either the x-ray emitter and detector vertically or the object during one rotation step. This leads to a 2D image after one step. (b) Data acquisition with a cone beam by rotating the object 360°. A full 2D image is acquired during each rotational step. Images from [25, 60].



(a) Resulting views of CT data acquisition.



(b) Data reconstruction with filtered backprojection.

Figure 2.3: (a) Set of views acquired during CT (usually hundreds or thousands). Each data value in a view represents the sum of image values along the ray for that sample. (b) Reconstruction of a slice using backprojection. Each view is filtered in a pre–processing step and then each sample of each view is smeared along its original path. Images from Smith [55].

2.1.2 Data Reconstruction

Different methods exist in order to obtain the actual slice image of a given set of views. The most important one is *filtered backprojection* [55]. During backprojection, each sample of each view is smeared along the path of its acquisition angle. However, in order to avoid blurring of the final image, the values of each view are first filtered. This results in a mathematically correct reconstruction of the slice image (Figure 2.3).

The result of data reconstruction of CT images yields a series of 2D slices through the

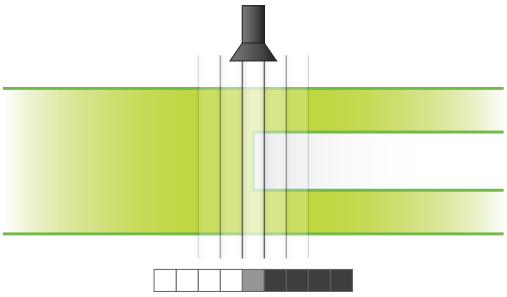


Figure 2.4: Illustration of a single view for a CT scan through an object with different materials. The limited resolution of the detector causes several x-rays that pass through the volume to be accumulated and averaged. Thus the material boundary inside the object is not accurately sampled in this case.

scanned object or a *3D Volume* with all slices stitched together.

2.1.3 Partial Volume Effects

CT data is often subject to certain artifacts like *beam hardening*, *ring artifacts* and a variety of other artifacts. An important artifact to note for this thesis is the so called *Partial Volume Effect* (PVE). Each pixel in a CT image represents the attenuation properties of material volume. If that volume consists of several different substances, the resulting CT value will actually be an average of these properties (see Figure 2.4).

The PVE describes an artifact that is common to all systems that attempt to represent a continuous signal with a finite number of samples [11]. Due to the inherent limitation in resolution of X-ray CT, all material boundaries are blurred to a certain degree. While it is undoubtedly desirable to have high resolution CT scans so that all structures within the object are accurately sampled, the PVE also provides an opportunity to extract information on a sub–pixel or sub–voxel level.

However, correct interpretation of values containing more than one material is not trivial and is still an important subject of research today.

2.2 Volume Rendering

During recent decades, several techniques have been developed to create two–dimensional visualizations of three–dimensional scalar fields. Some methods require pre-processing of these volumes in order to extract geometric primitives, like marching cubes [37, 46] or isosurfaces [57], however a more widely used approach is *Direct Volume Rendering* (DVR). Considerable work has been made to implement GPU-based instead of CPU-based approaches, in order to exploit the characteristics of modern graphics hardware and thus generate visualizations at interactive frame rates.

Generally, volume rendering tries to simulate the interaction between light and the participating media. The basic emission–absorption model is described as the following integral:

$$I(D) = I_0\, e^{-\tau(s_0,D)} + \int_{s_0}^{D} q(s')\, e^{-\tau(s',D)}\ \mathrm{d}s' \tag{2.1}$$

where s is the parameterized path of the light with s_0 being the starting point of the light flow and D the endpoint. I_0 is the light intensity that enters the volume from the background, $I(D)$ the radiance that reaches the camera and q the emission coefficient. τ represents the optical depth and is defined as $\tau(s_1, s_2) = \int_{s_1}^{s_2} \kappa(s)\, \mathrm{d}s$ where κ is the light absorption coefficient [21]. Since this integral typically cannot be evaluated analytically, numerical methods are devised to approximate the result as close as possible. Further reading is provided by Max [38] on different optical models for volume rendering and by Hege et al. [21] for optical models and other algorithmic aspects.

Overview of Rendering Methods

We can classify volume rendering techniques as either *image-order* or *object-order* methods. Image–order approaches use the 2D image plane of the result as a starting point for volume traversal, while object–order methods try to project individual data points of the volume onto the image plane. The most popular image–order method is *ray casting*, where for every pixel of the final image a ray is cast through the volume, compositing the volume rendering integral in an iterative fashion from sample points along this ray (Figure 2.5(a)). Since the rays are conceptually started at the camera, the traversal order of ray casting is front-to-back and uses the following equations to accumulate the colour and opacity at each sample point:

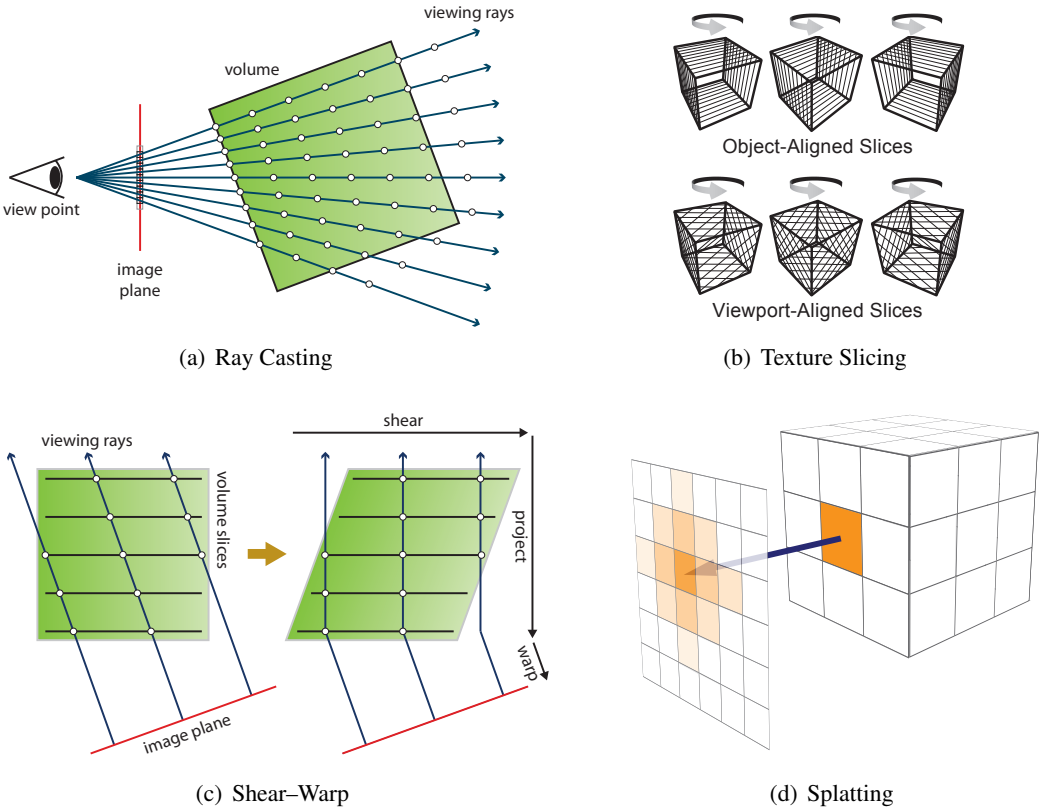
$$\begin{aligned} C_{\mathrm{dst}} &\longleftarrow C_{\mathrm{dst}} + (1 - \alpha_{\mathrm{dst}}) C_{\mathrm{src}} \alpha_{\mathrm{src}} \\ \alpha_{\mathrm{dst}} &\longleftarrow \alpha_{\mathrm{dst}} + (1 - \alpha_{\mathrm{dst}}) \alpha_{\mathrm{src}} \end{aligned} \tag{2.2}$$

with C_{dst} and α_{dst} being the accumulated colour and transparency values from the previous sample points of the current fragment and C_{src} and α_{src} the colour and transparency of the currently evaluated sample point.

Ray casting is a classic CPU based approach, introduced by Levoy [36], over 20 years ago. However recent development in computer graphics hardware made it possible to create GPU based implementations, first used by Roettger et al. [49] and Krüger and Westermann [30] with multiple passes initiated by the CPU. Since the introduction of shader model 3.0 it was possible to implement single–pass approaches as these GPUs support loops as well as dynamic branching in the fragment shader.

Before that, the dominant form of GPU based volume rendering was *texture slicing*, an object-order approach where 2D slices in object space are used to sample the volume. These 2D slices can be rendered using regular texture mapping on the GPU and are blended together in either a front–to–back or back–to–front fashion. The proxy geometry for these slices is either aligned to the view port or along the object space axis (Figure 2.5(b)).

Similar to texture slicing is *shear–warp volume rendering*. Here, the volume is projected onto an intermediate image plane and the volume itself is sheared in order to transform the tilted



(a) Ray Casting

(b) Texture Slicing

(c) Shear–Warp

(d) Splatting

Figure 2.5: Short overview of modern direct volume rendering techniques. (a) DVR by casting a ray through the volume for each fragment of the 2D image and accumulating the value at regular sample positions. (b) DVR by blending 2D object– or view–aligned slices of the volume together for the final image. Images from Rezk-Salama et al. [47]. (c) DVR by shearing 2D slices of the volume and warping the projected final image. (d) DVR by projecting a 3D reconstruction kernel onto the image plane for each voxel (object-order approach).

projection direction into a perpendicular one. This allows a faster implementation of the projection. The image of the intermediate plane is then warped to the final image plane (Figure 2.5(c)). More details can be found in the original paper by Lacroute and Levoy [33], which covers the case for an orthogonal projection. An extension to a perspective projection is described by Schulze et al. [52].

Another object–order approach is *splatting*. The idea is to project 3D reconstruction kernels onto the image plane for each data point (Figure 2.5(d)). The resulting 2D image of such a kernel is called a footprint [59]. Splatting can be used in a wide variety of data layouts. It can be used for uniform grid volumes as well as for scattered data like point clouds for example.

2.2.1 GPU–Based Ray Casting

Object–order approaches like texture slicing are easy to implement, however they suffer from some significant disadvantages. With increasing complexity of the volume data set, their performance degrades greatly, since the number of slices are directly determined by it. Therefore it is desirable to use algorithms that are sensitive to the output instead, like an image–space approach which can account for the complexity of the generated image.

That being said, ray casting provides some significant advantages for direct volume rendering. Each fragment of the final image plane creates a ray that evaluates the volume, which means that no oversampling of the result takes place. Instead the number of samples taken along each ray directly relates to the quality of the end result. Each ray can also be computed independently, which means there is a potential for performance improvement, if the computation can be run in parallel. The basic algorithm for ray casting is outlined in Algorithm 2.1.

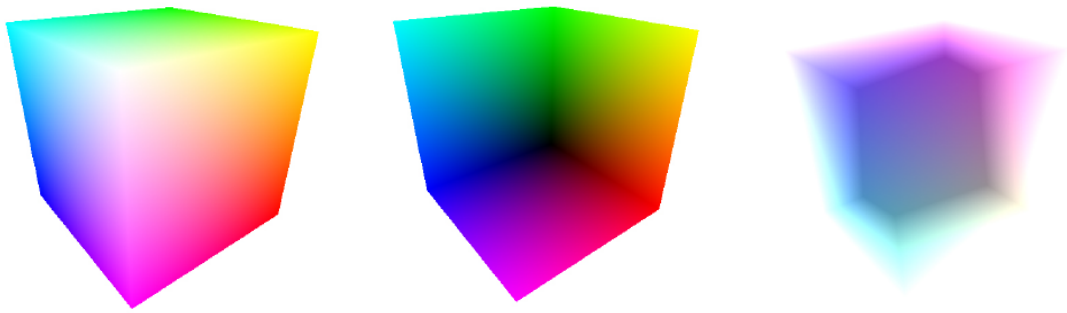
```
1 Determine volume entry position
2 Compute ray direction
3 while ray position in volume do
4 |   Access data value at current position
5 |   Compositing of color and opacity
6 |   Advance position along ray
7 end
```

Algorithm 2.1: Pseudo code for ray casting [19].

GPU–based ray casting takes advantage of several characteristics of modern graphics hardware. Volumes with a uniform grid can be directly represented as 3D textures while the computations exploit the built-in parallelism for GPU fragment processing through multiple pixel– or shader–pipelines. Sampling and traversal of the volume can be implemented through programmable shaders that are executed for each fragment.

Basic setup

Each fragment needs to know its volume entry position and direction, which are computed in two preliminary rendering passes. First the front faces of a volume bounding box are rendered, with the correct view transformation and perspective according to the camera position. The texture coordinates of the vertices of the bounding box represent the texture coordinates of the 3D volume texture. The GPU will interpolate these values and generate the entry positions as texture coordinates in the final image (Figure 2.6(a)). The direction for each ray is computed in a second render pass with only the back faces of the volume bounding box drawn (Figure 2.6(b)). During this pass the volume coordinates of the front faces are subtracted from the back faces. The length of the resulting value (i.e. the length of the ray within the volume) is stored in the alpha channel of the output, while the normalized direction is stored in the colour (Figure 2.6(c)).



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

Figure 2.6: Results of preliminary rendering passes for ray casting. (a) Front faces of the volume bounding box, with texture coordinates as starting positions (rendered as colour). (b) Back faces of the volume bounding box, with texture coordinates of the exit position for each ray. (c) Direction containing the normalized difference between front and back faces and the length of the ray within the volume encoded in the alpha channel. Images from Scharsach et al. [51].

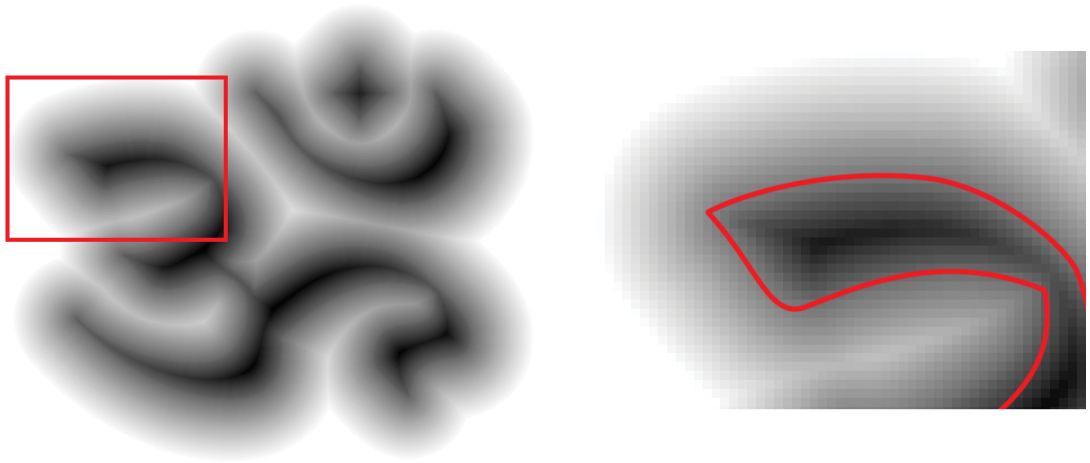
Traversal loop

The main component of the basic ray casting algorithm (see Algorithm 2.1) is the traversal loop, which is implemented as a fragment shader. By drawing a screen–sized rectangle, ray casting for each fragment will be triggered. The fragment shader receives the results of the previous passes as texture input parameters, as well as a transfer function for classification and a step size for advancing the sample position within the main loop.

At each sample position, different properties of the volume are determined, like the density or the local gradient for example. This might involve a reconstruction filter and can be done either through the capabilities of the graphics hardware with nearest neighbour or trilinear interpolation, or by using cubic filtering for example. The latter is computationally intensive, but there are some techniques to achieve interactive frame rates on today's graphics hardware (see Section 2.4).

Through *classification* these volume properties are mapped to a colour and opacity for each sample point. This is done by applying a *transfer function* which can be simple, one-dimensional mappings of the density value or the gradient magnitude for example, to a colour and opacity. However, multi–dimensional transfer functions play an important role in complex visualization and exploring tasks today. Many different types of volume properties have been explored for their use in different types of transfer functions, as well as their implementation in user interfaces, especially for complex, higher dimensional transfer functions. Kniss et al. [29] give a basic overview and interaction techniques for multi-dimensional transfer functions. Šereda et al. [58] presented a novel method to visualize material boundaries within volumetric data sets by using a two-dimensional transfer function on a so called *LH–Histogram* [58].

The computed colour and opacity is accumulated according to the front–to–back composit-



(a) (b)

Figure 2.7: A 2D distance field of an „Aum“ glyph. (a) The complete distance field, with white being „outside“ the glyph, black being „inside“. (b) Detail of the distance field, with an iso-contour rendered at the original object's boundary through reconstruction of the distance field (red line). Images from Rideout [48].

ing equation (Equation 2.2). In order to terminate a ray early, the main traversal loop is exited, once the opacity reaches 100% or is close to it. Otherwise the loop will be exited once the ray leaves the volume boundaries.

2.3 Distance Fields

In a *distance field* each value represents the known distance from that point to the closest point on the surface of any object within the domain of the field. Usually a *signed distance field* data set D will be computed, representing a surface S which is defined as:

$$\begin{aligned} D(p) &= sgn(p) \cdot min\{|p-q| : q \in S\} \\ sgn(p) &= \begin{cases} -1 & \text{if } p \text{ inside} \\ +1 & \text{if } p \text{ outside} \end{cases} \end{aligned} \tag{2.3}$$

where p is the current grid point in the distance field and q is a point on the surface S that is closest to p. $sgn(p)$ determines the sign of the distance from p to q, depending on whether p is inside the object described by the surface S or outside.

A distance field could represent two–dimensional objects with $D : \mathbb{R}^2 \rightarrow \mathbb{R}$ and $p \in \mathbb{R}^2$ or three-dimensional objects with $D : \mathbb{R}^3 \rightarrow \mathbb{R}$ and $p \in \mathbb{R}^3$. Distance fields have appeared as early as 1966 in an image processing paper by Rosenfeld and Pfaltz [50]. Its possible applications have increased ever since, ranging from high quality text representation [18, 48], virtual

endoscopy or skeletal representation [63, 64], morphing [13] as well as collision detection or the processing and analyzing of meshes. Frisken and Perry [15] also propose distance fields for use in interactive design systems, due to their volumetric as well as high accuracy surface representation. Figure 2.7 shows an example of 2D distance field representing an „Aum“ glyph, with the original contour of the glyph reconstructed in a close–up.

Generally, distance fields provide the most suitable anti-aliased representation of geometric objects for the purposes of volume graphics [26]. They can represent objects within the domain of the volume at a sub-voxel precision level, rather than just a binary classification of individual voxels for instance. The process of generating volumetric data from other three–dimensional object representations is commonly called *voxelization*.

Generating such a distance field is often not trivial, depending on the object and domain. A simple algorithm would be to compute all distances to all objects that comprises the surface from every voxel within the volume and only store the shortest distance. This approach can be impractical though when dealing with either large volumes or complex objects, or both, since it would lead to extremely long computation times. Therefore, different techniques are used to exploit spatial coherences and to approximate the distance values farther from the surfaces, if only the original surface region is of interest.

Triangle meshes are a common, wide spread representation for geometric objects. Within the context of volume graphics, it is often desired to convert these objects to volumetric data. In order to generate volume data from a triangle mesh, it is important that it divides its space into an interior and exterior, aside from the mesh surface itself. Bærentzen and Aanæs [6] list these requirements for such meshes:

- The mesh does not contain any self intersections: Triangles may share only edges and vertices and must be otherwise disjoint.
- Every edge must be adjacent to exactly two triangles.
- Triangles incident on a vertex must form a single cycle around that vertex.

These conditions are important in order to determine the *signed distance*, i.e. in order to classify the distance to the surface for a voxel as either outside or inside the object. Most of the time triangle meshes do not fulfill these requirements. Such meshes will generate errors in the result of the calculation, however in many cases they might not be noticeable.

When computing the signed distance field for a triangle mesh, the distance from a point p (voxel) to a triangle needs to be computed. This is done by projecting p onto the plane containing the triangle. This projected point p' will lie in one of the regions depicted in Figure 2.8. If p' lies within region R1, then the point is closest to the plane of the triangle. The distance is then simply calculated as $||p - p'||$. If p' lies in R2, R3 or R4, the distance from p to the closest point on the corresponding edge needs to be calculated. If p' lies within R5, R6 or R7, the distance from p to the corresponding vertex needs to be calculated. As mentioned earlier, the brute-force method would require a long computation time, when dealing with large datasets, since it requires $N \cdot M$ steps, with N being the number of voxels and M being the number of triangles. Therefore hierarchical organizations are often used during computation, or other optimizations

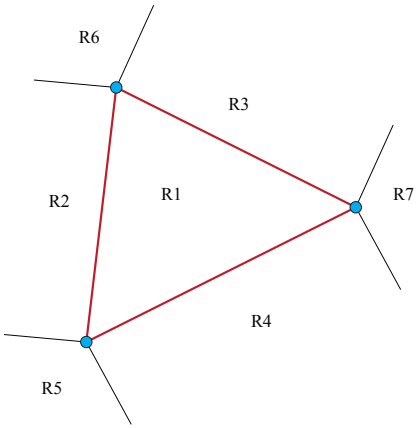


Figure 2.8: Calculating the distance to a triangle: A point p is projected onto the plane of the triangle. If p lies within region R1 it is closest to the plane. If it is inside R2, R3 or R4 it is closest to an edge. If it lies within R5, R6 or R7 it is closest to a vertex.

that depend on the desired output (e.g. when the distance field only needs to be exact near the surface of the original object). Jones et al. [26] provide an overview over many approaches that speed up the process.

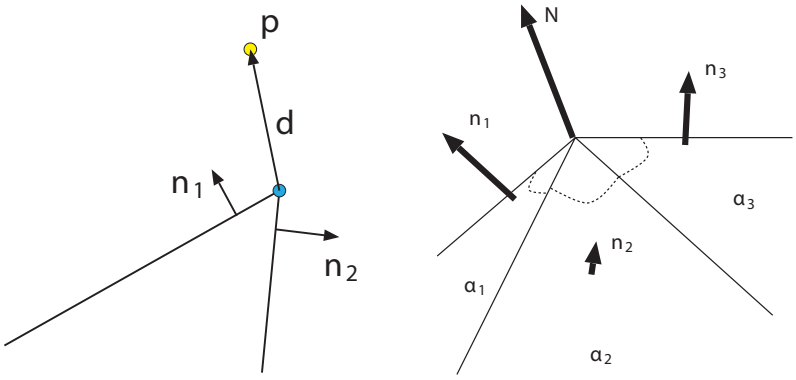
When it comes to calculating the sign of the distance for triangle meshes, there are basically three approaches:

Scan Conversion When a ray is cast from the voxel's location, the voxel is inside if it crosses the mesh an uneven number of times. This can be expanded to cast several rays in different directions for meshes where holes are present.

Using Characteristics The notion of *characteristics* was introduced by December and Mauch [14]. Each feature in the triangle mesh is converted to a polyhedron which includes the closest point to that feature. These characteristics are essentially truncated Voronoi diagrams. Through these characteristics it can be easily determined whether the closest point is inside or outside the mesh. However, they have to overlap slightly to avoid missing grid points inside the volume. This can lead to wrong voxel classifications in areas where the numerical distance is the same, but the sign is not.

Using Normals The simplest approach for any surface would be to use the surface normal in order to determine if a point is inside or outside the surface. The sign can be found by calculating the dot product of the normal n and a direction vector d from the closest point on the surface to the point p.

The latter method would be an obvious choice for any surface, but poses a problem with triangles meshes. There the normal is not defined for vertices and edges, only for the triangle itself. Using the triangle normal for an edge or vertex regardless does not work, because in many cases the sign will be different, depending on which triangle normal that is adjacent to a vertex



(a) p is closest to a vertex. (b) The angle weighted normal N.

Figure 2.9: Distance classification for triangle meshes. (a) The mesh feature that is closest to the point p is a vertex. the dot products $d \cdot n_1$ and $d \cdot n_2$ do not produce the same sign. (b) The angle weighted normal $N = \frac{\sum_i n_i \alpha_i}{||\sum_i n_i \alpha_i||}$. Images from Jones et al. [26].

is chosen (see Figure 2.9(a)). Baerentzen and Aanaes [7] solve this problem by calculating normals for edges and vertices as well. They use angle weighted normals for vertices, which are calculated by the sum of the normals of the incident faces, weighted by the angle between two edges that are incident to the vertex (Figure 2.9(b)).

When computing the distance field only for voxels in close proximity or adjacent to the original surface, the distance for the remaining voxels can be generated through so called *Distance Transforms*. Some of the popular methods are the *Chamfer Distance Transform*, *Vector Distance Transform* and *Eikonal Solvers*. These algorithms largely follow the same principle where they compute an estimated distance according to the distances stored in neighbouring voxels in either a *sweeping scheme* or a *wavefront scheme*. A survey on these techniques can also be found in Jones et al. [26].

These discrete distance fields are usually stored in uniform grids due to the simplicity of representation. Such 3D distance fields can also be used in direct volume rendering for standard first hit ray–casting, where the iso value to be searched for is the zero distance value for the original object. The downside of this representation is the need for a high resolution grid, when very small details of an object should be captured as well. In order to keep the memory impact low for such cases, hierarchical or run–length encoded grids could be used. A more flexible solution is offered by Frisken et al. [16] with *Adaptively Sampled Distance Fields*.

2.4 High Quality Texture Filtering

An important task in computer science is reconstruction of images and volumes from sampled data. While in many cases only nearest neighbour or linear filtering is sufficient, higher-order filtering is often desired in computer graphics applications. In literature the *sinc function* is given as the ideal reconstruction filter. However, it is impractical to use, since it expands infinitely in

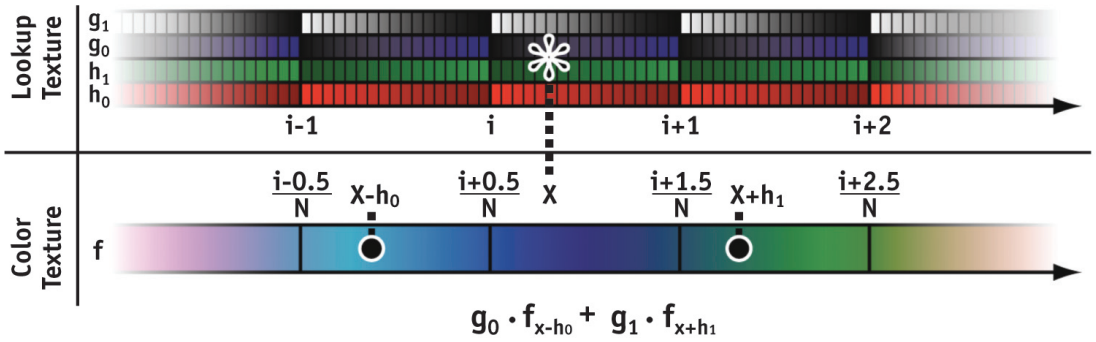


Figure 2.10: Cubic Filtering of a one–dimensional Texture. To reconstruct a colour texture of size N, a linear transformation for the reconstruction position x is performed (denoted by $*$ in the figure). This gives the texture coordinates for reading the offsets $h_i(x)$ and weights $g_i(x)$ from a lookup texture. Two linear texture fetches of the input colour texture are carried out at the offset positions (denoted by $\bullet$ in the figure). Finally, the output colour is computed by a linear combination of the fetched colours using the weights $g_i(x)$ [54].

the spatial domain. Therefore it is necessary to use functions that approximate the ideal filter. One type of higher–order filters are *Symmetric Cubic Filters*. Mitchell and Netravali [39] derived a family of such filters dependent on two variables called *BC–cubic splines*. Some values for the variables B and C correspond to known filters like the *cubic B-spline* or the *Catmull-Rom* spline. Another possibility of approximating the sinc function is to simply truncate it. Many of these functions can be found in literature nowadays and are usually called *windows*. Some of these are evaluated by Theußl et al. [56].

Reconstruction filters of higher order are computationally expensive and thus their use in interactive environments is limited when calculated purely on the CPU. When dealing with two- or three–dimensional textures in computer graphics, data reconstruction with nearest neighbour or linear filtering can be done fast, since it is natively supported by computer graphics hardware. Programmable Shaders and increased processing power of modern graphics cards make it possible to implement even higher order filters on the GPU. Generally, GPU shading programs are slower at math operations, but faster in texturing.

Filtering operations are usually done by convolving the original data with a kernel. These convolution kernels are usually stored as look–up textures, rather than analytically calculated on the fly on the GPU. Bjorke [8] gives a basic approach for implementing convolution kernels on the GPU as well as high-quality filtering through a cubic filter kernel. In the case for two-dimensional textures, this requires a total of 21 texture fetches.

An efficient single–pass approach for fast third–order texture filtering on the GPU has been presented by Sigg and Hadwiger [54]. By reformulating the general convolution sum as follows:

$$w_0(x)\cdot f_{i-1}+w_1(x)\cdot f_i+w_2(x)\cdot f_{i+1}+w_3(x)\cdot f_{i+2} = g_0(x)\cdot f_{x-h_0(x)}+g_1(x)\cdot f_{x+h_1(x)}, \quad (2.4)$$

it is possible to reduce the cubic filtering process to a series of linear texture fetches. $g_0(x)$

and $g_1(x)$ are the summed weights $w_0(x) + w_1(x)$ and $w_2(x) + w_3(x)$ respectively at sample position x, while $h_0(x)$ and $h_1(x)$ are offsets for the linear texture fetch between two input samples. These values are stored in a look–up texture and used in the shader program to compute the cubic filtered output colour (Figure 2.10).

This results in only 3 texture fetches for cubic filtering of one–dimensional textures, 6 for two dimensions, and 64 summands for tricubic filtering can be evaluated with only 9 linear texture fetches.

2.5 Subvoxel Quantification

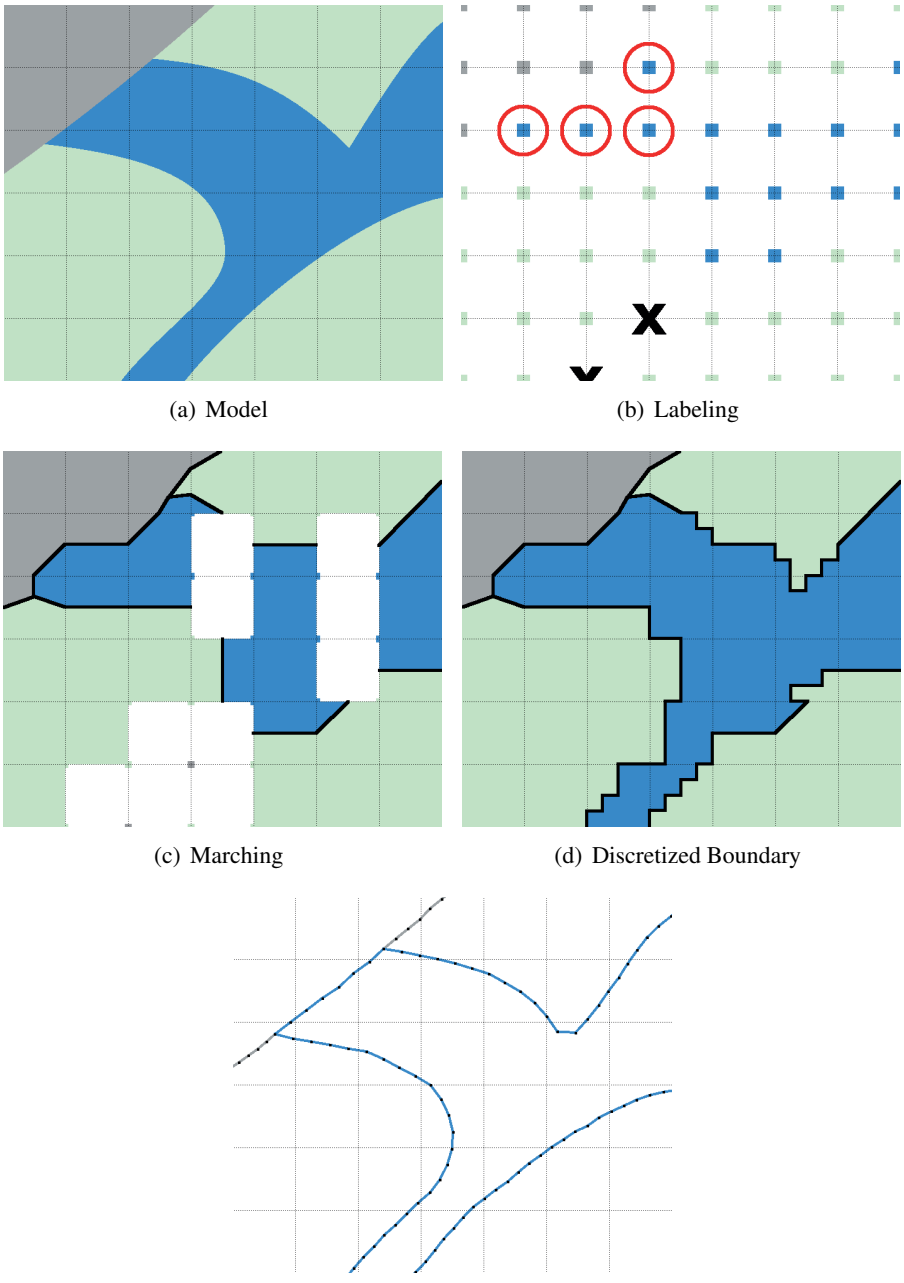
Many algorithms for segmenting CT or MRI data only work well in regions where a voxel contains only a single material and do not take the boundaries of these regions into account. A common method for acquiring such binary segmented data is thresholding for example. Surveys for such techniques are provided by Sezgin and Sankur [53] and Nacereddine et al. [41]. *Region growing* [1] or water shedding may be another source for binary segmentations of volume data.

Typically the resolution of data which is analyzed in various sciences is limited. Thus it is often desired to extract *fractional volumes* or accurately determine *surfaces* of segmented features within the data of a structural grid. Various techniques have been developed to improve the segmentation process. In many cases these techniques are specially made for various types of data, e.g. results from „multi–fluid hydrodynamics simulations“ [3] or medical as well as industrial CT images and volumes.

2.5.1 Material Interface Reconstruction

The goal of the *Material Interface Reconstruction* (MIR) problem for instance is to create smooth, continuous boundary interfaces between regions of homogeneous materials. Anderson et al. [3] provide a method to create smooth, volume-accurate material interfaces for data with cell–based volume fractions. In these types of data sets for an n-material problem, each cell has an associated n-tuple describing the fraction of each material within that cell. Their approach consists of several stages. First, vertices within the mesh of the data are labeled according to certain rules (Figure 2.11(b)). Through these labeled vertices an initial interface is created using a rule–based marching method (Figure 2.11(c)). For cells that remain without an interface model (due to ambiguous cases in the previous stages) a so called *discretized boundary method* is applied to determine the interface within these cells (Figure 2.11(d)). This method is described in Anderson et al. [2]. Based on the material fractions, each volume cell is divided into several subcells. The subcells are then labeled by a material so that the number of subcells for each material is proportional to the volume of the material in the current cell. Each subcell is attributed with a local energy, depending on the number of its neighbouring subcells with a different label. This is a metric known as the Potts model energy [62]. The energy in the cell is optimized iteratively by randomly swapping the labeling of two different subcells. The change of the label will be accepted based on the change in energy.

In a final step, the interface is smoothed through a volume–adaptive reconstruction (Figure 2.11(e)). During the smoothing process, each vertex is subjected to several „forces“. The



(a) Model (b) Labeling

(c) Marching (d) Discretized Boundary

(e) Reconstruction

Figure 2.11: Method for Material Interface Reconstruction by Anderson et al. [3]. (a) Original data model. (b) Labeling of vertices within the data domain. (c) Rule–based marching to create initial interfaces. (d) Discretized boundary method to more accurately determine the interface on a subvoxel level for ambiguous cases (see Anderson et al. [2]). (e) Adjustment of the created interface to create a smooth, but still volume accurate mesh representation of the interface. Images from Anderson et al. [3].

smoothing force wants to drive the vertex to the average position of its surrounding vertices. Additionally several „outward“ forces exist, based on the face normals incident to the vertex and the vertex normal itself. The outward force is weighted depending on the volume fractions of the cell, which will counteract the smoothing force, if the resulting volume is lower or higher than the given one in the cell.

2.5.2 Partial Volume Bayesian Algorithm

For CT and MRI data, Laidlaw [34] and Laidlaw et al. [35] devised an algorithm that uses probability estimates in order to classify voxels in these datasets. Rather than having a binary classification for each voxel, this method creates „material volume ratio densities“ and thus quantifies the volume of a material on a subvoxel level. An overview of the algorithm is sketched in Figure 2.12.

First the histogram for the entire dataset is created onto which parameterized material „probability density functions“ (PDF) are fitted, in order to get an estimate for parameters that are constant throughout the dataset. Then histograms for smaller regions of voxels are created, in order to find the combination of materials that fit the histogram most closely.

Bayesian probability theory is used in order to determine whether a given histogram was produced by a certain set of parameter values. This is used to find the material PDF parameters for the histogram of the entire dataset and later on to classify voxel regions.

From this Bayesian probability framework another classification method was devised that determines the distance from the center of the voxel to the boundary between two materials. In this variant, new histogram basis functions are used and the dataset parameters are estimated with a different algorithm. Figure 2.13 illustrates these new basis functions.

2.5.3 Mesh Smoothing

Instead of trying to extract the original material boundaries from a dataset directly, algorithms for smoothing binary segmented data are often applied, to improve the visual appearance of these regions. Most methods act on a polygonal mesh created from the segmentation.

Gibson [17] fits a mesh around the voxels belonging to a binary segmented region. This *Constrained Elastic Surface Net* consists of nodes connected to three or four neighbours. The surface net is then iteratively relaxed by moving the position of a node towards the average position of its connected neighbours, in order to reduce the overall energy of the surface net (Figure 2.14). This energy is defined by the squared distances between one node and its neighbours for instance, though other measures like the local curvature could be applied as well. This repositioning of such a surface net node is constrained by a bounding box around the original position of the node, in order to preserve features of the original shape and to prevent the surface net from contracting to a single point.

Based on these Constrained Elastic Surface Nets Holmstrom [23] created a modification specifically for marching cube meshes, though it could be applied to any mesh. Different constraint methods are evaluated that rely on the current structure of the mesh, with a constraint based on the Voronoi diagram of the original mesh being the best method. A similar work is done by Bade et al. [5], also comparing different constraint methods for meshes created by

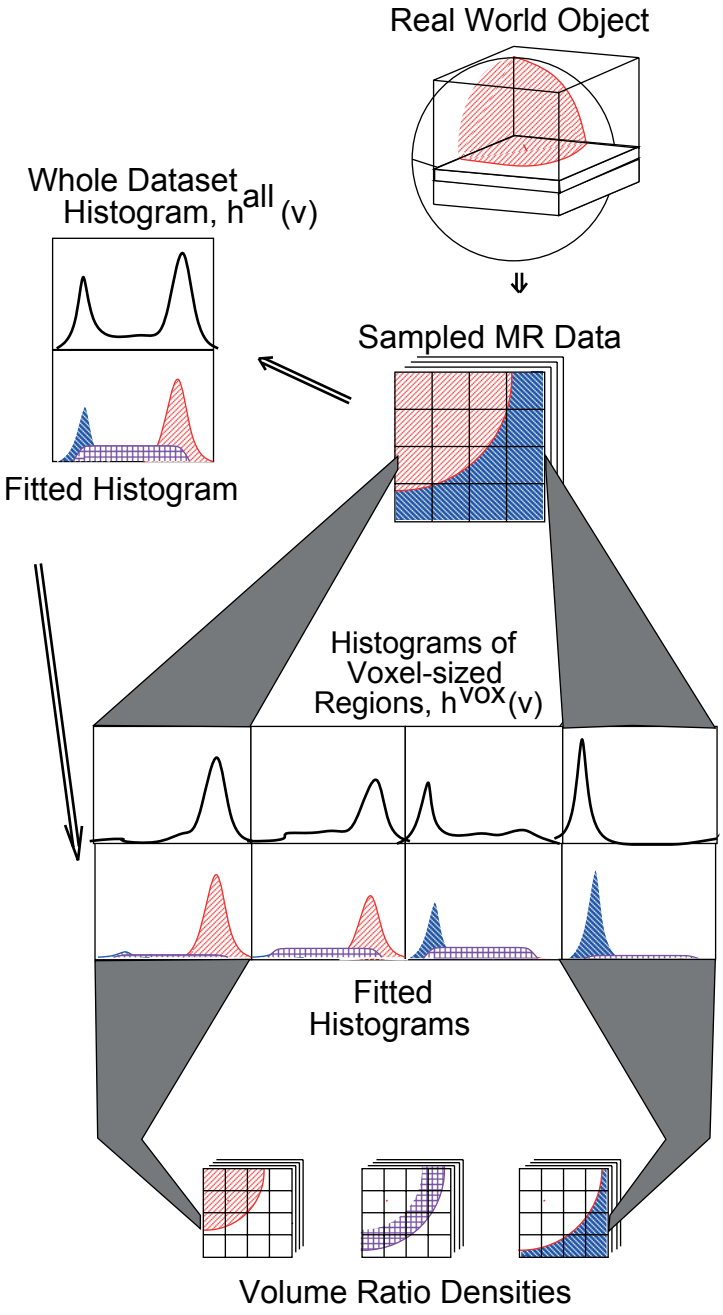


Figure 2.12: Steps in the classification process of the Partial Volume Bayesian method. Image from Laidlaw [34].

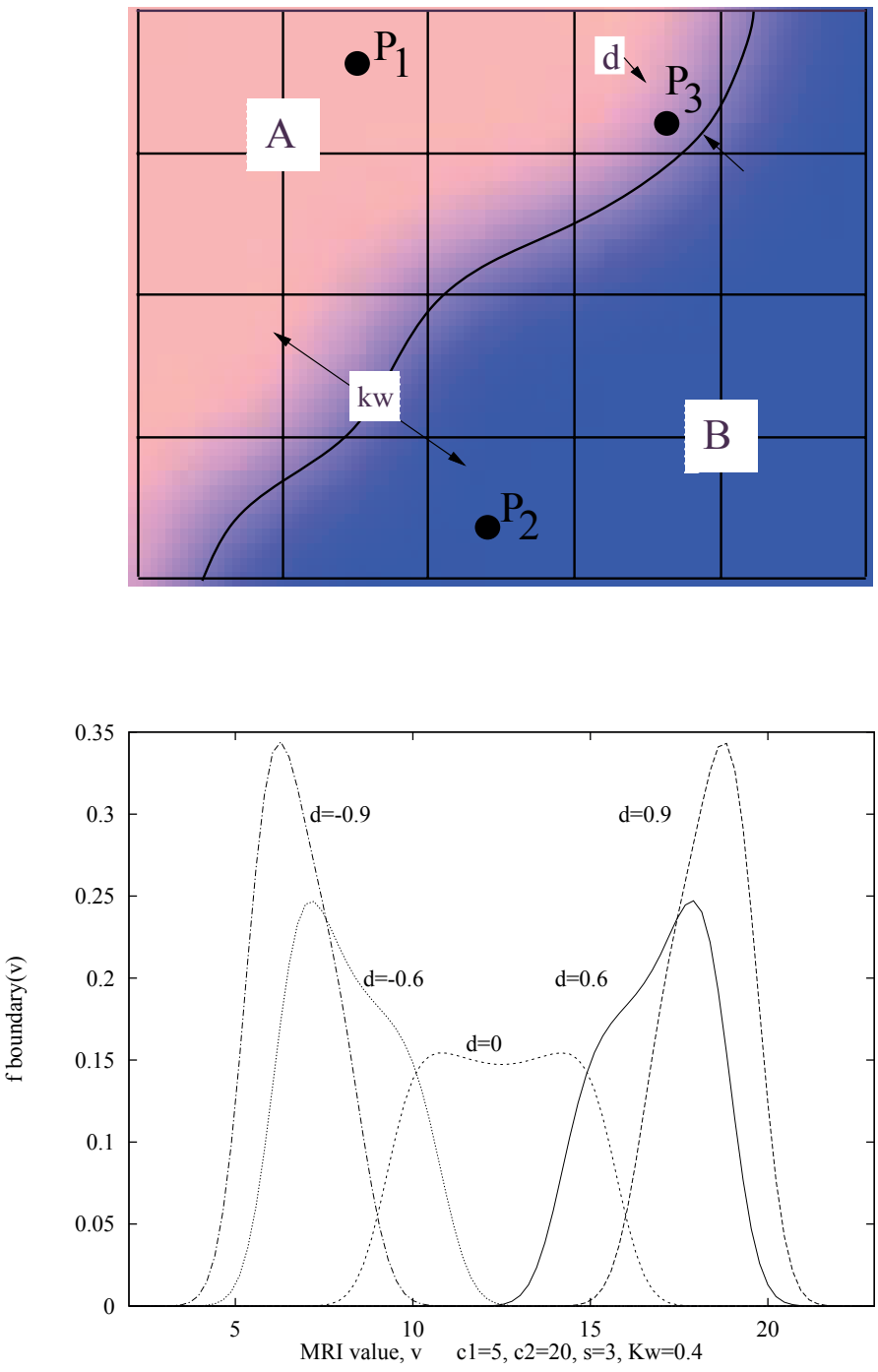


Figure 2.13: Top: Illustration of the parameter d (distance to boundary from voxel center) and k_w (width of material mixture region). Bottom: shapes of the histogram basis functions for different values of d. Image from Laidlaw [34].

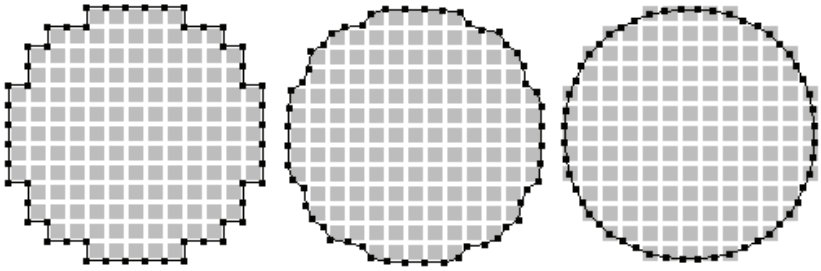
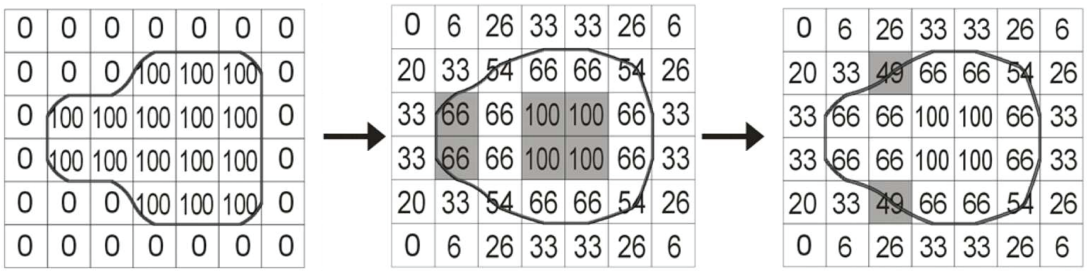


Figure 2.14: Constrained Elastic Surface Nets around binary segmented data. Pictures showing state of the Surface Net after 0, 1 and 10 relaxations. Images from Gibson [17].



(a) (b) (c)

Figure 2.15: (a) Voxelization of a binary segmented region as a distance field with $v_1 = 0$, $v_2 = 100$ and $v_{\text{iso}} = 50$. (b) ISO–surface after smoothing the distance field. Voxels of the reference object are marked in gray. (c) Original inside–outside classifications of the voxels marked in gray are restored. Images from Neubauer [42].

marching cubes. A survey over general mesh smoothing techniques is also provided by Bade et al. [4], with result comparisons from medical volume datasets.

Since the original, binary segmented data was extracted from medical volumes, Gibson [17] converted the smoothed surface net back into a format suitable for direct volume rendering. Using the techniques described in Section 2.3, a distance transform of the mesh was created. Transforming this mesh to a distance field again further smoothes the object, especially when cubic reconstruction is used during rendering.

Distance fields were also used for smoothing binary segmented data in a PhD thesis by Neubauer [42]. The region is directly voxelized into a distance field by setting all outside voxels to v_1 and all inside voxels to v_2, which defines the ISO–surface v_{iso} of the segmented region as $v_{\text{iso}} = \frac{v_1+v_2}{2}$ (Figure 2.15(a)). The resulting distance field is of course still heavily aliased, so a smoothing process is applied. Instead of directly applying a low–pass filter kernel to the distance field volume (e.g. a Gaussian), smoothing is achieved by creating a *reference object* first and assigning new distance values according to a voxel's distance to the surface of the reference object. The reference object is created through erosion of the region, however special measures are taken to prevent the erosion of important features of the region. Figure 2.15(b) shows the

reference object marked in gray and the resulting distance values after smoothing. The algorithm preserves the cone shaped feature of the region on the left. In a last step shown in Figure 2.15(c), the original classification of the voxels are restored. The voxels marked in gray are set to a value smaller than v_{iso}, because they were not part of the original object.

CHAPTER 3

Subvoxel Boundary Detection

Finding the original, subvoxel position of the boundary between two materials of a scanned object is the main part of this work. This chapter first describes some related work in Section 3.1, which provides the basis for the main algorithm that is described in Section 3.2. Some theories and assumptions regarding specific attributes of CT data are discussed, which will be exploited for finding the subvoxel boundary. Each step of the algorithm is described in more detail in the following sections. Section 3.3 explains the implementation of Constrained Elastic Surface Nets, a method for smoothing binary segmented data which was mentioned in Section 2.5.3. Actual quantification and measurement processes on these results are then described in Chapter 4.

3.1 Region Growing

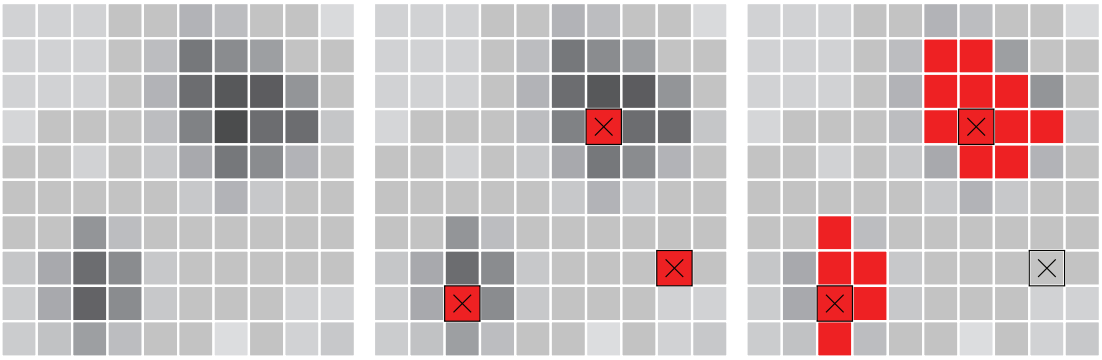
In the HVR framework, inclusions in CT datasets can be automatically segmented into *features* through *seeded region growing* [1]. Starting from a specific seed voxel, the idea is to add surrounding voxels to the current region incrementally, if certain criteria allow it. In the original publication, voxels are added to a region if the difference between the density of the new voxel and the average density of the whole region is within a given threshold

$$\epsilon : |v - v_r| < \epsilon, \tag{3.1}$$

with v being the voxel's density and v_r the region's average density. The implementation used for this thesis includes several alterations and additions to this concept which are briefly described in the following part.

3.1.1 Region Growing Criteria

In addition to the original method, it is possible to include a boundary growing pass after the region has finished growing. In this pass, voxels adjacent to the region are included according to either a gradient magnitude criterion or by checking the voxel's *LH value* [58].



(a) Original data (b) Seed voxels (c) Resulting regions

Figure 3.1: Basic steps in seeded region growing. Every voxel in the original data set (a) is considered a seed voxel initially. The result of three exemplary seed voxels (b) is shown in Figure (c). The feature resulting from the seed voxel in the lower right is discarded, because it became too big. It grew the region of the surrounding, dense material.

An alternative region growing criterion is described by Huang et al. [24], based on the standard deviations of density and gradient magnitude, with a scale factor k as an input parameter:

$$\begin{aligned} f_{ca} &= \frac{|v - v_s|}{k\sigma_v}, \\ f_{cb} &= \frac{|g - g_s|}{k\sigma_g}, \\ f_{cc}(p) &= p f_{ca} + (1-p) f_{cb}, \end{aligned} \tag{3.2}$$

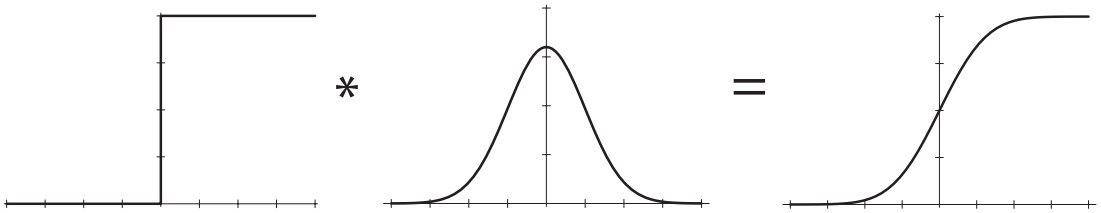
with v being a voxel's density value, v_s the density of the seed voxel, g the gradient magnitude, g_s the gradient magnitude of the seed, σ_v and σ_g the corresponding standard deviations of the seed neighborhood and p being a constant value.

3.1.2 Seed Selection

Instead of requiring the user to select one or several specific seed voxels for regions in order to manually segment the data, the implementation in the HVR framework considers *every* voxel in the dataset as a seed voxel initially. Region growing is applied iteratively for each voxel, however a seed can only be a voxel which does not belong to a feature yet or was not part of a discarded feature. Features are discarded by either being too small or too large (see Figure 3.1). These thresholds are additional input parameters which are set by the user.

3.1.3 Multi-Pass Region Growing

Equations 3.1 and 3.2 rely on user specified input parameters ϵ and k respectively. In order to easily explore the results for different values of these input parameters, multi-pass region



(a) Original Boundary (b) Gaussian (c) Measured Boundary

Figure 3.2: Measured boundaries (c) are step functions (a) blurred by a Gaussian (b). Images from Kindlmann and Durkin [28].

growing is applied. Here a number of *time steps* and a minimum and maximum *variance* value is defined. For the first pass, region growing is executed with the corresponding variance as the input parameter for either ϵ or k, depending on the method used. In the subsequent passes, existing regions are grown further from their boundary voxels with the new variance value. New seeds are created at yet unassigned voxels. During this process, features can also merge according to the region growing criteria.

The result of this region growing method can be explored interactively in GPU–based direct volume rendering. Through an intricate system that uses a feature volume and feature look–up tables in the fragment shader of the main volume rendering pass, this result can be interactively explored. Features can be selected by their size or other statistical properties and at specific time–steps through a 2.5D transfer function. This system is explained in more detail in Hadwiger et al. [20].

The remainder of this thesis is only concerned with the resulting regions at a specific time–step. These automatically determined regions provide the basis for the main part of this work, which is described in Section 3.2.

3.2 Gradient Integration

The goal of this part of the thesis is to devise a method for extracting the possible *subvoxel boundary* of materials, or in this case specific regions of the original volume data. The process of region growing divides the volume into binary segmentations, where each voxel either fully belongs to a region or not. However due to the inherently limited resolution during the acquisition process of the data, individual voxels may contain a mixture of different materials due to the Partial Volume Effect (see also Section 2.1.3). This effect cannot be accounted for with region growing alone. Therefore it is desirable to formulate the properties of the PVE in the final data and devise a method to exploit these properties in order to closely reconstruct the original boundary on a subvoxel level.

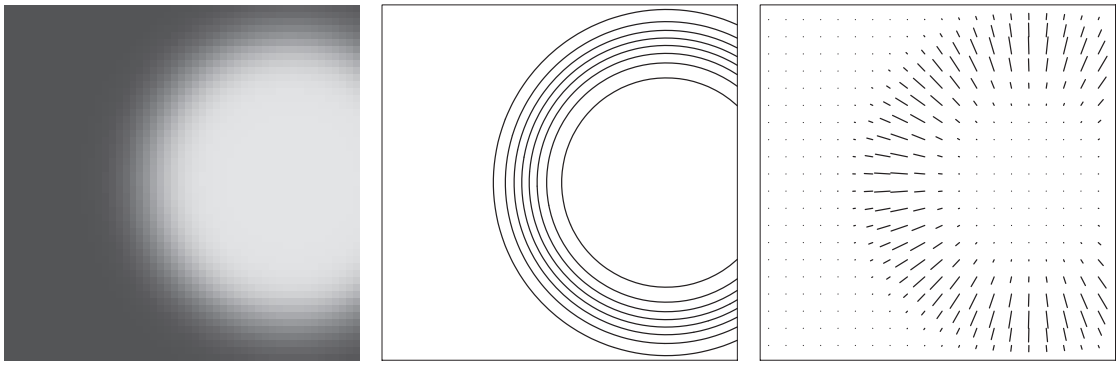


Figure 3.3: A dataset (left) and its isocontours (middle) and gradient field (right). Images from Kindlmann and Durkin [28].

3.2.1 Boundary Model Assumptions

For the purpose of defining an ideal boundary model, the same assumptions as in Kindlmann and Durkin [28] are used. The first assumption is that the measurement process for acquiring CT or MRI data is band–limited with a Gaussian frequency response. This causes a measured boundary to be blurred by a Gaussian function. Thus the boundary between two materials will be spread over a range of data values after the sampling process (Figure 3.2). Chiverton and Chiverton [11] describe this as a concept known as the Point Spread Function (PFS) of the acquisition system. This spreading typically affects multiple voxels along a boundary and is determined by the action of the PSF. Furthermore it is assumed that this blurring occurs uniformly in all directions.

Based on the mathematical property that the gradient vector at some arbitrary position inside the data is always perpendicular to the ISO–surface through that position, the gradient vector can be used to find the direction perpendicular through the object boundary (see Figure 3.3). In order to find this boundary, the directional derivatives of the scalar field f along the gradient vector $\mathbf{v}$ are evaluated. The derivatives are denoted as f' and f'' for the first and second derivative. Since these are actually directional derivatives along the gradient direction, a more accurate description would be $\mathbf{D}_{\widehat{\nabla f}}$ and $\mathbf{D}^2_{\widehat{\nabla f}}$ for the first and second directional derivative respectively, with $\widehat{\nabla f}$ being the normalized gradient direction.

Figure 3.4 shows the measurements of density values $f(x)$ and first and second derivatives $f'(x)$ and $f''(x)$ along a spatial extent defined by the gradient vector at each position x. The step function of the original boundary is lost during the acquisition, however an exact location for the said boundary can be found by taking the maximum of f' or the zero–crossing of f'', a concept that is also widely used in computer vision for edge detection for example. Thus we need to formalize, how the second derivative of the volume data can be obtained, along a specific direction.

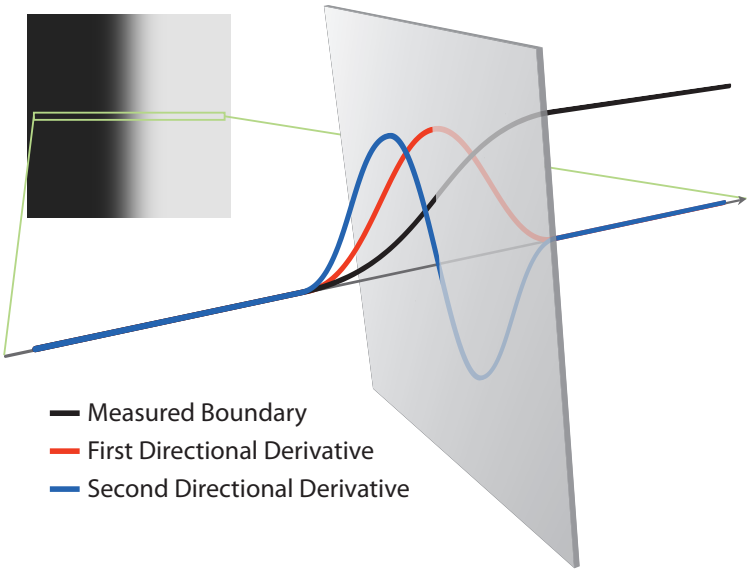


Figure 3.4: Values for the measured boundary f, the first directional derivative f' and the second directional derivative f'' across a boundary.

Calculating the first derivative

The first derivative of the data along its gradient direction can be formalized as follows:

$$\mathbf{D}_{\widehat{\nabla f}} f = \nabla f \cdot \widehat{\nabla f} = \nabla f \cdot \frac{\nabla f}{\|\nabla f\|} = \|\nabla f\|\,, \tag{3.3}$$

which is simply the gradient magnitude. The gradient of a scalar function $f(x_1, x_2, ..., x_n)$ from an Euclidean space E^n is the first derivative of f with respect to the n-dimensional vector components [31]. It is defined as

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, ..., \frac{\partial f}{\partial x_n}\right). \tag{3.4}$$

The simplest approach to calculate the gradient of a discrete function is through forward, backward or central differences. While this is the fastest method, it is also very inaccurate. Möller et al. [40] provide a comparison between various reconstruction methods while applying the central differences method. In computer vision, the *Sobel operator* is often used for calculating the gradient in images for edge detection. In $\mathbb{R}^3$ the operator uses three $3 \times 3 \times 3$ kernels which are convolved with the original data to calculate the approximate derivative. One operator for changes in values in the direction of each axis. The 3D Sobel kernel in z–direction

for instance is defined as

$$h'_z(:,:,-1) = \begin{bmatrix} +1 & +2 & +1 \\ +2 & +4 & +2 \\ +1 & +2 & +1 \end{bmatrix} h'_z(:,:,0) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} h'_z(:,:,1) = \begin{bmatrix} -1 & -2 & -1 \\ -2 & -4 & -2 \\ -1 & -2 & -1 \end{bmatrix} \quad (3.5)$$

Depending on the application, sophisticated algorithms are required to replace the estimation of partial derivatives. Neumann et al. [43] propose a linear regression model. Chowdhury et al. [12] provide an evaluation of 3D gradient filters for volume rendering.

Calculating the second derivative

Kindlmann and Durkin [28] propose three different methods for calculating the second derivative. Based on the first derivative along the gradient direction, the second derivative can be created by applying Equation 3.3 twice:

$$\begin{aligned} \mathbf{D}^2_{\widehat{\nabla f}} f &= \mathbf{D}_{\widehat{\nabla f}}(\|\nabla f\|) = \nabla(\|\nabla f\|) \cdot \widehat{\nabla f} \\ &= \frac{1}{\|\nabla f\|} \nabla(\|\nabla f\|) \cdot \nabla f. \end{aligned} \quad (3.6)$$

This is basically the normalized product of the gradient with the gradient of the gradient magnitude. The involved calculations are relatively simple and the computation time can be improved, if the gradient magnitude has already been precomputed for the whole volume.

The most accurate method can be achieved by using the Taylor expansion of f along the gradient:

$$\mathbf{D}^2_{\widehat{\nabla f}} f = \frac{1}{\|\nabla f\|^2} (\nabla f)^{\mathrm{T}} \mathbf{H} f \nabla f, \quad (3.7)$$

with $\mathbf{H}f$ being the Hessian of f, a matrix consisting of the second order partial derivatives:

$$\mathbf{H}f = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x^2 \partial y^2} & \frac{\partial^2 f}{\partial x^2 \partial z^2} \\ \frac{\partial^2 f}{\partial y^2 \partial x^2} & \frac{\partial^2 f}{\partial y^2} & \frac{\partial^2 f}{\partial y^2 \partial z^2} \\ \frac{\partial^2 f}{\partial z^2 \partial x^2} & \frac{\partial^2 f}{\partial z^2 \partial y^2} & \frac{\partial^2 f}{\partial z^2} \end{bmatrix}. \quad (3.8)$$

The complete calculation of the Hessian matrix would have a severe impact on the computation time. Another variant with the least computation cost would be the Laplacian operator to approximate $\mathbf{D}^2_{\widehat{\nabla f}} f$:

$$\mathbf{D}^2_{\widehat{\nabla f}} f \approx \nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2}. \quad (3.9)$$

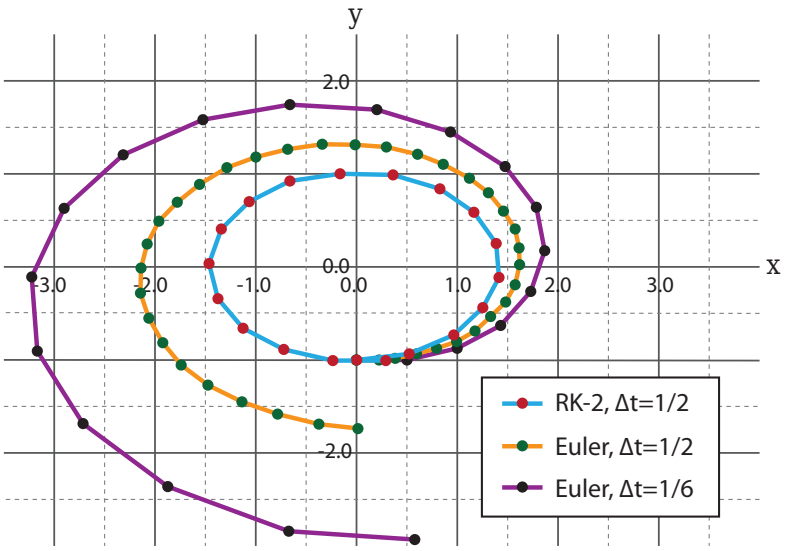


Figure 3.5: Comparison between Euler's method and second order Runge–Kutta (RK-2) integration over a vector field $\mathbf{v}$ defined by $\mathbf{v}_x = \frac{\mathrm{d}x}{\mathrm{d}t} = -y, \mathbf{v}_y = \frac{\mathrm{d}y}{\mathrm{d}t} = \frac{x}{2}$. Euler's method diverges from the correct solution by far, even with a much smaller step size Δt.

The Laplacian operator only requires a single convolution in order to approximate the second derivative. However, the response to single kernels with a small width of 3 or 5 voxels is very sensitive to noise. Thus the first variant, Equation 3.6 provides a good trade–off between computation time and accuracy.

3.2.2 Integration along the Gradient

Instead of evaluating the zero–crossing of the second derivative along the unit vector of the gradient $\widehat{\nabla f}$ from an arbitrary point in the volume, the derivative will be evaluated along a path defined by the gradient vector field. This path is defined by the line integral

$$\mathbf{s}(t) = \mathbf{s}_0 + \int_{0 \leq u \leq t} \mathbf{v}(\mathbf{s}(u))\mathrm{d}u \tag{3.10}$$

where $\mathbf{v}$ is the gradient vector field of f and $\mathbf{s}_0$ the starting point. This means

$$\mathbf{D}_s^2 f = 0 \tag{3.11}$$

would need to be solved. Of course such an integral cannot be solved analytically. Thus numerical methods for solving the integral $\mathbf{s}(t)$ are employed. The simplest one is Euler's method [10]:

$$\mathbf{s}_{i+1} = \mathbf{s}_i + \Delta t \cdot \mathbf{v}(\mathbf{s}_i), \tag{3.12}$$

with the complete path being defined as

$$\mathbf{s}_n = \mathbf{s}_0 + \sum_{0 \leq u \leq n} \Delta t \cdot \mathbf{v}(\mathbf{s}_u). \tag{3.13}$$

Starting from position s_0, the vector $\mathbf{v}$ at the current position s_u is evaluated and the position is advanced using this vector multiplied by the step size Δt. The idea behind this is, that the solution to the integral is locally nearly linear. Thus with a small enough step size Δt, the integral can be approximated iteratively by going from the current point $\mathbf{s}_i$ forward or backward in the direction of $\mathbf{v}(\mathbf{s}_i)$. However, even with a small step size the error will increase significantly with each step. A better solution is the second order Runge–Kutta method:

$$\mathbf{s}_{i+1} = \mathbf{s}_i + \Delta t \cdot \mathbf{v}(\mathbf{s}_i + \frac{\Delta t}{2} \cdot \mathbf{v}(\mathbf{s}_i)), \tag{3.14}$$

advancing at $\mathbf{s}_i$ with the vector from half an Euler–step forward. This leads to more accurate results, even with bigger step sizes (see Figure 3.5).

3.2.3 Distance Field Generation

The result of finding the subvoxel boundary using the techniques previously mentioned will be stored as a signed distance field (see also Section 2.3). Essentially a distance transform for the subvoxel boundary is created, i.e. where the second derivative of the data is zero. Positive values of the distance field represent the „outside“ of a feature (the surrounding material) while negative values represent the „insides“ of a feature. The zero–distance ISO–surface of this distance field represents the subvoxel boundary of a feature and can be reconstructed with high precision, using high quality cubic filtering for instance.

3.2.4 Algorithm

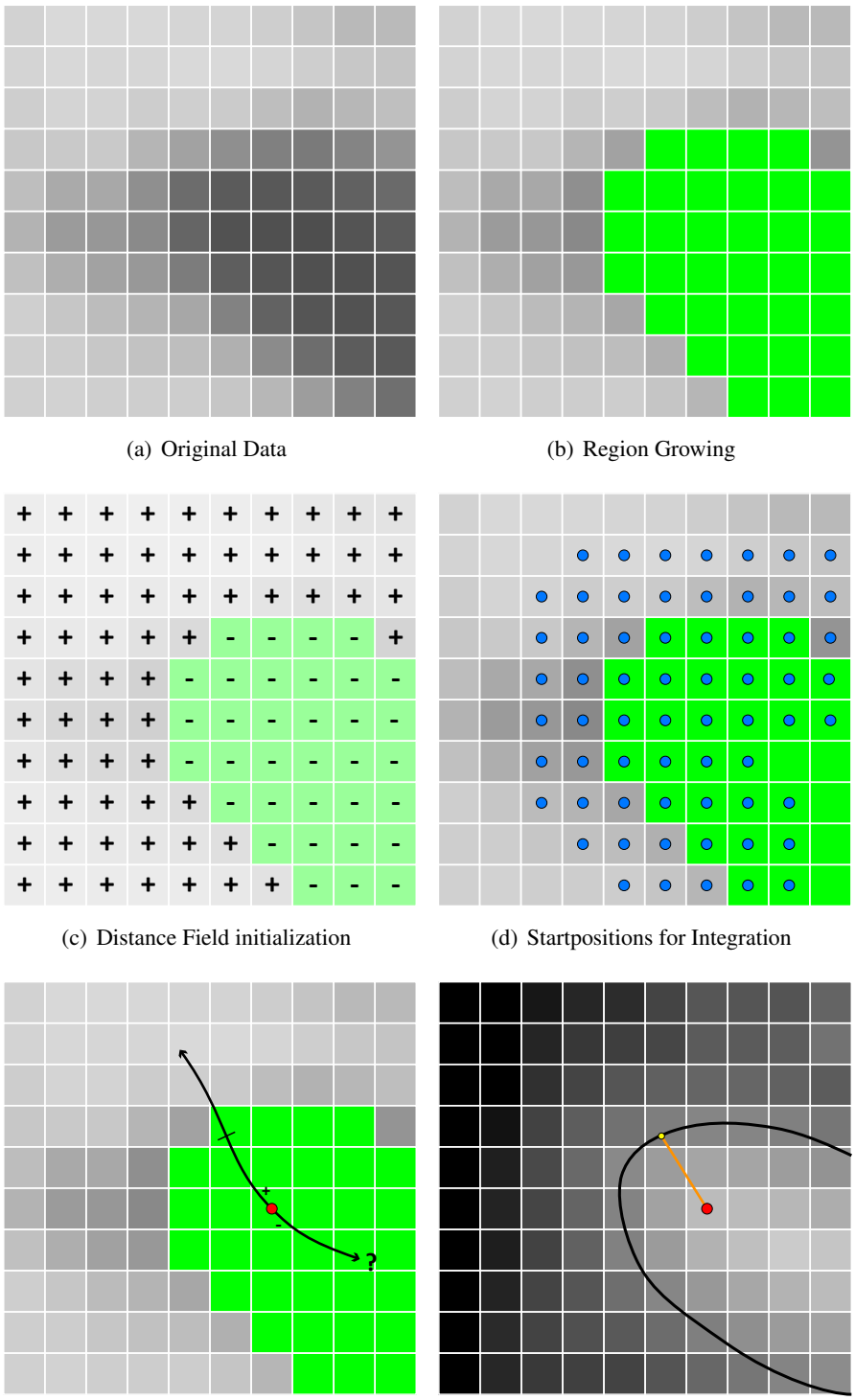
The main algorithm for finding the subvoxel boundary is outlined in Algorithm 3.1. Figure 3.6 provides an illustration of the process as well, with images produced from a real dataset. In the following section, each component of the algorithm is described in more detail.

Region Growing

The presented method relies on the automatically segmented data provided by the region growing technique described in Section 3.1 (Figure 3.6(b)). The algorithm can then be applied on all features of the volume or on specific features, if the user wants to save time or is generally only interested in those. In the framework, a single feature can be selected for this purpose, or multiple features can be selected through a 2D transfer function for instance.

Distance Field initialization

The first step is to initialize a distance field for the whole volume, with the same size as the underlying data. Since positive values in the distance field represent the surrounding area of



(a) Original Data

(b) Region Growing

(c) Distance Field initialization

(d) Startpositions for Integration

(e) Resulting point along gradient path

(f) Resulting Distance Field

Figure 3.6: Steps during the main algorithm for finding the subvoxel boundaries of features.

input : Selected features from region growing
output: Distance field with feature id information

```
1  initialize distance field with positive infinity
2  forall the selected features do
3      set distance to negative infinity on feature voxel positions
4      calculate integration starting positions
5      forall the starting positions do
6          run gradient integration
7          analyze result and update distance field
8      end
9  end
10 prepare distance field for GPU
```

Algorithm 3.1: Main algorithm that creates the distance field representing the subvoxel boundary of the original material borders from the real object of the dataset.

the features, every voxel of the distance field is initialized with a positive infinity value (some arbitrary high number).

Then for each feature, every voxel belonging to that feature will be initialized with a negative infinity value (Figure 3.6(c)). This is important because the algorithm only calculates the distances to the subvoxel boundary near the supposed boundary of the feature. Nothing will be evaluated for voxels that are far from the boundary, so this is done to get a consistent inside/outside classification for the whole feature. Of course this happens under the assumption, that the provided segmented region does not contain any erroneous holes inside the feature.

Calculate starting positions

The algorithm creates starting positions along the boundary of the binary segmented feature (Figure 3.6(d)). Starting positions are created at positions which are two voxels within the boundary and two voxels beyond the boundary. This is done through a dilation of the boundary voxel positions of the feature and then a second dilation on these new positions, this time only considering positions outside the feature (Figure 3.7).

Gradient integration

For each of the starting positions from the previous section, the numerical integration along the gradient vector field of the dataset is initiated in the negative and positive direction of the gradient vector (Figure 3.6(e)). Algorithm 3.2 shows this gradient integration process in more detail. The basis for one integration process is the starting position P_0, the direction dir and the initial step size Δt_0. The state of the integration process consists of a current position P_{curr}, the gradient at the current position G_{curr}, the gradient of the gradient G^2_{curr}, the first and second

(a) (b) (c)

Figure 3.7: Creating starting positions for the gradient integration. (a) Boundary voxel positions (white) are dilated, resulting in (b) new starting positions (blue). (c) Starting positions outside of the feature (white) are dilated again.

derivative f' and f'', the current step size Δt and a result state s_r, which describes whether a point at the zero–crossing of the second derivative was actually found. Most of these variables are initialized at first depending on the starting position (Line 1–8).

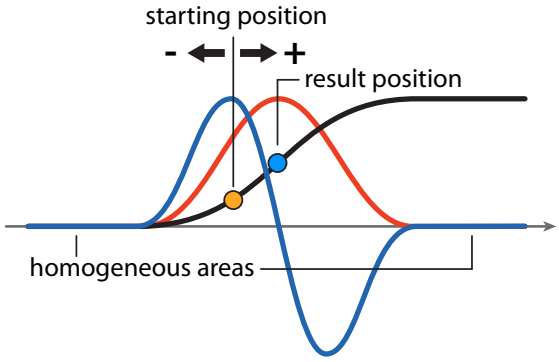
The main loop of the integration runs until either one of the breaking points within the algorithm or a maximum loop count is reached. At each iteration step, the first (Line 10) and second (Line 11) derivative f' and f'' are calculated at the current position P_{curr}, according to Equations 3.3 and 3.6. The gradient computation $grad(x)$ and $grad^2(x)$ is done through a Sobel Kernel (Equation 3.5) with tri–cubic filtering.

What follows is a series of breaking points for the integration:

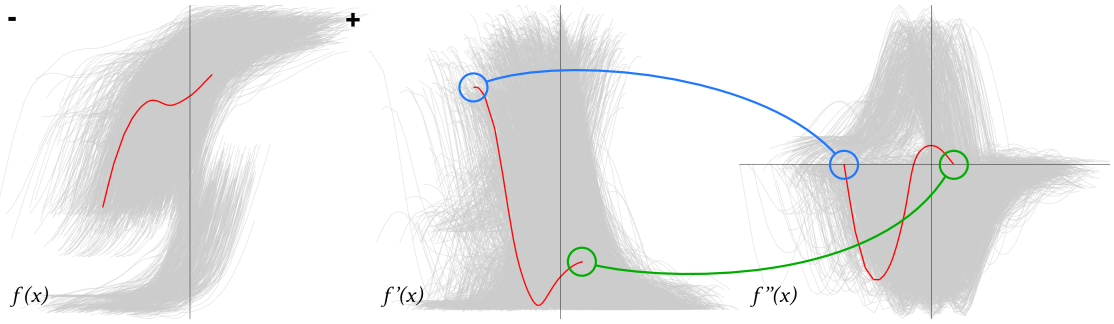
- The zero–crossing of the second derivative is reached (Line 12), i.e. when f'' is very close to zero. But a valid point will only be found, if the first derivative (the gradient magnitude) is above some minimum threshold, since zero–crossings can occur outside of actual boundaries as well.
- The integration reached a homogeneous area, if the first derivative is below a certain homogeneity threshold (Line 15).
- The integration reached a point that is farther than a certain maximum distance from the initial starting position (Line 17).

In most cases the integration will never hit a position where f'' comes close enough to zero. Usually it will cross zero at some point, so in order to exactly find the position of the zero–crossing, the step size will be adaptively reduced in that case and the integration continues from the previous point (Line 19). For this purpose, the previous position P_{prev} and gradient G_{prev} will be stored as well (Line 25–27).

The last section of the algorithm (Line 28–31) actually executes the numerical integration along the gradient vector field, according to the second order Runge–Kutta method described in



(a) Ideal result



(b) Ambiguous result

Figure 3.8: (a) Ideally only one direction leads to a result. (b) Ambiguous case in a real dataset (red plot). The result with the higher gradient magnitude $f'(x)$ will be the valid result position (negative direction in this case).

Equation 3.14, and evaluates the new gradient G_{curr} and gradient of the gradient G^2_{curr} at the new position.

Since the gradient integration process for each starting position (i.e. each voxel near the boundary) does not affect the computations of other voxels, current multi–core architectures can be easily exploited, in order to decrease the computation time considerably. Basically, the forward and backward gradient integration and the result analysis afterwards for one voxel run in independent threads for each starting position. Shared data between all threads include the volume of the original data set, which is just a read–only access, and the distance field volume, for which the write access is thread–safe, since one thread only writes to one particular voxel position.

Result analysis and distance field update

When the integration along the negative and positive gradient direction is finished, i.e. when it reached either of the three conditions or the maximum iteration count, their results need to be analyzed. Ideally, only one direction leads to a result, while the other direction leads to the

input : $P_0, dir, \Delta t_0$: start position, direction, initial step size
output: P_r, G_r, s_r: result position, result gradient, result state

$P_{curr} \longleftarrow P_0$
$G_{curr} \longleftarrow grad(P_{curr})$
$G^2_{curr} \longleftarrow grad^2(P_{curr})$
$P_{prev} \longleftarrow P_{curr}$
$G_{prev} \longleftarrow G_{curr}$
$f''_{prev} \longleftarrow 0$
$\Delta t \longleftarrow \Delta t_0$
$s_r \longleftarrow \bot$
while *maximum iteration count is not reached* **do**
    $f' \longleftarrow ||G_{curr}||$
    $f'' \longleftarrow \frac{G^2_{curr} \cdot G_{curr}}{f'}$
    if $abs(f'') <$ *some epsilon* $\wedge f' >$ *some minimum threshold* **then**
        $s_r \longleftarrow \top$
        break
    if $f' <$ *some homogeneity threshold* **then**
        break
    if $||P_0 - P_{curr}|| \geq$ *some maximum distance* **then**
        break
    if $crossedZero(f'', f''_{prev}) \wedge f' >$ *some minimum threshold* **then**
        $\Delta t \longleftarrow \frac{\Delta t}{2}$
        $P_{curr} \longleftarrow P_{prev}$
        $G_{curr} \longleftarrow G_{prev}$
    else
        $\Delta t \longleftarrow \Delta t_0$
    $P_{prev} \longleftarrow P_{curr}$
    $G_{prev} \longleftarrow G_{curr}$
    $f''_{prev} \longleftarrow f''$
    $P_{preview} \longleftarrow P_{curr} + G_{curr} \cdot dir \cdot \frac{\Delta t}{2}$
    $P_{curr} \longleftarrow P_{curr} + grad(P_{preview}) \cdot dir \cdot \Delta t$
    $G_{curr} \longleftarrow grad(P_{curr})$
    $G^2_{curr} \longleftarrow grad^2(P_{curr})$
$P_r \longleftarrow P_{curr}$
$G_r \longleftarrow G_{curr}$

Algorithm 3.2: Numerical integration loop for finding the subvoxel boundary, i.e. the zero–crossing of the second derivative.

homogeneous area of either the feature itself or the surrounding material (Figure 3.8(a)). In real datasets however, ambiguous cases can occur where the algorithm finds a result in both directions. In such a case the result with the higher gradient magnitude is chosen, since the boundary for which we search for with the zero–crossing of the second derivative is also where the first derivative, the gradient magnitude, reaches a maximum.

Figure 3.8(b) shows plots for $f(x)$, $f'(x)$ and $f''(x)$ along the gradient path for each starting position of a single feature. The data is taken from the gradient integration result of feature #5 of the cast housing dataset (Section 5.3). The horizontal axis reflects the travel distance during the integration in the positive (right side) and negative (left side) gradient direction. The highlighted plot in red shows an ambiguous result. In this case the result in the negative direction is chosen as the correct result, since its gradient magnitude is much higher (marked in blue). This trace of the data values and its derivatives along the gradient path also shows a case where there are actually several zero–crossings of the second derivative. There are two crossings in the negative direction, however the first crossing was not identified as a valid result position during the gradient integration, since the gradient magnitude did not exceed the defined minimum threshold at this position.

The distance from the starting position to the result position is then calculated and stored within the voxel corresponding to the starting position (Figure 3.6(f)). However, storing only the distance in the distance field is not sufficient. As mentioned before, a signed distance field is needed. Thus the voxel for which we want to update the distance needs to be classified as either an inside or outside voxel, which in turn defines the sign for the distance. In this procedure, the classification simply depends on the direction in which the result was found and the material density of the feature. If the feature is surrounded by a more dense material, the gradient vector field points from the inside to the outside of the feature. Therefore if the result was found in the *positive* direction of the gradient, the voxel will be classified as *inside* and a negative sign is applied to the distance. If the result was found in the *negative* direction of the gradient, the voxel will be classified as *outside* and a positive sign is applied to the distance. The classification needs to be reversed, if the feature's density is higher than its surrounding density.

For voxels where neither direction leads to a definite result, nothing will be updated in the distance field. The voxel retains its initial value in this case.

Prepare distance field for GPU usage

Rendering as well as quantification of the resulting signed distance field will be done on the GPU. After the previous steps have been applied to all features, the distance field needs to be prepared to be usable as a three–dimensional texture. The 32-bit signed float values need to be mapped into a compatible format that is usable by the GPU. In the distance field volume the featured ID to which the distance points is stored as well. At least 16-bit are need for that, since there can be more than 256 features in a dataset. In a 32-bit volume dataset, this leaves us with another 16-bit for the distance values.

Using the actual absolute maximum distance d_{max} of the result (without $infinity$) in order to get the maximum precision into a 16-bit value, the distance values will be mapped from their

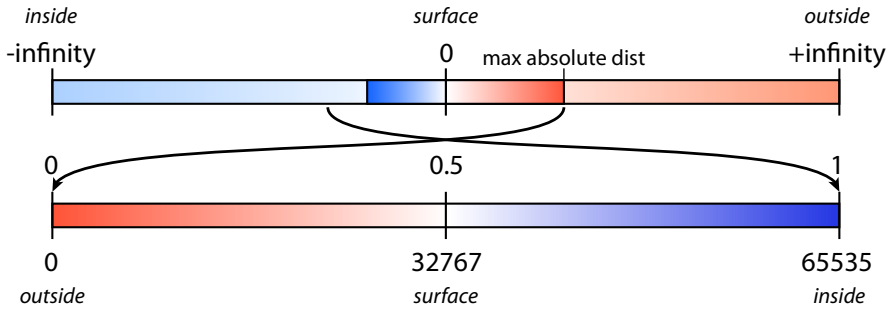


Figure 3.9: Mapping of the original distance field values into a format usable for a three dimensional texture on the GPU.

original values as follows:

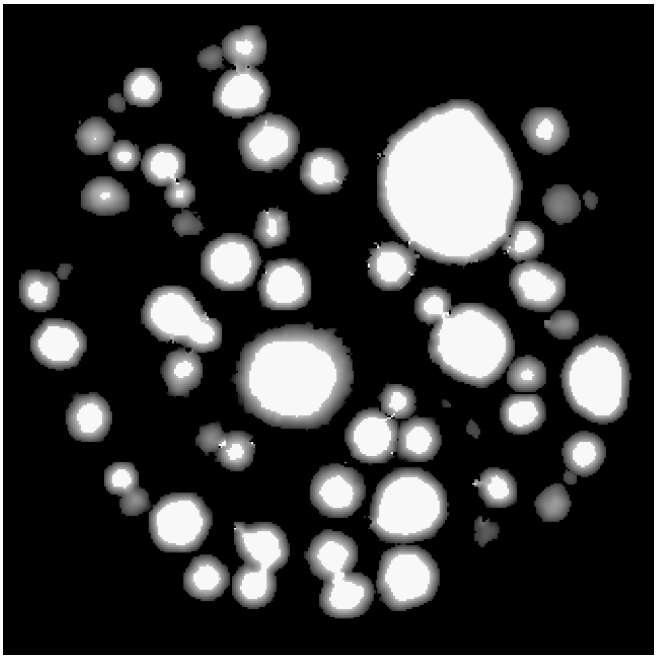
$$d_i = int\left(clamp\left(0.5 - \frac{d_i}{2 \cdot d_{max}}, 0, 1\right) \cdot 65536\right) \tag{3.15}$$

Figure 3.9 illustrates this process. All positive and negative infinity distances are clamped to 0 and 1 and all distance values in between are transformed into a $[0..1]$ range, relative to the determined maximum distance d_{max}. This value is then expanded to the 16-bit integer format used for the texture. Instead of casting the resulting value directly to an integer, a rounding strategy could be applied here as well. The reversal of the values around the zero–distance is done, because in traditional ISO–surface rendering through ray–casting, the ISO–surface is detected when a sample point is greater or equal to the ISO–value (during front–to–back ray–casting). The distance field volume can now be uploaded as a three–dimensional texture to the GPU and used in direct volume visualization or for the hardware–accelerated quantification described in Section 4.1. Figure 3.10(a) shows a slice through a resulting distance field volume created by the algorithm described in this section. White values represent the maximum negative distance from the boundary, while black values represent the maximum positive distance. Figure 3.10(b) shows the direct volume rendering of the zero–distance ISO–surface from the same dataset. The results from this dataset are discussed in more detail in Section 5.2.

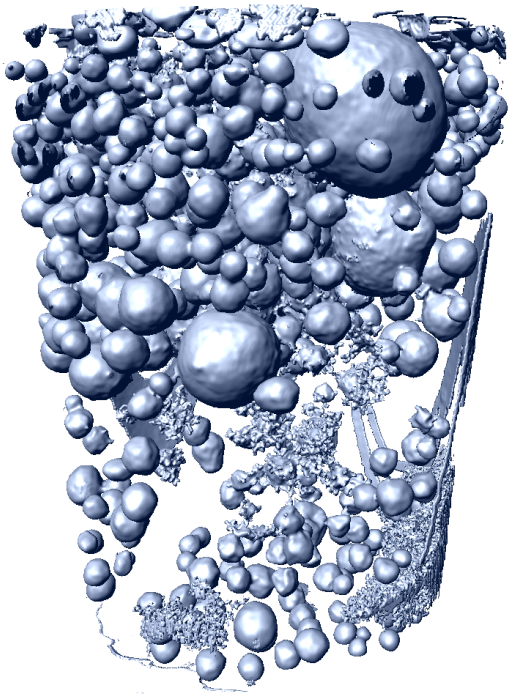
3.3 Constrained Elastic Surface Nets

To compare the results from the subvoxel boundary detection with results from smoothing methods, *Constrained Elastic Surface Nets* [17] were implemented as well. Since the original paper also uses a distance field representation of the smoothed surface, the volume quantification results (described in Section 4.1) can be compared directly.

As already mentioned in Section 2.5.3, the general principle is to create a net over the surface of the binary segmented volume data. This surface net consists of nodes and each node is connected to its direct neighbours on the surface. The position of these nodes will then be relaxed, in order to minify the energy of the surface net. There can be different definitions for

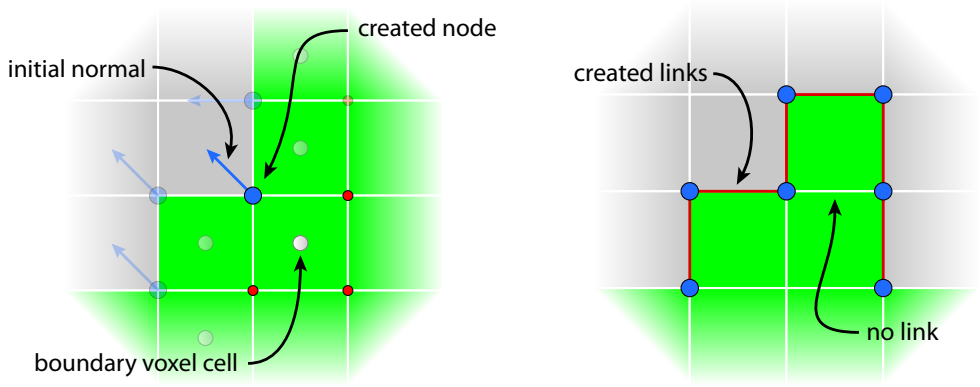


(a)



(b)

Figure 3.10: Resulting distance field for the RPTS dataset described in Section 5.2. (a) A slice through the distance field. (b) Direct volume rendering of the zero–distance ISO–surface.



(a) (b)

Figure 3.11: (a) Process for creating surface net nodes. For each boundary voxel of the region, all corners are checked, whether they are adjacent to at least one voxel that does not belong to the region. (b) Linking the surface net nodes.

what constitutes the total energy of the surface net. A common approach would be to define the energy as the sum of the squared distances from one node to its neighbors:

$$e = \sum_i \sum_j (n_i - n_j) \cdot (n_i - n_j) \tag{3.16}$$

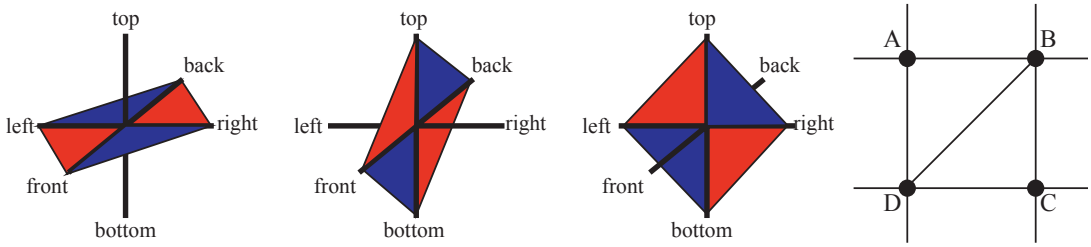
with $n_i \in S$ being a node of surface S and n_j being the linked neighbours of n_i. The basic steps during the creation of a surface net are:

- Initialize the surface net with nodes and correct links between nodes.
- Create a triangle structure based on the surface net.
- Relax the position of each node iteratively and update triangle normals.
- Create a distance field from the geometry.

The triangle structure is generated before the relaxation of the surface net, in order to track the correct orientation of triangle normals. Correct normals are important for the distance field generation later on. The initial normals for each node are determined by their relative position to the underlying binary segmented region. Figure 3.11(a) shows these initial node normals based on their surrounding region voxels in 2D.

3.3.1 Surface Net Generation

The first step is to create the surface net around a region of voxels. Nodes are created in between voxel cells, but only on the surface of the region. Based on a 26–neighbourhood, the boundary voxels of the region are determined. A voxel of the region is considered a boundary, if at least



(a) (b)

Figure 3.12: (a) 12 possible cases for triangle generation in the surface net (red and blue triangles). (b) If triangle A, B, D already exists, no triangle will be generated for A, B, C and A, C, D. Only triangle B, C, D can be generated. Images from Gibson [17].

one of its 26 neighbours does not belong to the region. For each of these boundary voxels, each of the eight corners of the voxel are checked for whether a node should be created. A Node will be created, if at least one of the eight voxels adjacent to the corner does not belong to the region. After the creation, an initial normal will be determined, based on the layout of the voxels adjacent to the node which belong to the feature. Figure 3.11(a) illustrates this in 2D.

When a node has been created, it will be linked to existing, neighbouring nodes. However, a link will only be established, if both nodes share at least one and less than four adjacent region voxels. Figure 3.11(b) shows a case in 2D, where two directly neighbouring surface net nodes are not linked together.

3.3.2 Triangle Generation

After initialization of the surface net, triangles will be produced where the nodes of the surface net represent the vertices of a triangle. There are 12 possible cases for generating triangles with neighbouring surface net nodes, which are illustrated in Figure 3.12(a). For each node, 4 possibilities for generating a triangle with neighbouring nodes exist along each plane in three–dimensional space. A triangle will only be generated if these conditions are fulfilled:

- The involved nodes required for one of the triangle cases actually exist.
- No triangle involving the current node and the diagonal node exists yet (Figure 3.12(b)).
- The diagonal node (if existent) shares at most one adjacent region voxel with the current node. This avoids triangles being generated inside the region.

The direction of the surface normal for each triangle is determined by the initial normals of its surface net nodes.

3.3.3 Surface Net Relaxation

The position of each of the surface net nodes will be relaxed one after another. During this process, the position of the node is moved towards an averaged position of its linked neighbours.

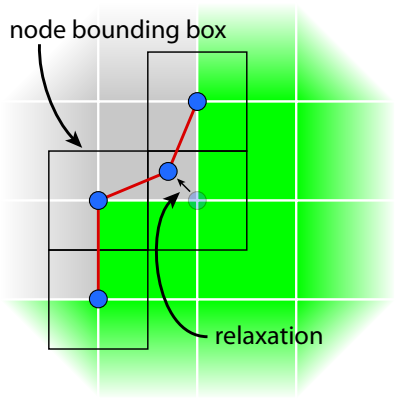


Figure 3.13: Constrained relaxation of the surface net nodes.

However, the position is constrained by a bounding box. The bounding box is centered around the initial position of the node and its size is the size of one voxel cell (Figure 3.13).

After each complete relaxation, the energy of the surface net is computed. Relaxation is then applied again, until the energy reaches a minimum (e.g. if the difference to the previous iteration falls below a certain threshold) or the maximum iteration count defined by the user was reached.

3.3.4 Distance Field Generation

After the relaxation is completed, the normals of the resulting triangles are updated, based on the new node positions. The normal of each node is then updated based on the normals of their adjacent triangles, using angle–weighted normals (see Figure 2.9(b)). Using the process described in Section 2.3, a distance transform of the triangle mesh from the surface net is generated.

Similar to the gradient integration method, the distance field is first initialized with positive infinity values outside of the region and negative infinity values inside of the binary segmented region. Then for each triangle of the surface net, the shortest distance to that triangle is evaluated for voxels in close proximity. It is determined whether a voxel is closest to an edge, a vertex or the surface of the triangle and then the distance to that feature of the triangle is taken. The sign is calculated by using the available normals of the triangle. If the voxel is closest to a vertex of the triangle, the dot product of the vertex normal and the direction vector from the vertex to the voxel is taken. If the voxel is closest to an edge, the average of the adjacent triangle normals is used for the calculation. If the voxel is closest to the surface of the triangle, the normal of the triangle is used. The value for each voxel is only updated, if the absolute value of the new distance is smaller than the absolute of the old distance. After updating the distances to each triangle, the distance field is then processed exactly the same as described on page 38.

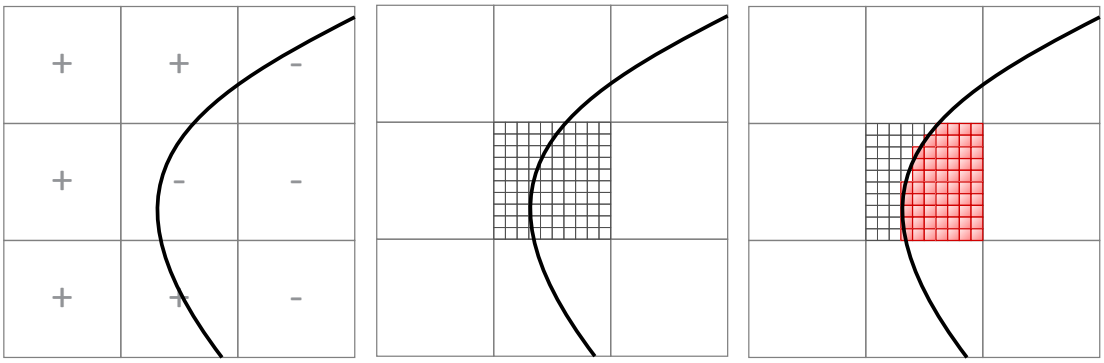
CHAPTER 4

Measurement and Quantification

4.1 Volume Sampling

There are different approaches for calculating the actual subvoxel volume of the areas in a distance field that are classified as inside the surface. A common approach is to create a mesh from the volume data at the ISO–surface location, the zero–distance ISO–surface in the case of the distance field. This can be done through the *Marching Cubes* method as described in Lorensen and Cline [37]. A more advanced method is presented by Raman and Wenger [46], with an extend lookup table and an improved algorithm.

For this work, a different approach was chosen for calculating the subvoxel volume of each individual feature: approximating the volume by sampling the distance field with a high res-



(a) (b) (c)

Figure 4.1: (a) Resulting distance field describing the subvoxel boundary and the inside and outside of the features. (b) Subdividing a voxel cell into n subvoxels. (c) Evaluating the distance field for each subvoxel and counting all the subvoxels that are „inside“, i.e. where the evaluated distance value is $\geq d_{zero}$.

olution. Basically each voxel cell of the regular volume (Figure 4.1(a)) is subdivided into n subvoxels (Figure 4.1(b)). For each of these subvoxels, the distance field is reconstructed using tri–cubic interpolation. Then simply all subvoxels are counted, which have a distance value $\geq$ the zero–distance value of the distance field (Figure 4.1(c)). Naturally, higher resolutions (higher values of n) lead to more accurate approximations of the volume. High quality filtering of the distance field at such high resolutions (e.g. 20 or 40 times higher than the original resolution of the dataset) would lead to long computation times on the CPU. Therefore it is helpful to accelerate this process by using the GPU for example.

Implementation

A simple approach was taken, using only a regular fragment shader in order to achieve a hardware accelerated volume quantification. The volume is processed slice by slice along an arbitrary axis (e.g. the z–axis). Using an orthogonal projection, a viewport aligned rectangle in the dimensions of the volume along that axis is drawn to trigger the fragment computation for each voxel along the current slice. Additionally, the texture coordinates t of the rectangle will range from $(0, 0, \frac{z+0.5}{dim_z})$ to $(1, 1, \frac{z+0.5}{dim_z})$, which provides the fragment shader with the correct voxel position information in the form of volume texture coordinates. The resulting texture from the fragment shader consists of two channels, one for the subvoxel volume count and one for the feature id. The results from each slice are accumulated for each feature accordingly, in order to get the final total subvoxel volume. Algorithm 4.1 outlines this process.

input : Distance field with feature id information
output: Subvoxel volume per feature

create framebuffer with size (dim_x, dim_y)
set orthogonal projection and viewport with size (dim_x, dim_y)
bind framebuffer and fragment shader
pass parameters and volume to fragment shader
foreach *slice along* z*–axis* **do**
    clear framebuffer
    draw viewport aligned rectangle with size (dim_x, dim_y)
    read and process the framebuffer
end

Algorithm 4.1: Method for super sampling the volume in order to calculate the volume for each feature in the distance field.

Parameters for the fragment shader consist of the number of samples n_s to be taken along each axis, the actual distance field volume texture V and the ISO–value for the zero–distance ISO–surface d_{zero}. Using the mapping described on page 38, this ISO–value will be 0.5. Apart from further parameters for cubic filtering for instance, the shader also needs to know the voxel–distance $dist_i$ in each direction in texture space ($[0..1]$). Using this voxel–distance, the starting position s_0 is calculated as well as the step size $step_i$ in each direction for determining the

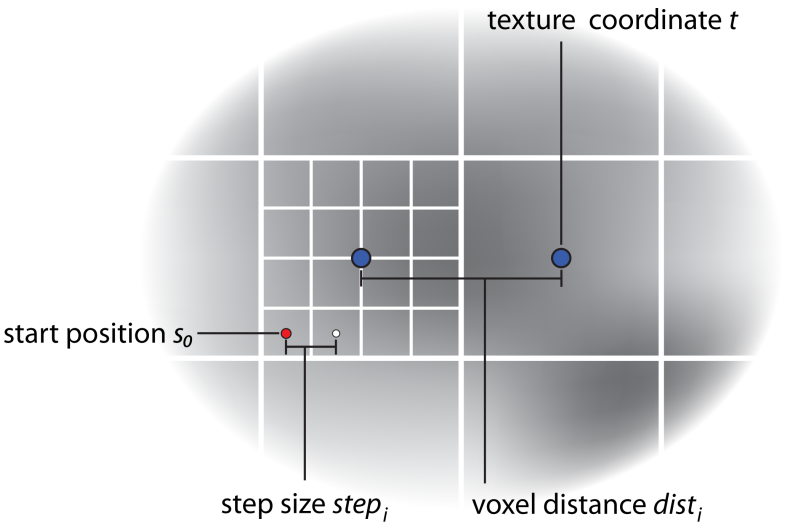


Figure 4.2: Illustration of parameters and variables during the sampling process. The start position s_0 and step size $step_i$ is calculated using the voxel–spacing parameter $dist_i$, the number of samples n_s and the texture coordinate t of the current voxel.

sample positions within the voxel (Figure 4.2). The step size is calculated by $step_i = \frac{dist_i}{n_s}$ and the start position by $s_0 = t - \frac{dist_i}{2} + \frac{step_i}{2}$. The sample positions in all three dimensions of the voxel are processed in nested loops as shown in Algorithm 4.2.

For each sample position, the distance field value is reconstructed with cubic filtering, using the method described in Section 2.4. If the resulting distance value is greater or equal (on the surface) than the zero–distance value, the subvoxel will be considered inside the feature and the subvoxel count v is incremented. At the end, the shader stores the subvoxel count in the first channel of the output texture and the feature id of the processed voxel in the second channel. When the resulting texture is analyzed, the resulting subvoxel value is simply divided by n_s^3. This subvoxel volume is then simply accumulated for each feature while processing the volume slice by slice, using the feature ID information in the second channel of the output texture.

4.2 Measure Tools

Simple measure tools can be helpful during the quantitative analysis of spatial relations in visualizations of industrial or medical CT or MRI data. Such measuring tasks can include:

- Determining distances between features of the data or their extents.
- Measuring angles between elongated features.
- Estimating volumes of features.

input : n_s, V, d_{zero}
output: v

calculate starting position s_0 *and step sizes* $step_i$;
$v \longleftarrow 0$;
for $x \longleftarrow 0$ **to** n_s **do**
  for $y \longleftarrow 0$ **to** n_s **do**
    for $z \longleftarrow 0$ **to** n_s **do**
      $$s \longleftarrow s_0 + \begin{pmatrix} step_x \cdot x \\ step_y \cdot y \\ step_z \cdot z \end{pmatrix};$$
      $d \longleftarrow fetchCubic(s, V)$;
      if $d \geq d_{zero}$ **then**
        $v \longleftarrow v + 1$;

Algorithm 4.2: Counting the subvoxel volume in a fragment shader.

In order to create such measurements, there must be the possibility to create and manipulate these tools within the visualization framework. When discussing the interaction with measurement tools, the term *widget*, a combination of *window* and *gadget*, is well established and characterizes the interaction facility with the measuring tool and its behaviour.

In the following sections three different measure widgets are presented that were implemented for this work. Section 4.2.1 explains how the distance field of the previous chapter can be used for subvoxel measurements and Section 4.2.2 describes a method for integrating arbitrary geometry into the volume visualization with GPU–based ray–casting. Preim et al. [45] provides further reading for measurement tools integration, in particular how 3D–interaction can be done and how some measurements could be automated.

Generally, these measure widgets should have a basic 2D representation and interaction within the slice viewer of the volume visualization, as well as a 3D representation within the volume rendering view. Simple estimates can be created on a per–slice basis, while more complex measurements across the 3D space of the volume are also possible.

Distance Measurements The most basic measuring tool would be a widget for determining the distance between two arbitrary points. Usually the distance between certain features of the volume data is of interest for the analyst. Such features could be tumors in medical data for example or air inclusions and certain other materials within an industrial work part. The distance line widget can also be used to quickly assess the extents of objects within the dataset along arbitrary axes (Figure 4.3).

Radius Measurements While the process of measuring the radius is the same as for measuring the distance, the dedicated radius widget visually helps to determine the approximate ra-

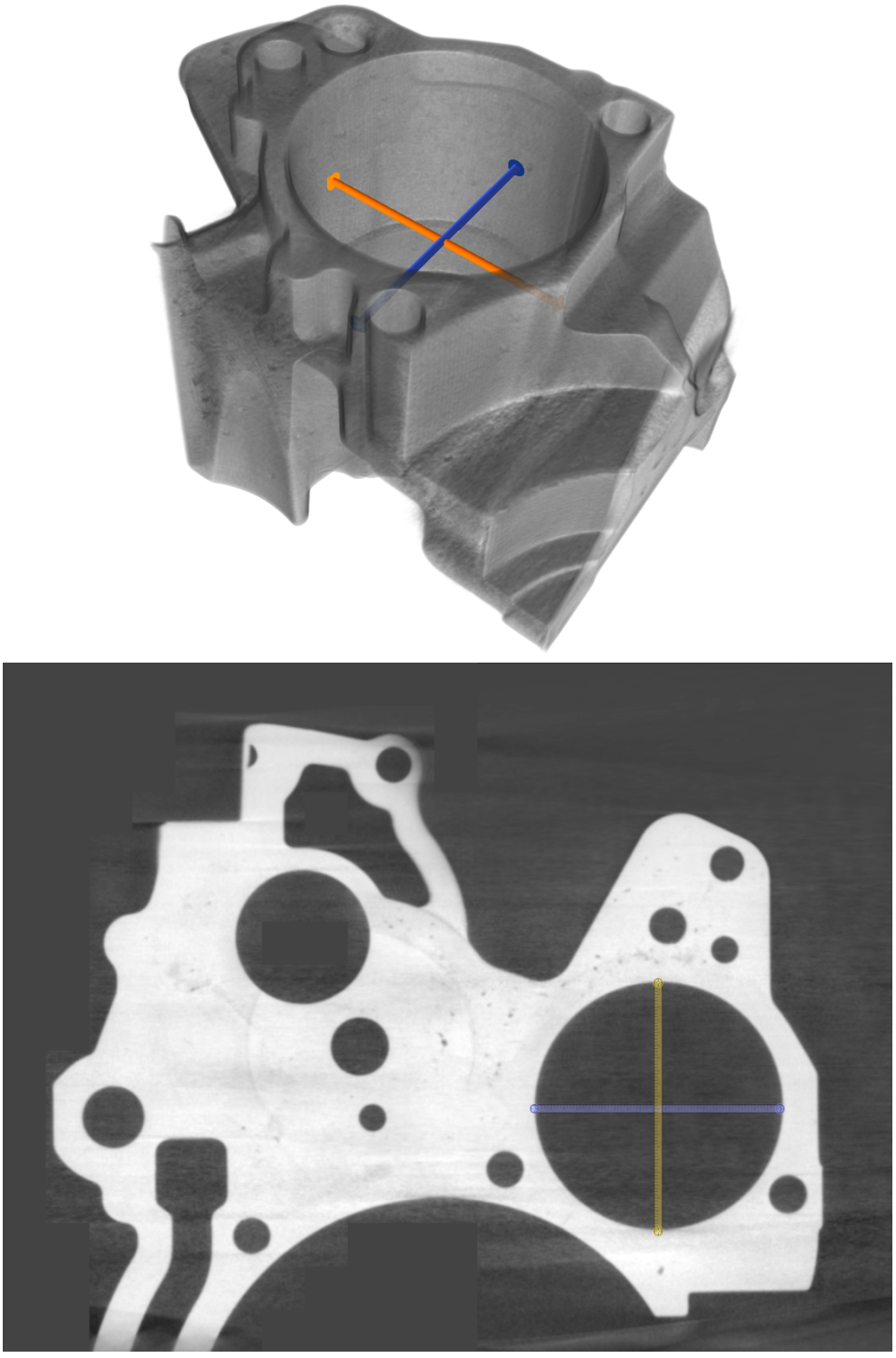


Figure 4.3: Distance Measurement Tool.

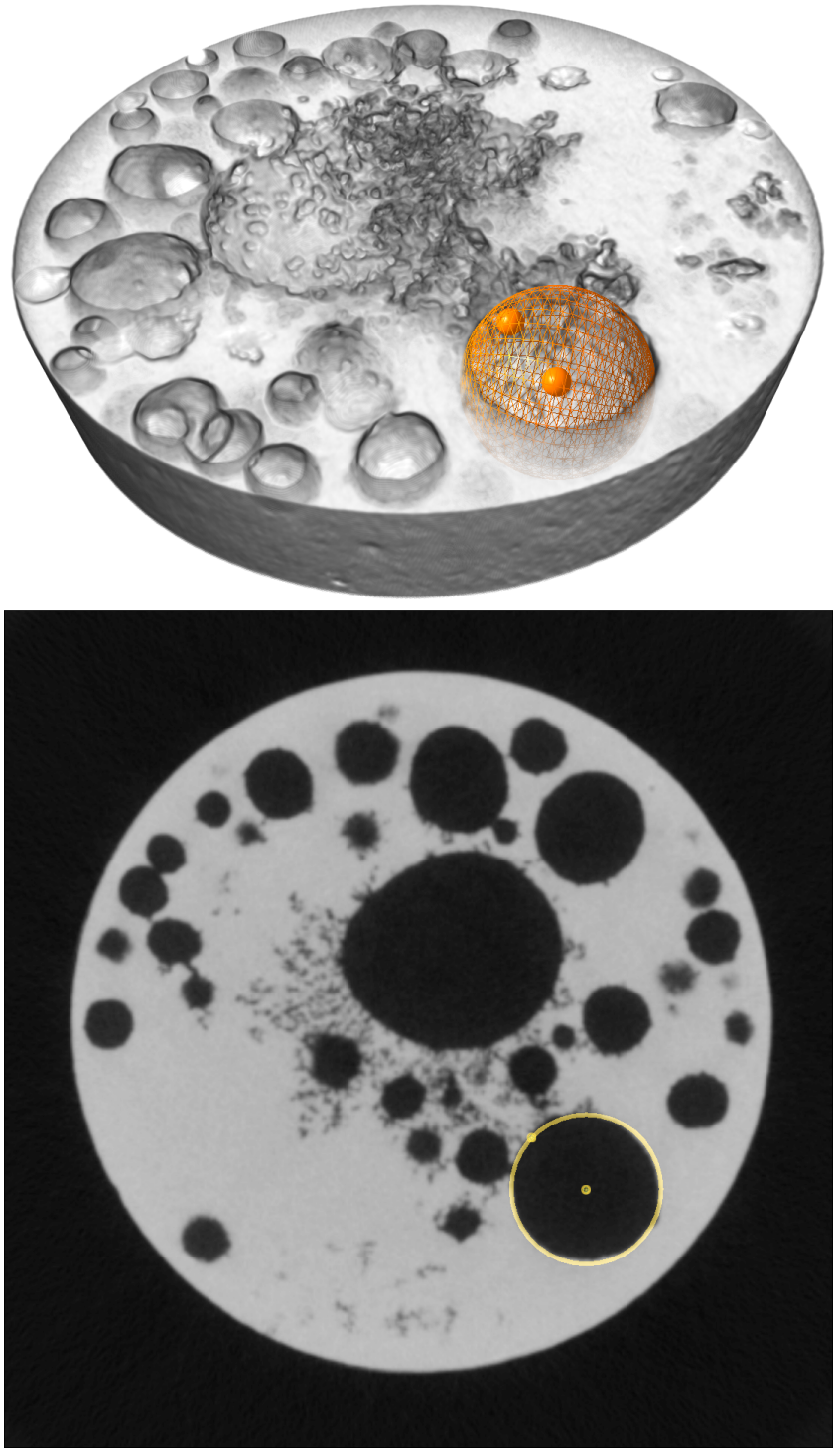


Figure 4.4: Radius and Volume Measurement Tool.

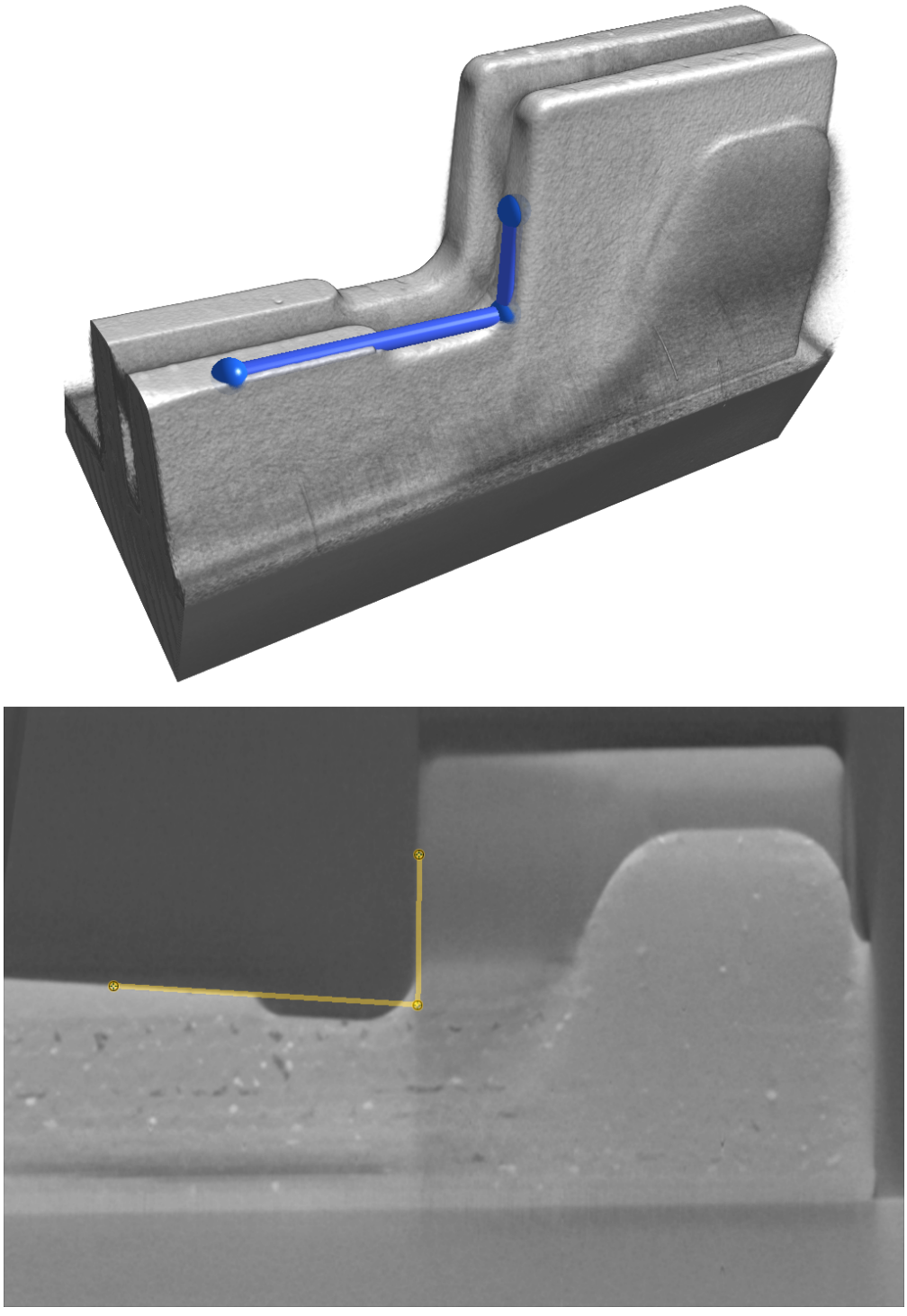
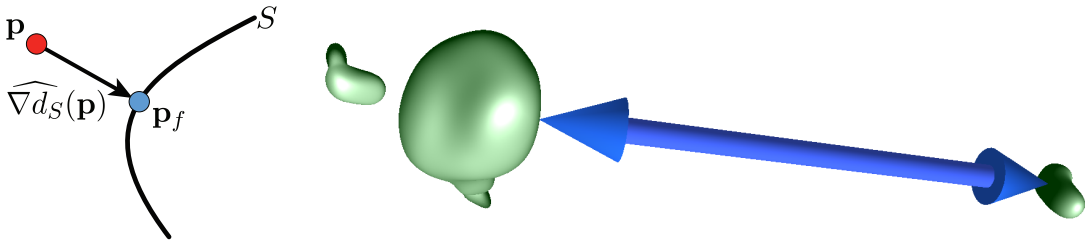


Figure 4.5: Angle Measurement Tool.



(a) (b)

Figure 4.6: (a) Projection of a point in the distance field to its surface using Equation 4.1. (b) Visualization of a 3D measurement widget within the volume view, snapped to the subvoxel boundary of two features.

dius of spherical features as well as directly displays the volume of the sphere defined by the radius (Figure 4.4).

Angle Measurements Angle measurements are often used for medical data, for example between two bones or teeth. Within industrial CTs an angular measurement widget can be used to determine the angle between two elongated features of the data (Figure 4.5).

4.2.1 Snapping to the Subvoxel Surface

In order to improve the accuracy of the measurement process, it would be useful to guide the interaction using information from preliminary segmentation operations. For instance in order to measure distances between segmented features more easily, the endpoints of a measure widget could automatically snap to voxels belonging to a feature. Subvoxel accuracy can be introduced by snapping to the zero–distance ISO–surface of the distance field from the method described in Chapter 3. This can be done easily by projecting a point within the distance field to the surface of the distance field. In a distance field that is well defined throughout the whole volume, any arbitrary point $\mathbf{p}$ can be projected onto the surface S of the distance field using the simple formula

$$\mathbf{p}_f = \mathbf{p} - \widehat{\nabla d_S}(\mathbf{p}) d_S(\mathbf{p}), \tag{4.1}$$

where $\widehat{\nabla d_S}(\mathbf{p})$ is the normalized gradient of the distance field at position $\mathbf{p}$ and $d_S(\mathbf{p})$ is the distance to the surface at position $\mathbf{p}$ (Figure 4.6(a)). $\mathbf{p}_f$ is the projected point on the surface S and sometimes called the *foot point* of $\mathbf{p}$ [26].

The resulting distance field from Chapter 3 only has actual distance values near the boundary itself. This is not a problem, since snapping should only occur near the boundary anyway. Figure 4.6(b) shows the three–dimensional visualization of a distance measurement widget within the volume view. It is snapped to the subvoxel boundary of two features within the cast housing dataset of Section 5.3.

4.2.2 Integration in Volume Rendering

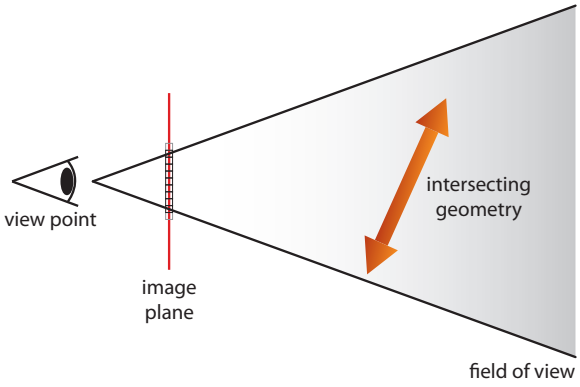
Volume visualization within the high–quality hardware volume renderer framework is done via GPU–based ray–casting. Since this is an image space method, arbitrary geometry (for rendering measurement widgets for instance) cannot be combined with the volume rendering, without making some modifications.

Basically, intersecting geometry should prevent rays from being created, where it occludes the volume behind it. And rays need to be stopped early, when they hit the intersecting geometry. One way to make these decisions is to compare the view depth of the intersection geometry with the depth of a sample along one ray [19, page 286]. This method was also used by Scharsach et al. [51]. Thus the process for integrating arbitrary intersection geometry into the volume visualization uses the following steps:

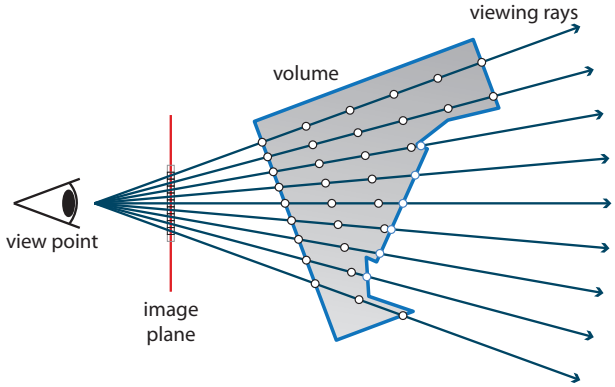
- Render the intersection geometry into a separate colour and depth buffer (Figure 4.7(a)).
- During rendering of the front faces, enable the depth test and use the same depth buffer as before. This way, no starting positions for the ray–casting process will be generated for fragments that are occluded by the intersection geometry.
- During rendering of the back faces, which creates the direction texture (see Section 2.2.1 and Figure 2.6), compare the depth of the intersection geometry (using the previously generated depth map) with the depth of the back face geometry. The smaller depth will be used in the direction texture, encoded in the alpha channel as the length of the ray during ray casting.
- Render the volume using the altered direction texture (Figure 4.7(b)) and blend the final image over the texture from the intersection geometry pass (Figure 4.7(c)).

In addition to the depth map texture, the fragment shader during the back faces pass (for generating the direction texture) needs to know the inverse model–view–projection matrix (including the transformation into the volume texture space). In order to compare the position of the intersection geometry with the position of the back–face volume geometry, the depth value is projected back into the homogeneous volume space. Algorithm 4.3 shows pseudo code for calculating the volume position from the depth map. MVP' is the inverse model–view–projection matrix, $depthMap$ the depth map texture and $winCoord$ the normalized coordinates of the current fragment.

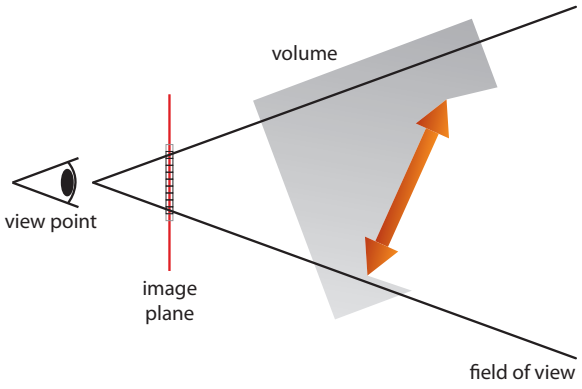
The final volume image after ray casting contains completely transparent fragments at positions where the intersection geometry is in front of the volume, and correct colour and transparency information at fragments where a ray hits the intersection geometry. This volume image can then simply be blended over the intersection geometry image.



(a)



(b)



(c)

Figure 4.7: Steps for integrating intersecting geometry with GPU–based ray–casting. (a) Creating a depth map of the intersection geometry. (b) Volume rendering with altered stop positions. (c) Blending of the final volume image over the intersection geometry scene.

input : MVP', $depthMap$, $winCoord$
output: $volPos$

```
// view space position for intersection geometry
```
1 $viewPos_{xy} \longleftarrow winCoord$;
2 $viewPos_{z} \longleftarrow texture(depthMap, winCoord)$;
3 $viewPos_{w} \longleftarrow 1$;

```
// transform to clipspace
```
4 $viewPos \longleftarrow viewPos \cdot 2 - 1$;

```
// back project to homogeneous volume space and normalize
```
5 $volPos \longleftarrow \mathrm{MVP}' \cdot viewPos$;
6 $volPos \longleftarrow volPos / volPos_{w}$;

Algorithm 4.3: Calculating the volume position from a depth map value.

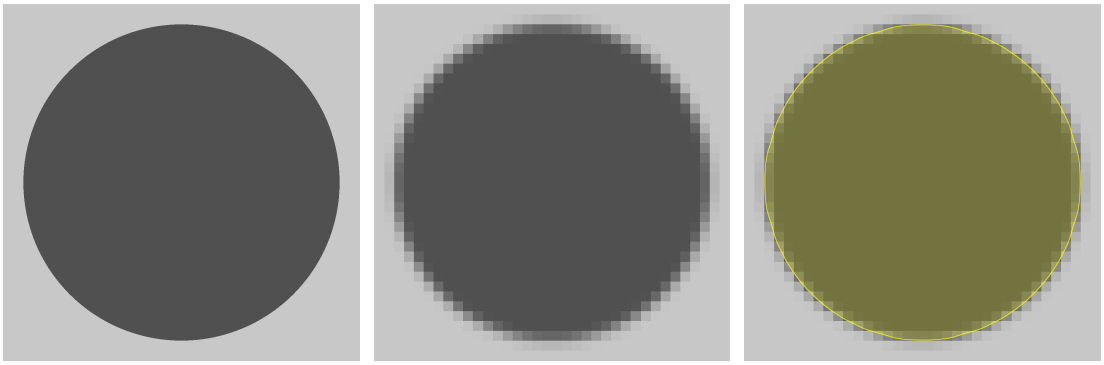
CHAPTER 5

Results and Conclusions

In this chapter some results and insights of the subvoxel boundary detection on artificial as well as on real, industrial datasets are provided. Section 5.1–5.4 show visual and measured results, in comparison with the binary segmentation and a smoothing technique. Section 5.5 outlines the general performance, problems and limitations of the subvoxel boundary detection method presented in this work.

5.1 Proof of Concept

The subvoxel boundary detection presented in this work basically provides the expert with an additional tool to measure and quantify the volume. However, aside from empirical analysis of



(a) (b) (c)

Figure 5.1: Artificial dataset for proof–of–concept. (a) Slice through original volume. (b) Downsampled volume. (c) Resulting subvoxel contour.

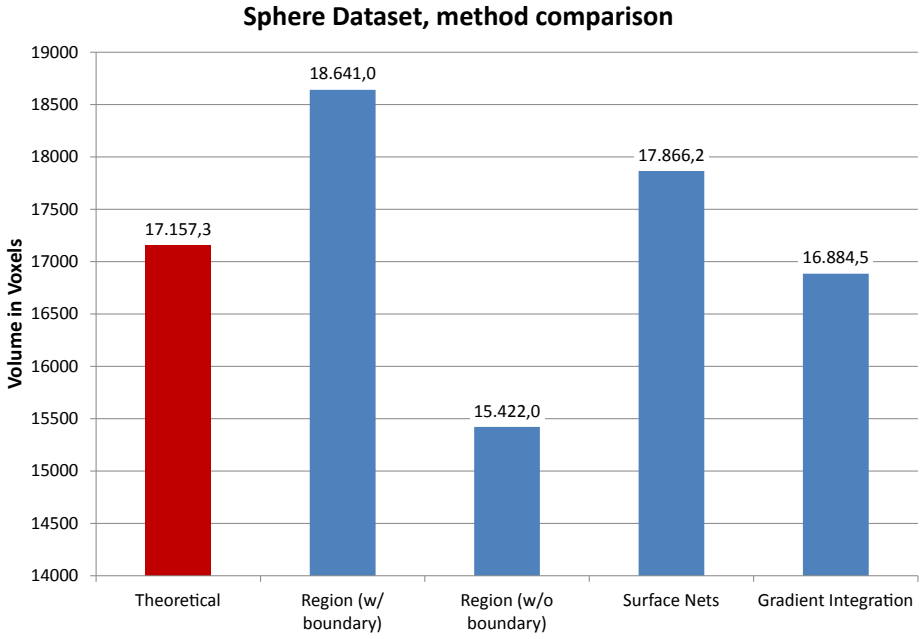


Figure 5.2: Volume quantification results. The original, theoretical volume (in voxels) is compared to results from region growing, the subvoxel boundary detection through gradient integration and smoothing through Constrained Elastic Surface Nets.

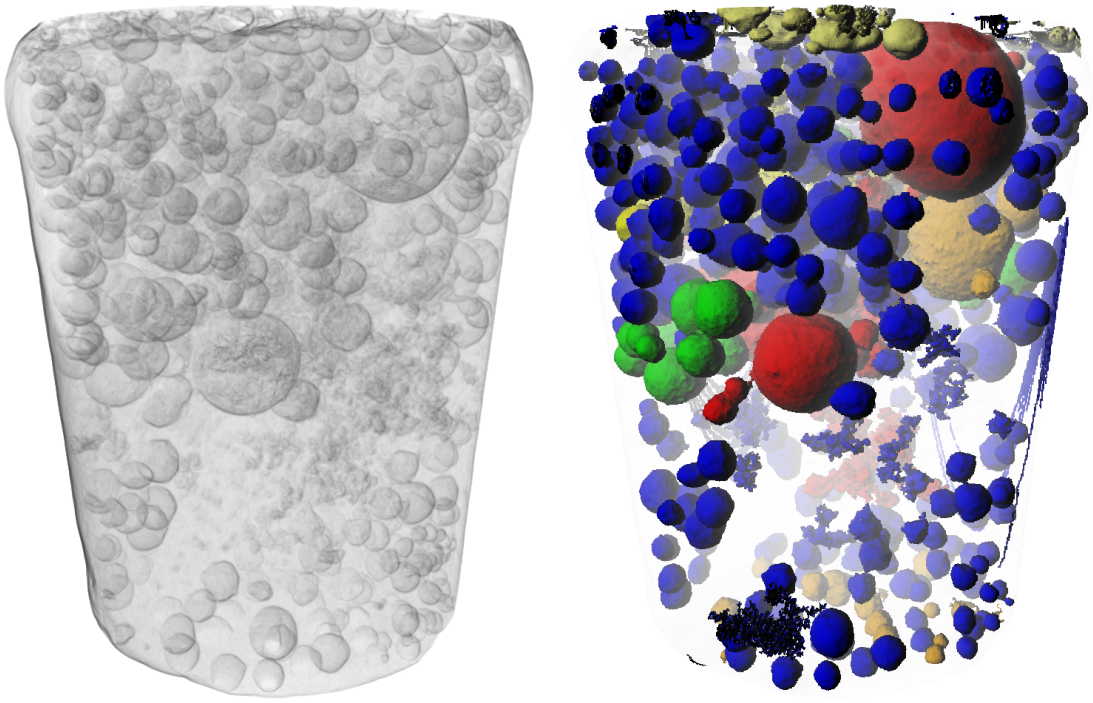
the result, it is impossible to know whether the results in a real dataset are actually accurate or more accurate than other quantification methods.

In this section a proof–of–concept is provided, to show that the subvoxel boundary detection method used in this thesis does indeed have the potential to lead to more accurate results. This proof–of–concept is provided by generating an artificial dataset which includes an object (feature) of which the original volume is known. By simulating the loss of resolution and boundary blurring that occurs during the acquisition of a real dataset, it can be determined if the presented method can reconstruct the original boundary.

This artificial dataset is created in the following way: first a large volume is created, with a high density value. Inside this volume, a sphere of lower density is placed in the center, with a certain radius. This dataset is then downsampled using a Gaussian filter, to a much smaller volume, to simulate the loss of resolution. Region growing is then applied to the dataset and the subvoxel boundary calculations afterwards.

Figure 5.1(a) shows a slice through the original dataset. Its size is $2048 \times 2048 \times 2048$, the radius of the sphere is 512 voxels. Figure 5.1(b) shows the downsampled volume with size $64 \times 64 \times 64$ (the images show a magnified section of the dataset). After applying the segmentation and subvoxel boundary detection, the resulting contour can be seen in Figure 5.1(c).

By comparing the results from the volume quantification (Figure 5.2), we can see that the subvoxel boundary detection through gradient integration comes closest to the theoretical vol-



(a) (b)

Figure 5.3: Reduced–Pressure–Test Sample. (a) Translucent direct volume rendering of the dataset. (b) Binary segmentation from region growing, with colour–coding depending on the size of each feature.

ume $V = \frac{4}{3}\pi 16^3$ inside the downsampled sphere, assuming the sphere inside the downsampled dataset would now have a radius of 16 voxels. The error based on the subvoxel volume is therefore $\approx 1.6\%$. The volume quantification through super–sampling of the dataset was done with 40^3 samples per voxel.

5.2 Reduced–Pressure–Test Sample

One important factor for the quality of aluminium or copper casts is „gas porosity“ which is caused by concentrations of hydrogen within the melt. In order to gain optimum quality, the amount of hydrogen dissolved in the melts must be known prior to casting. One of the techniques to quantify hydrogen in liquid aluminium is the so called „reduced–pressure–test“ [32]. During this process about 30 cm^3 of the melt is solidified at a pressure of 8000 Pa. This causes the gas within the melt to form pores and the shrinkage of the metal during the solidification creates cavities.

Figure 5.3 shows such a reduced–pressure–test sample (RPTS). Figure 5.3(a) shows a direct volume rendering of the data set, with a 1D transfer function for a translucent appearance.

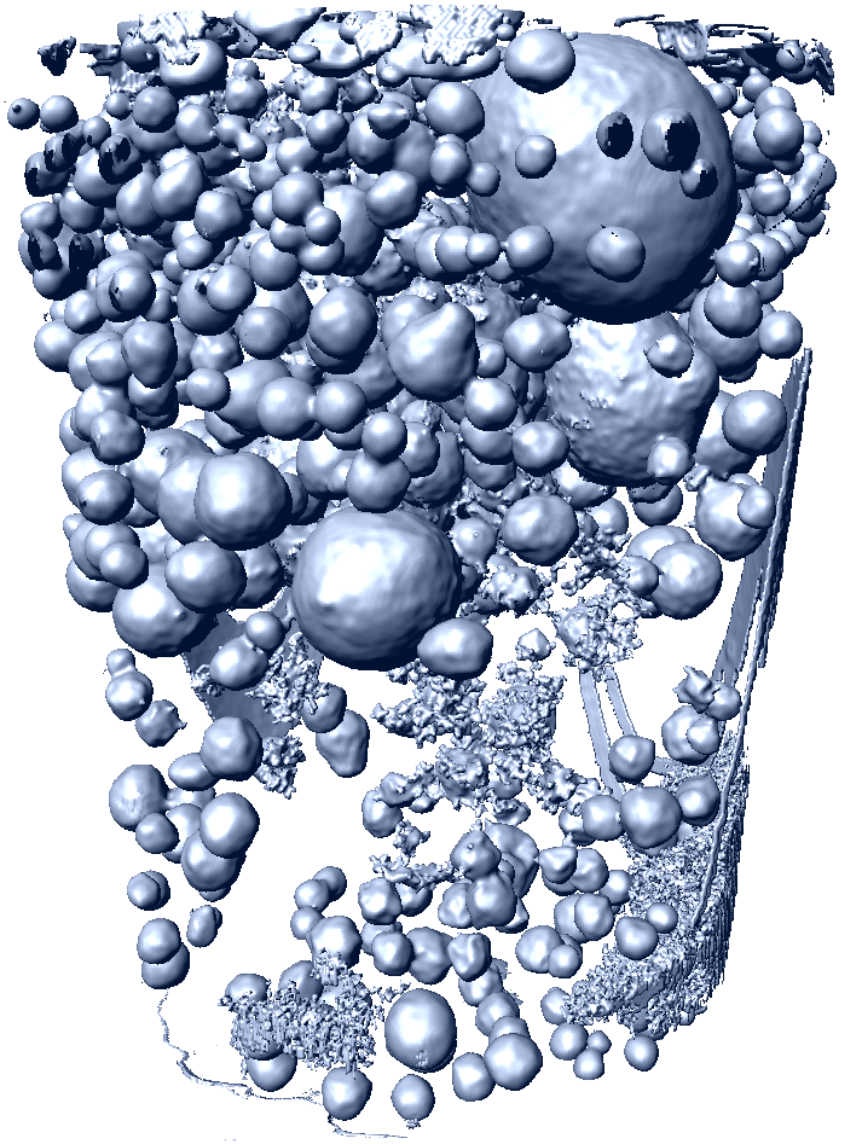


Figure 5.4: Resulting distance field of the RPTS.

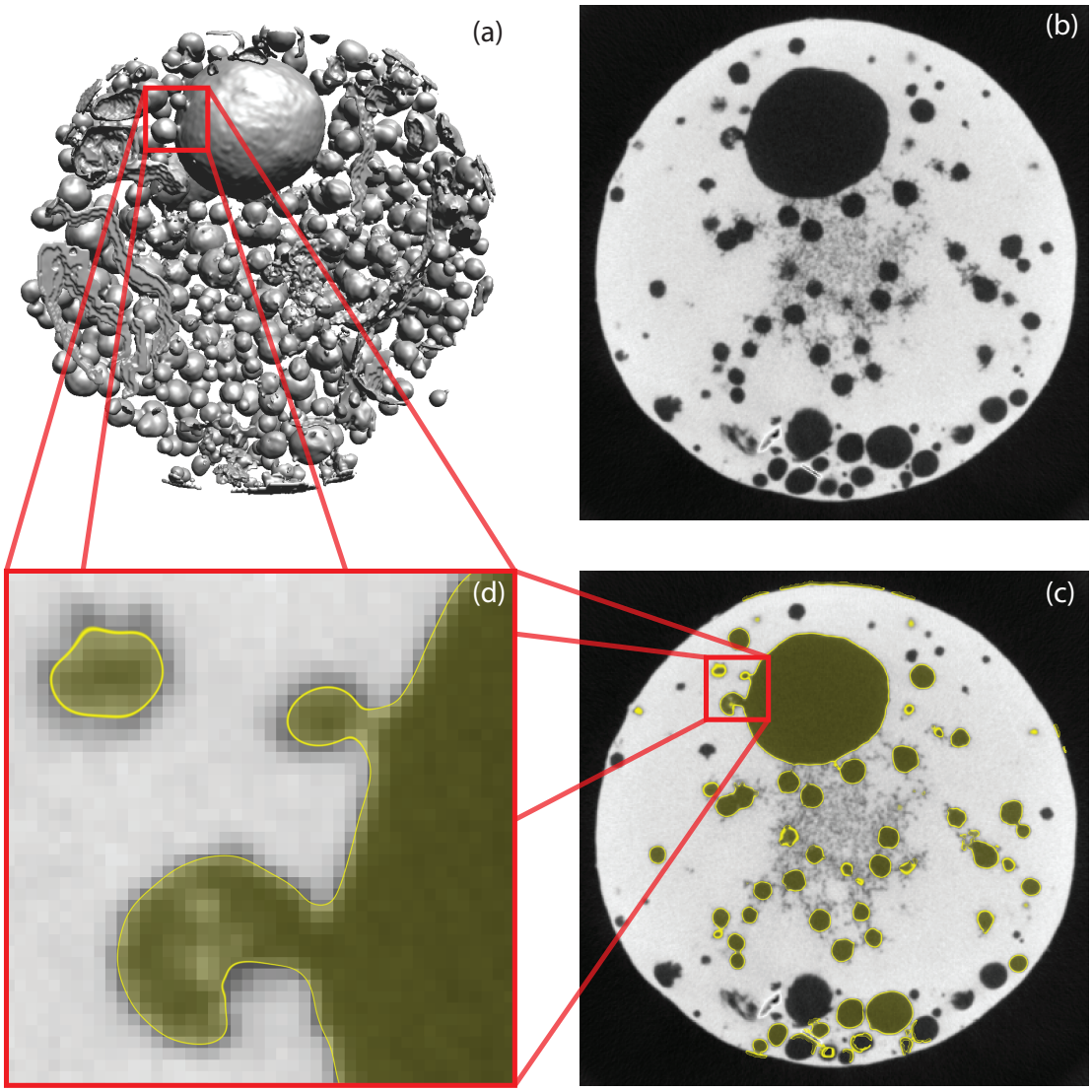


Figure 5.5: Detail of the RPTS data set. (a) The subvoxel surface of the features of a section of the dataset. (b) A slice through the data, with mainly pores being visible. It also shows a prominent feature of the dataset, the largest pore, which is also partially connected to some neighbouring pores. (c) The subvoxel boundary visualized in this slice. (d) Detail of the dataset and the subvoxel boundary.

Figure 5.3(b) shows the binary segmented features after the region growing process. Visible are the large spherical shapes of the pores as well as the smaller cavities in the lower part of the dataset. Figure 5.4 shows the zero–distance ISO–surface rendering of the distance field from the subvoxel boundary detection of all detected features.

Figure 5.5 gives a more detailed view on the subvoxel boundary detection result. The top left

		feature ID	
		#135 (Fig. 5.6(a))	#309 (Fig. 5.6(b))
segmented region	voxels	715378.00	6181.00
	mm^3	876.27	7.57
subvoxel boundary	voxels	712596.38	4903.37
	mm^3	872.86	6.01
surface net	voxels	711950.56	4321.19
	mm^3	872.07	5.29

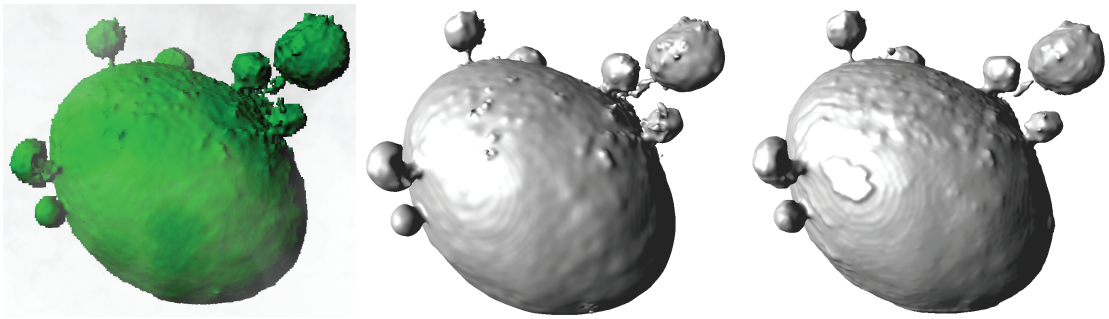
Table 5.1: Volume quantification comparison of two specific features from the RPTS dataset. Volume size: $373 \times 377 \times 512$. Voxel size: $0.107 \times 0.107 \times 0.107$ mm.

image again shows the direct volume rendering of the distance field ISO–surface in a particular section of the dataset. The top right image shows a slice through the data of this section. The bottom right image shows the same slice, but with the zero–distance ISO–surface visualized. The yellow contours represent the subvoxel boundary, while the transparent yellow areas within represent the insides of the features, as defined in the distance field. The bottom left image shows a detail within that slice where it can be seen more closely, how the contour follows the boundary between the pore and its surrounding material. The feature presented in this detail is the largest pore within the sample (also seen in Figure 5.6(a)). At some points on its surface, it is even very narrowly connected to neighbouring pores.

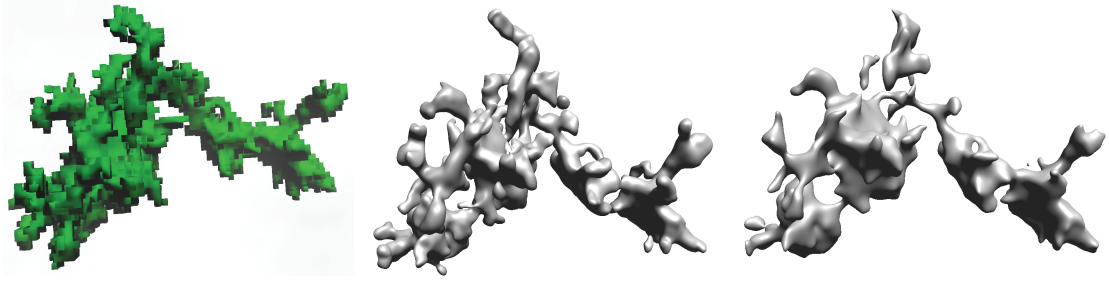
During the analysis, quantifications of these individual features are of importance to the expert. Basic statistical properties can be obtained from the region growing result, which includes the size in voxels, the position of each feature within the dataset, the surface area of the binary region as well as the feature's extents. The subvoxel boundary detection presented in this thesis provides additional results for quantification of the volume for each feature. These results are more accurate, since the original boundary of the feature is reconstructed and sampled more closely. Table 5.1 shows different volume quantification results for two specific features of the dataset.

Figure 5.6 shows the visual comparison between the different quantification methods for these features. Shown from left to right are: region growing result, gradient integration distance field and surface net distance field. The volume quantification through super–sampling of the dataset was done with 20^3 samples per voxel.

Generally, the subvoxel boundary detection works really well for this dataset. The boundaries of the large, spherical pores can be reconstructed without significant problems. Cases during the gradient integration where no valid result can be found in either direction do not have a big impact on the overall result. The original voxelization of the binary segmented region covers these cases sufficiently. Even the boundary of the much smaller and more importantly thinner cavities can be reconstructed in most cases. However, there are definitely more ambiguities during the search for the subvoxel boundary. A wrongly chosen minimum gradient magnitude threshold can have a severe impact on the result for these features.



(a) Feature #135



(b) Feature #309

Figure 5.6: Visual comparison between the segmentation and quantification results. From left to right: region growing, gradient integration distance field, surface net distance field.

5.3 Cast Housing

Figure 5.7 shows a cast housing that is part of an engine block in the automobile industry. It consists of an Al-Si alloy which are generally used in a wide variety of fields in the industry. Such alloys usually have casting defects like porosity and inclusions. Apart from the chemical composition, such defects have a great influence on the structural and mechanical properties of the cast. During the analysis of these work parts, properties like the total volume of such voids are evaluated, as well as their distribution and individual size.

Table 5.2 shows the volume quantification results of three particular features in the dataset and Figure 5.8 shows a visual comparison of the segmentation results. The top row shows a direct volume rendering of these features with a manually defined 1D transfer function, the second row shows the binary segmentation through region growing, the third row shows the ISO–surface rendering of the resulting distance field from the subvoxel boundary detection and the bottom row shows a detail of this subvoxel boundary in a slice through the volume.

Each of the presented features provides a challenge to the implemented subvoxel boundary detection. Manual inspection of the data for feature #5 (Figure 5.8(a)) suggests, that this void might actually continue on in an elongated way at the bottom of the feature. This appendix consists of higher densities than the homogeneous area of the feature itself, but it is unclear whether it represents a very narrow void that is connected to feature #5, or another material

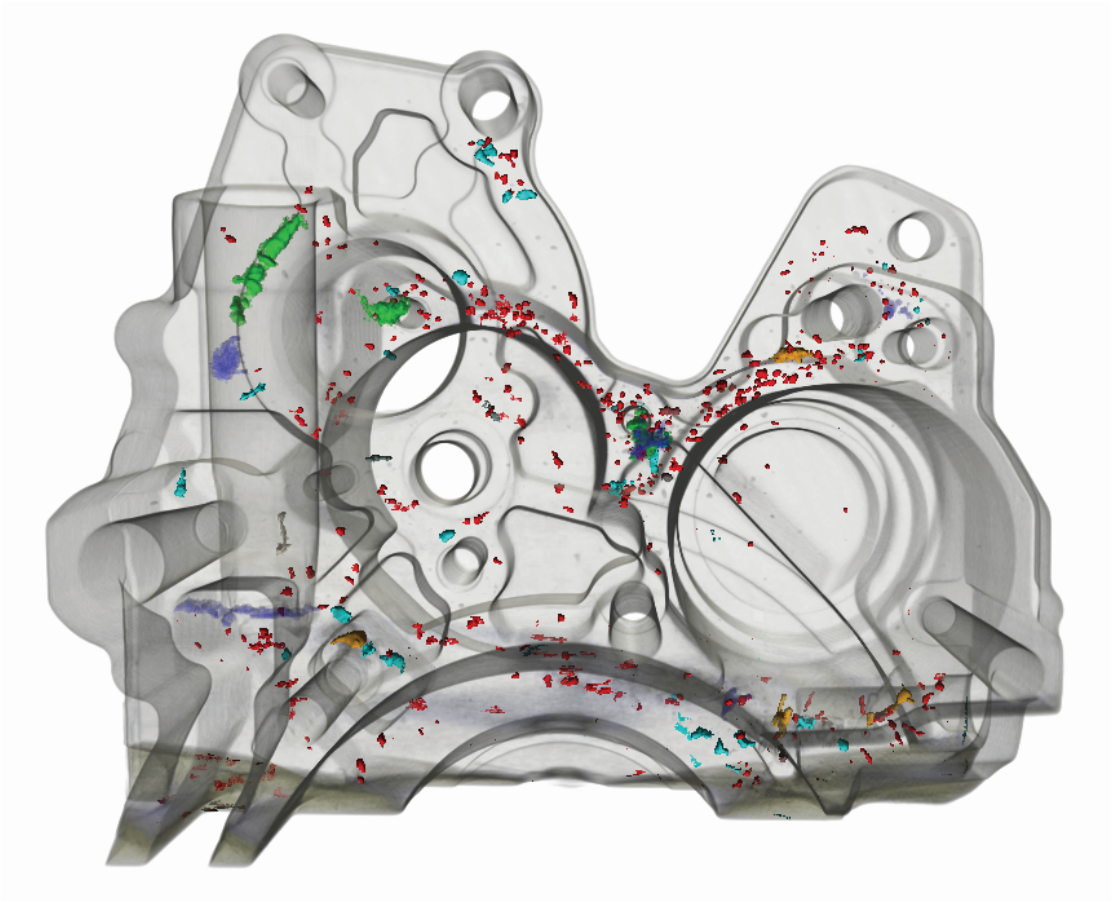
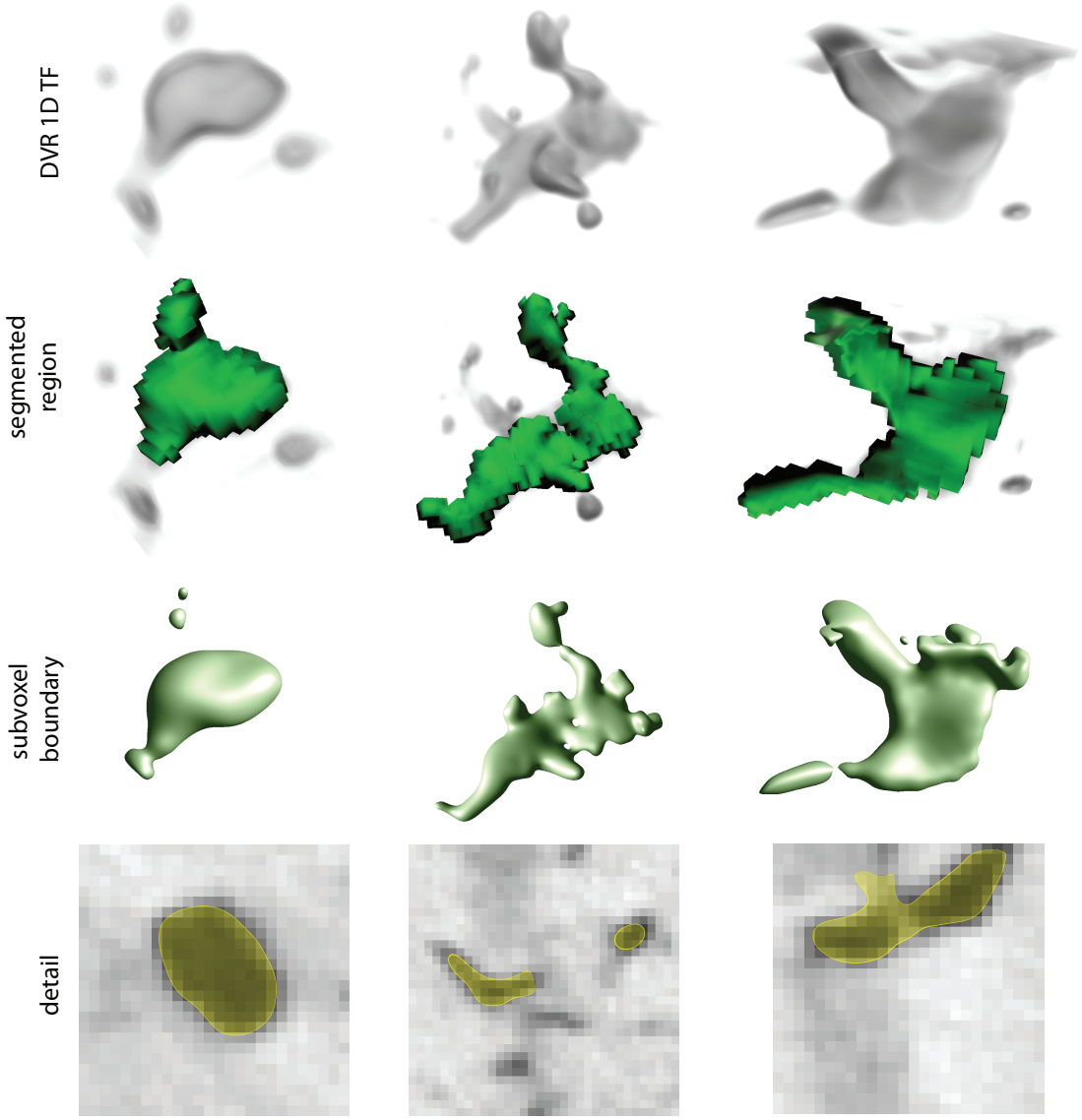


Figure 5.7: Volume visualization of a cast housing, including visualization of features. Image from Hadwiger et al. [20].

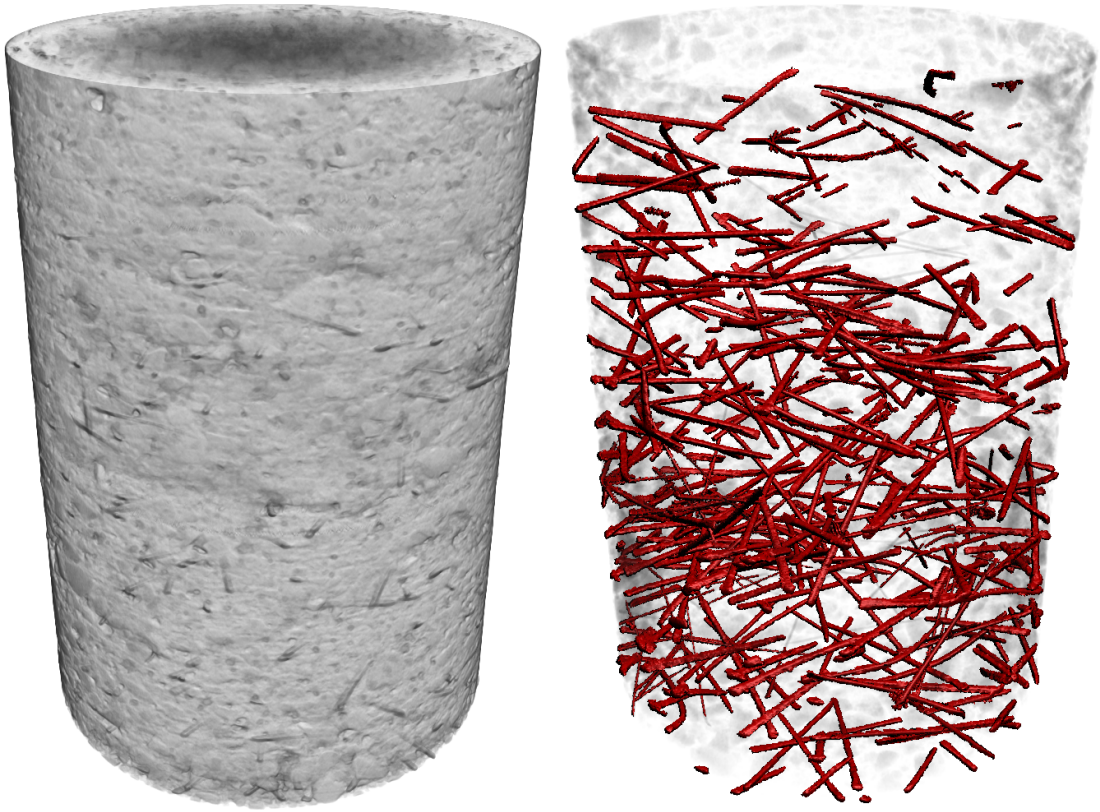
		feature ID		
		#5 (Fig. 5.8(a))	#14 (Fig. 5.8(b))	#15 (Fig. 5.8(c))
segmented region	voxels	620.00	604.00	774.00
	mm^3	620.00	604.00	774.00
subvoxel boundary	voxels	455.33	463.73	626.01
	mm^3	455.33	463.72	626.01
surface net	voxels	465.97	289.35	507.39
	mm^3	465.97	289.35	507.39

Table 5.2: Volume quantification comparison of three specific features from the cast housing dataset. Volume size: $667 \times 465 \times 512$. Voxel size: N/A.



(a) Feature #5 (b) Feature #14 (c) Feature #15

Figure 5.8: Visual comparison of three different features within the cast housing dataset.



(a) (b)

Figure 5.9: (a) Steel Fibre Reinforced Sprayed Concrete (SFRSpC) dataset and (b) its visualized steel fibres.

in itself. During the segmentation, these voxels did not become part of the feature. Since the subvoxel boundary detection searches for the subvoxel boundary near the surface of the binary segmented region, this part of the feature is missing from the distance field as well.

Feature #14 (Figure 5.8(b)) has a rather complex topology, much like the cavities in the RPTS dataset. It is very narrow, which could lead to erroneous result positions or inside–outside classifications for individual voxels. Feature #15 (Figure 5.8(c)) has similar properties, but it additionally consists of some areas, that have a wide transition into the density of the surrounding material. In these parts of the feature, the subvoxel boundary is found far beyond the surface of the binary segmented region.

5.4 Reinforced Sprayed Concrete

Figure 5.9(a) shows a dataset of Steel Fibre Reinforced Sprayed Concrete (SFRSpC). Steel fibre reinforced concrete is a highly important material in the building industry. Its application as sprayed concrete has been developed in more recent years. The steel fibres provide an additional

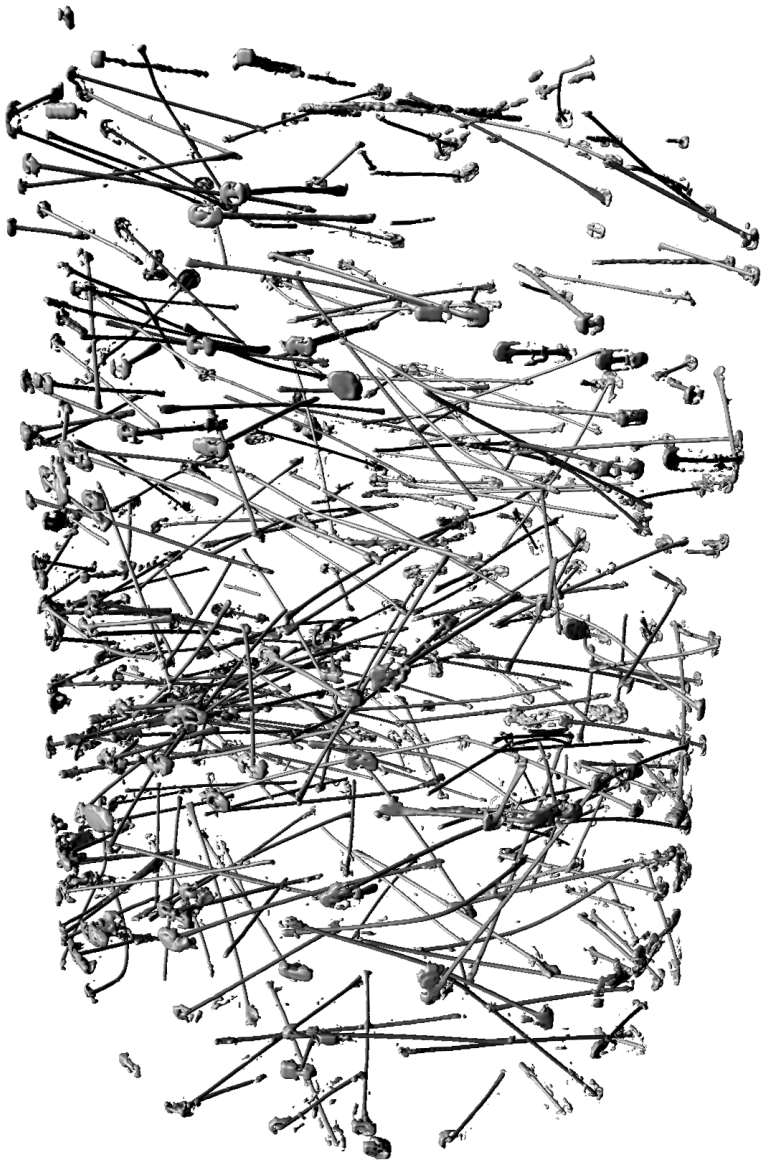


Figure 5.10: The resulting subvoxel boundaries of the fibres within the SFRSpC dataset.

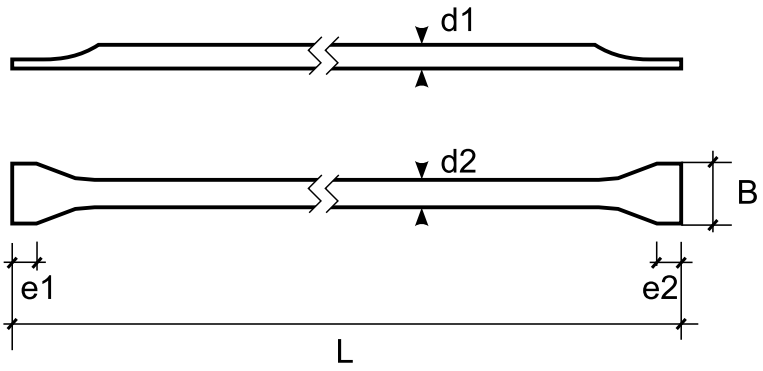


Figure 5.11: Steel fibre type TrefilArbed FE 65/35. Image from Wikipedia [61].

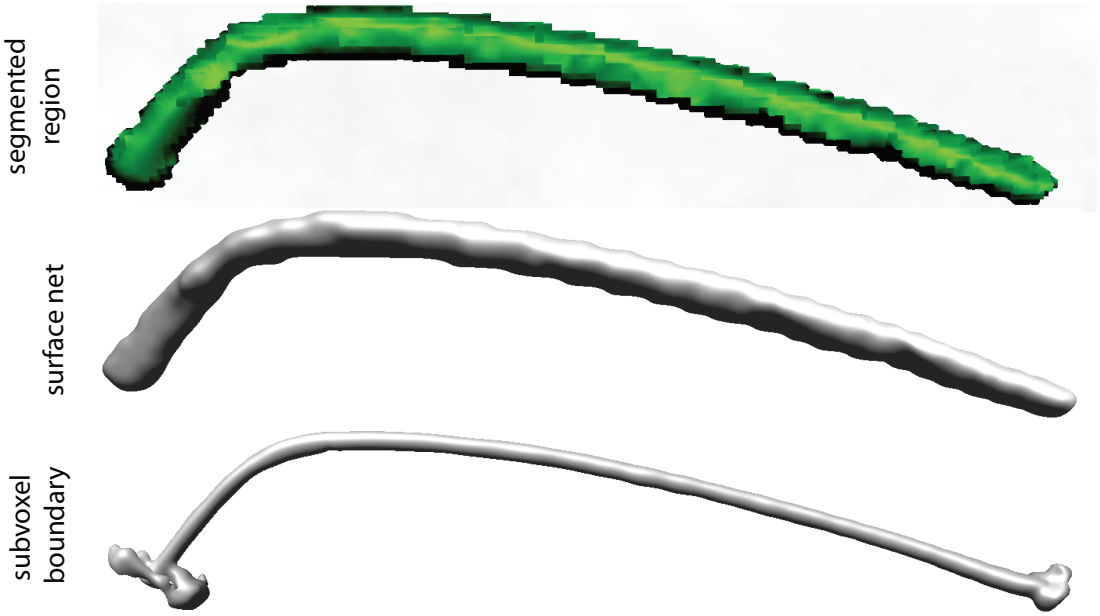


Figure 5.12: Feature #71, a bent fibre.

reinforcement to the sprayed concrete and their purpose is to absorb tensile force once cracks in the concrete are formed. Distribution and alignment of these fibres are important quality criteria and need to be verified. Drill core samples are taken from the sprayed concrete and then scanned via CT. The steel fibres can be automatically identified within the sample, using segmentation. Figure 5.9(b) shows a visualization of these automatically segmented fibres and Figure 5.10 shows the result from the subvoxel boundary detection.

There are different types of steel fibres, with various lengths and shapes. The fibres used in the aforementioned dataset are of the type TrefilArbed FE 65/35. Fibres of this type have

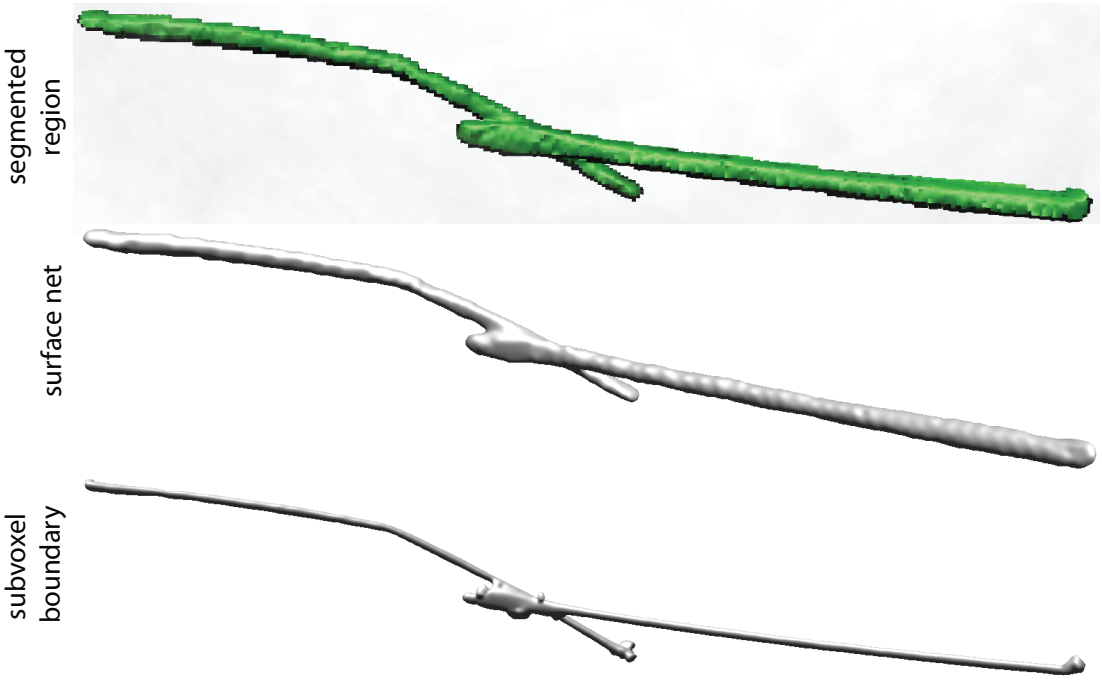


Figure 5.13: Feature #140, two adjacent fibres.

a diameter **d1** of > 0.5 mm and **d2** < 0.9 mm, as depicted in Figure 5.11, with an overall diameter of 0.65 mm. Their length is 35 mm [9]. Naturally, these appear as very thin structures within the dataset and provide a challenge for accurate quantification. Figure 5.12, 5.13 and 5.14 show results for different fibres. A visual comparison between the different methods presented in this thesis is provided as before. The top image in each case is the binary segmentation of the region growing step. The segmentation of the fibres suffers from heavy over–segmentation, meaning that the segmentation goes too far beyond the actual boundary of the material (see also Section 5.5.4). The second image is the resulting distance field from the constrained elastic surface net. Since the surface net always contracts the original region, the result appears smaller and smoother, but it is still too big. The third image shows the resulting distance field of the subvoxel boundary detection, which provides a more accurate representation of the fibres.

Figure 5.12 shows a fibre, that bent during the processing of the concrete. Its left end actually pokes out the surface of the sample, resulting in a few errors during segmentation and the subvoxel boundary detection. Figure 5.13 actually shows two fibres, that cross each other very closely at their ends. This results in a merged segmentation and also causes some problems during the subvoxel boundary detection. Figure 5.14 shows an almost straight fibre, with the shape of its ends being vaguely perceptible.

Table 5.3 shows quantification results for the aforementioned features within the SFRSpC dataset. Included in this result are diameter measurements for each fibre at a point in its center. These measurements were taken with the distance measure widget, with snapping to the feature

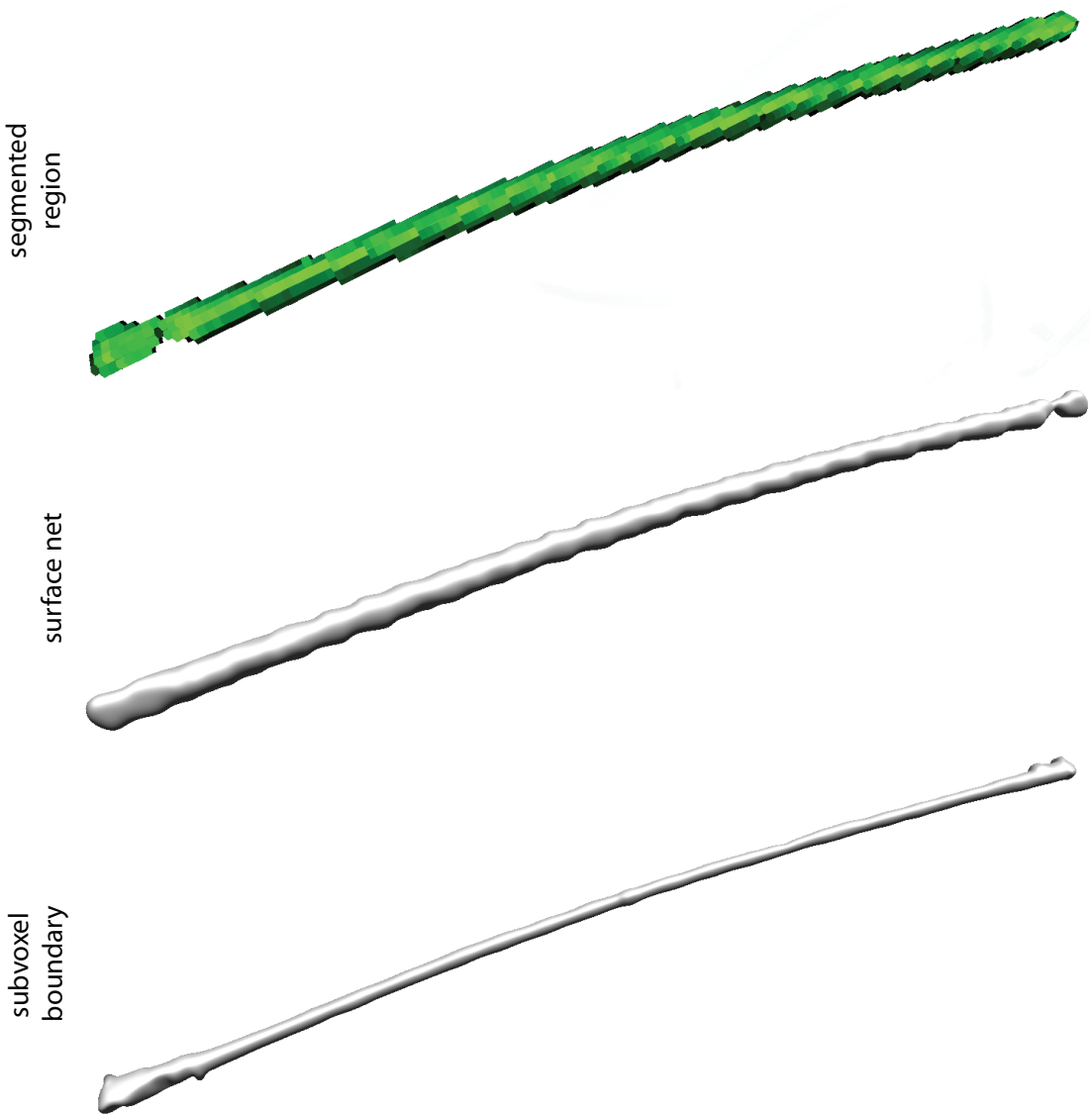
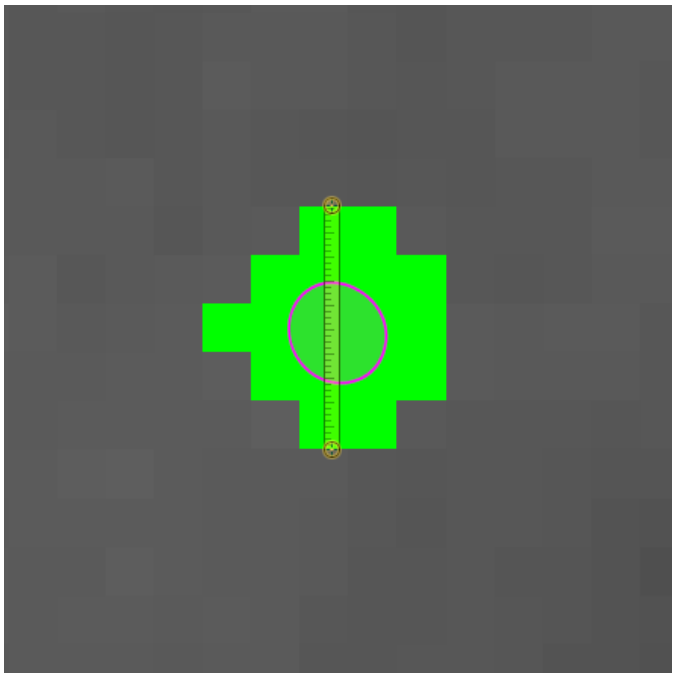
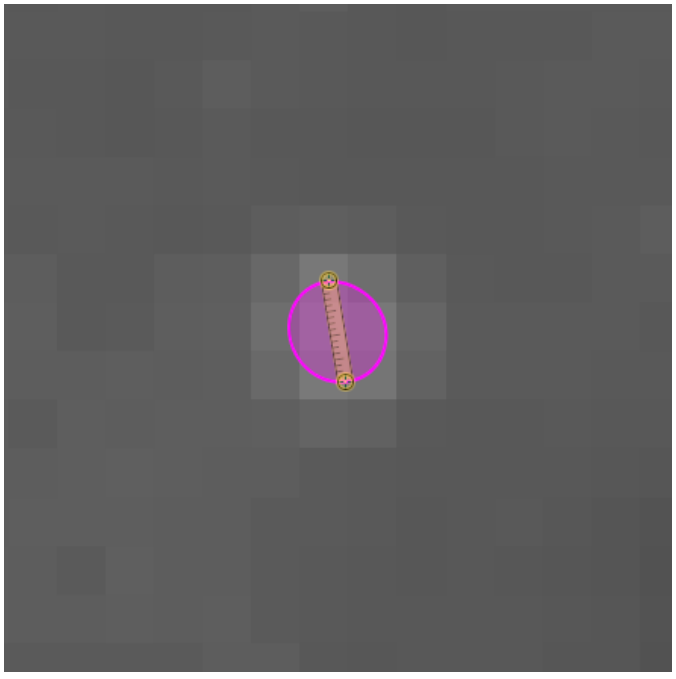


Figure 5.14: Feature #538, an almost straight fibre.



(a)



(b)

Figure 5.15: Diameter measuring on feature #538 (Figure 5.14), (a) based on the segmented region (b) and the subvoxel boundary.

		feature ID		
		#71 (Fig. 5.12)	#140 (Fig. 5.13)	#538 (Fig. 5.14)
segmented region	voxels	2188.00	5645.00	2335.00
	mm^3	26.57	68.57	28.36
surface net	voxels	1482.16	3863.61	1561.53
	mm^3	18.00	46.92	18.96
subvoxel boundary	voxels	647.50	1516.33	604.67
	mm^3	7.86	18.42	7.34
diameter	region	1.25 mm	1.45 mm	1.14 mm
	subvoxel	0.50 mm	0.50 mm	0.49 mm

Table 5.3: Volume and diameter quantification comparison of three steel fibres in the SFRSpC dataset. Volume size: $440 \times 440 \times 705$. Voxel size: $0.23 \times 0.23 \times 0.23$ mm.

voxels (Figure 5.15(a)) or the subvoxel surface respectively (Figure 5.15(b)). This dataset offers an opportunity to quantify the margin of error for the different quantification methods, since the dimensions and volume of its features are actually known beforehand. The volume of a fibre should be $\approx 11.6\ mm^3$. Due to the over–segmentation of the region growing process, its volume and diameter quantification introduces an error of over 100%. The contracting surface net comes closer with an error of $\approx 55\%$. The subvoxel boundary detection on the other hand under–estimates the volume and diameter of a fibre by $\approx 25\%$. Even though this method was unable to reconstruct the boundary with 100% accuracy, its results are still closer to the real object, than the other methods.

5.5 Performance, Problems and Limitations

5.5.1 Computation Time

The total time required for processing the gradient integration in order to find the subvoxel boundary largely depends on the size of the features within the dataset. The RPTS dataset shown in Section 5.2 represents the worst case scenario: a high number of very large features that take up a majority of the volume. On a Core i7-930 CPU with 2.80 GHz, the whole gradient integration process takes ≈ 45 minutes when using only tri–linear interpolation of the data. Using tri–cubic interpolation will have a significant increase in computation time of course. In case of the RPTS dataset, it would take about four times as long. The preliminary segmentation step through region growing already takes up a long time as well, depending on the input parameters. The total time for this step can range between 15 minutes up to an hour in a large dataset. The computation time for the last step, the volume quantification through super–sampling, increases linearly with the size of the data. Sampling the complete RPTS dataset with 20^3 samples per voxel takes ≈ 40 minutes on a GeForce 280 GTX.

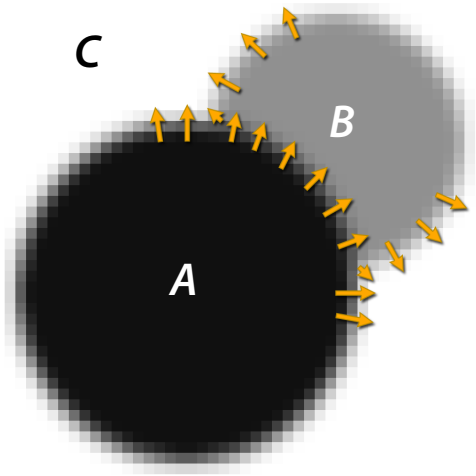


Figure 5.16: n–Material problem: the distances to the boundary for material *B* cannot be reliably classified using the gradient direction, without further adjustments to the algorithm.

5.5.2 Threshold Parameters and Classification

In the current implementation, the minimum gradient magnitude and homogeneity threshold used in the gradient integration process (see page 34) are defined by the user as a percentage from the maximum gradient magnitude in the dataset. However, the maximum gradient magnitude along the boundary of two different materials can vary. If a high minimum gradient threshold percentage is chosen, a subvoxel boundary might not be found between two regions that only have slightly different densities. A solution to this problem might be to do preliminary integrations along the gradient, starting from some boundary voxels, in order to analyze the density and gradient profile of the data locally. If an LH–Histogram was computed already, these LH values could be used for this purpose as well.

The inside–outside classification relies on the gradient direction, as described in the previously mentioned section. The actual assignment for inside and outside to the gradient direction is also done as an additional user input and is applied to all the gradient integration steps of the subvoxel boundary detection process. In reality, features can have lower or higher densities than the material surrounding them, within the same dataset. A solution in this case can also be preliminary integrations in order to find out which direction leads to the inside of a feature.

5.5.3 n–Material Problem

Features that are adjacent to two or more materials (which might also be other features) pose a problem with the subvoxel boundary detection presented in this work. It is generally assumed, that all the features of interest are surrounded by either a higher or lower density material. The two–channel distance field can only store one distance to one feature in one voxel. Therefore it cannot represent two different features, that are directly adjacent to each other, since that would require different distance values for each feature.

Another problem with adjacent, different materials is the classification of the distance value to the subvoxel boundary, which relies on the gradient direction. Figure 5.16 shows an example: Materials A and B are surrounded by material C. The distances to the boundary for material A can be classified by the gradient direction, however for material B it would lead to a false classification at the boundary between materials A and B, if the classification is determined by the gradient direction from B to C.

A solution for this problem, as well as the problems discussed in Section 5.5.2 before, could be to always perform a full integration until a homogeneous area is reached in both directions. The sign classification would then be determined by which integration direction reaches the homogeneous area inside the feature (defined by the binary segmentation).

5.5.4 Reliance on Region Growing

Since the subvoxel boundary detection uses the volume segmentation provided by the region growing process as a starting point, a good region growing result is necessary. Figure 5.5(c) shows a result from the subvoxel boundary detection, where some features are not present in the distance field. This is because no segmented region existed there in the first place.

There are also cases where single voxels or even groups of voxels inside a feature are not added to the surrounding binary segmented region, due to high noise for example. This poses a problem, since these voxels are also not initialized as an inside–voxel in the distance field then. Start positions for the gradient integration will be created there, since these voxels are technically boundary voxels now. However, these will likely lead to no result, because they are inside the homogeneous area of the feature and thus the initial value will be retained, which creates holes in the distance field within the feature.

Another problem is under–segmentation and over–segmentation. If the segmented region does not reach a point near the correct boundary or goes too far beyond, the subvoxel boundary cannot be accurately reconstructed anymore, using the method of this thesis. Additionally, in the latter case, voxels far outside the feature will be classified as inside initially. This can prevent a consistent boundary definition in the distance field, when no conclusive result is found for such voxels.

CHAPTER 6

Summary

This thesis presents a method for improving quantification results of automatically segmented regions in industrial CT data. By combining different concepts of the visual computing field, additional information from volumes is extracted on a subvoxel level, essentially trying to reconstruct the boundaries between materials in the original, real object. There are some difficulties, which will impact the result of these quantifications. Due to the limited resolution and artifacts that occur during the acquisition process of the data, very small details will inadvertently get lost and cannot be reconstructed by this method. However, within certain limitations, this method provides a relatively robust and more importantly automatic method of determining the original subvoxel boundaries of detected features in a dataset. Also presented are methods for quantifying these results, by calculating the actual subvoxel volume for each segmented region of the data and by providing simple measurement tools, which are also guided by these results. Quantification results are compared with the original segmentations and a simple smoothing technique for these segmented regions.

Future Work

There is still potential for improvement in the method used for finding subvoxel boundaries in this thesis, as well as some possibilities for alternative ways for representation and quantification of these boundaries. Some possible improvements were already mentioned in Section 5.5.

Boundary detection Alternative ways of determining the subvoxel boundary can still be explored. Three methods for calculating the second directional derivative were presented in Section 3.2.1, a fourth method would be to simply find the maximum of the first directional derivative, the gradient magnitude. These could be compared and analyzed further.

Local thresholds As mentioned before, the subvoxel boundary detection needs certain thresholds of the first directional derivative, in order to discard false results. This threshold is currently applied globally, though results can be improved by analyzing the gradient magnitude profile locally first.

Distance classification The inside–outside classification is a crucial part of the methodology, but could fail completely under certain circumstances, like the n–material problem described in Section 5.5.3. Prior analysis of a region could improve this.

Multi–distance field An inherent limitation of the method used in this thesis, is the inability to represent different features in the distance field, that might be very close together and thus more than one distance or at least more than one classification would be needed for one element of the distance field.

Mesh–based representation Another idea is to use a mesh based representation instead of the distance field, where vertices will be moved along the gradient direction to the subvoxel boundary of the region and additional vertices will be spawned adaptively, to sample the subvoxel boundary as accurately as possible. Such a technique was also used by Heinzl et al. [22].

While the distance field representation has the advantage of implicitly describing a surface in high detail when using advanced reconstruction techniques, its usage in this thesis also limits some possibilities of the subvoxel boundary detection. The definition and actual location of the boundary might be completely different from the result of the preliminary segmentation, but the detection process is limited by this segmentation result. Using the mesh based idea, the subvoxel boundary could be tracked and represented beyond that limitation.

Bibliography

[1] R. Adams and L. Bischof. Seeded region growing. *IEEE Trans. Pattern Anal. Mach. Intell.*, 16:641–647, June 1994. ISSN 0162-8828. doi: http://dx.doi.org/10.1109/34.295913. URL http://dx.doi.org/10.1109/34.295913.

[2] J. C. Anderson, C. Garth, M. A. Duchaineau, and K. Joy. Discrete multi-material interface reconstruction for volume fraction data. *Computer Graphics Forum (Proc. of Eurographics/IEEE-VGTC Symposium on Visualization 2008)*, 27(3), 2008. URL http://graphics.cs.ucdavis.edu/~cobalt.

[3] J. C. Anderson, C. Garth, M. A. Duchaineau, and K. I. Joy. Smooth, volume-accurate material interface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 16:802–814, September 2010. ISSN 1077-2626. doi: http://dx.doi.org/10.1109/TVCG.2010.17. URL http://dx.doi.org/10.1109/TVCG.2010.17.

[4] R. Bade, J. Haase, and B. Preim. Comparison of fundamental mesh smoothing algorithms for medical surface models. In *SimVis'06*, pages 289–304, 2006.

[5] R. Bade, O. Konrad, and B. Preim. Reducing Artifacts in Surface Meshes Extracted from BinaryVolumes. *Journal of WSCG*, 15(1-3):67–74, 2007.

[6] J. A. Bærentzen and H. Aanæs. Generating signed distance fields from triangle meshes. Technical report, Technical University of Denmark, Department of Informatics and Mathematical Modeling, Image Analysis and Computer Graphics, 2002. URL http://www2.imm.dtu.dk/pubdb/p.php?1833.

[7] J. A. Baerentzen and H. Aanaes. Signed distance computation using the angle weighted pseudonormal. *IEEE Transactions on Visualization and Computer Graphics*, 11:243–253, May 2005. ISSN 1077-2626. doi: http://dx.doi.org/10.1109/TVCG.2005.49. URL http://dx.doi.org/10.1109/TVCG.2005.49.

[8] K. Bjorke. High-quality filtering. *GPU Gems: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, pages 391–415, 2004.

[9] C. Blasch. Orientierung von Stahlfasern im Stahlfaserspritzbeton. Bachelor thesis, University of Leoben, 2007.

[10] J. C. Butcher. *Numerical methods for ordinary differential equations*. John Wiley & Sons, 2003.

[11] J. P. Chiverton and C. J. P. Chiverton. *Probabilistic Partial Volume Modelling of Biomedical Tomographic Image Data*. PhD thesis, Centre for Vision, Speech and Signal Processing, School of Electronics and Physical Sciences, University of Surrey, 2006.

[12] T. A. Chowdhury, O. Ghita, and P. F. Whelan. Evaluation of 3d gradient filters for estimation of the surface orientation in CTC. In *Proceedings of the 10th International Machine Vision and Image Processing Conference*, IMVIP 2006, Dublin, Ireland, 30th August - 1st September 2006.

[13] D. Cohen-Or, A. Solomovic, and D. Levin. Three-dimensional distance field metamorphosis. *ACM Trans. Graph.*, 17:116–141, April 1998. ISSN 0730-0301. doi: http://doi.acm.org/10.1145/274363.274366. URL http://doi.acm.org/10.1145/274363.274366.

[14] S. M. December and S. Mauch. A fast algorithm for computing the closest point and distance transform. Technical Report 077, California Institute of Technology, Accelerated Strategic Computing Initiative, 2000.

[15] S. F. Frisken and R. N. Perry. Designing with distance fields. In *ACM SIGGRAPH 2006 Courses*, SIGGRAPH '06, pages 60–66, New York, NY, USA, 2006. ACM. ISBN 1-59593-364-6. doi: http://doi.acm.org/10.1145/1185657.1185675. URL http://doi.acm.org/10.1145/1185657.1185675.

[16] S. F. Frisken, R. N. Perry, A. P. Rockwood, and T. R. Jones. Adaptively sampled distance fields: a general representation of shape for computer graphics. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 249–254, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co. ISBN 1-58113-208-5. doi: http://dx.doi.org/10.1145/344779.344899. URL http://dx.doi.org/10.1145/344779.344899.

[17] S. F. F. Gibson. Constrained elastic surface nets: Generating smooth surfaces from binary segmented data. In *Proceedings of the First International Conference on Medical Image Computing and Computer-Assisted Intervention*, MICCAI '98, pages 888–898, London, UK, 1998. Springer-Verlag. ISBN 3-540-65136-5. URL http://dl.acm.org/citation.cfm?id=646921.709482.

[18] C. Green. Improved alpha-tested magnification for vector textures and special effects. In *ACM SIGGRAPH 2007 courses*, SIGGRAPH '07, pages 9–18, New York, NY, USA, 2007. ACM. doi: http://doi.acm.org/10.1145/1281500.1281665. URL http://doi.acm.org/10.1145/1281500.1281665.

[19] M. Hadwiger, J. M. Kniss, C. Rezk-Salama, D. Weiskopf, and K. Engel. *Real-time Volume Graphics*. A. K. Peters, Ltd., Natick, MA, USA, 2006. ISBN 1568812663.

[20] M. Hadwiger, F. Laura, C. Rezk-Salama, T. Höllt, G. Geier, and T. Pabel. Interactive volume exploration for feature detection and quantification in industrial ct data. *IEEE Transactions on Visualization and Computer Graphics*, 14:1507–1514, November 2008. ISSN 1077-2626. doi: 10.1109/TVCG.2008.147. URL http://dl.acm.org/citation.cfm?id=1477066.1477455.

[21] H. C. Hege, T. Hollerer, and D. Stalling. Volume rendering mathematical models and algorithmic aspects. Technical Report TR 93-7, Konrad-Zuse-Zentrum Berlin, 1993.

[22] C. Heinzl, J. Kastner, and E. Gröller. Surface extraction from multi-material components for metrology using dual energy ct. *IEEE Transactions on Visualization and Computer Graphics*, 13:1520–1527, November 2007. ISSN 1077-2626. doi: http://dx.doi.org/10.1109/TVCG.2007.70598. URL http://dx.doi.org/10.1109/TVCG.2007.70598.

[23] V. Holmstrom. Modified surfacenets: Smoothing a marching cubes mesh. 1998. College of William & Mary.

[24] R. Huang, K.-L. Ma, P. McCormick, and W. Ward. Visualizing industrial ct volume data for nondestructive testing applications. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, VIS '03, pages 72–, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-2030-8. doi: http://dx.doi.org/10.1109/VISUAL.2003.1250418. URL http://dx.doi.org/10.1109/VISUAL.2003.1250418.

[25] Jesse Garant & Associates. Types of industrial computed tomography scanners, . URL http://www.jgarantmc.com/blog/industrial-ct-scanning-services/types-of-industrial-computed-tomography-scanners/. Accessed: 2011-10-17.

[26] M. W. Jones, J. A. Baerentzen, and M. Sramek. 3d distance fields: A survey of techniques and applications. *IEEE Transactions on Visualization and Computer Graphics*, 12:581–599, July 2006. ISSN 1077-2626. doi: 10.1109/TVCG.2006.56. URL http://portal.acm.org/citation.cfm?id=1137246.1137512.

[27] R. A. Ketcham and W. D. Carlson. Acquisition, optimization and interpretation of x-ray computed tomographic imagery: applications to the geosciences. *Comput. Geosci.*, 27:381–400, May 2001. ISSN 0098-3004. doi: 10.1016/S0098-3004(00)00116-3. URL http://portal.acm.org/citation.cfm?id=374764.374770.

[28] G. Kindlmann and J. W. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *Proceedings of the 1998 IEEE symposium on Volume visualization*, VVS '98, pages 79–86, New York, NY, USA, 1998. ACM. ISBN 1-58113-105-4. doi: http://doi.acm.org/10.1145/288126.288167. URL http://doi.acm.org/10.1145/288126.288167.

[29] J. Kniss, G. Kindlmann, and C. Hansen. Multidimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 8:270–285, July 2002. ISSN 1077-2626. doi: http://dx.doi.org/10.1109/TVCG.2002.1021579. URL http://dx.doi.org/10.1109/TVCG.2002.1021579.

[30] J. Krüger and R. Westermann. Acceleration techniques for gpu-based volume rendering. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, VIS '03, page 38, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-2030-8. doi: http://dx.doi.org/10.1109/VIS.2003.10001. URL http://dx.doi.org/10.1109/VIS.2003.10001.

[31] L. Kuptsov. *Encyclopaedia of Mathematics*. Springer, 2001.

[32] W. La-Orchan. *Quantification of the reduced pressure test*. McGill University, 1994. URL http://books.google.com/books?id=ulweQwAACAAJ.

[33] P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH '94, pages 451–458, New York, NY, USA, 1994. ACM. ISBN 0-89791-667-0. doi: http://doi.acm.org/10.1145/192161.192283. URL http://doi.acm.org/10.1145/192161.192283.

[34] D. H. Laidlaw. Geometric model extraction from magnetic resonance volume data. Technical report, California Institute of Technology, Pasadena, CA, USA, 1995.

[35] D. H. Laidlaw, K. W. Fleischer, and A. H. Barr. *Partial-Volume Bayesian Classification Using Voxel Histograms*. Department of Computer Science, Brown University, 2001.

[36] M. Levoy. Display of surfaces from volume data. *IEEE Comput. Graph. Appl.*, 8:29–37, May 1988. ISSN 0272-1716. doi: 10.1109/38.511. URL http://portal.acm.org/citation.cfm?id=44650.44652.

[37] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '87, pages 163–169, New York, NY, USA, 1987. ACM. ISBN 0-89791-227-6. doi: http://doi.acm.org/10.1145/37401.37422. URL http://doi.acm.org/10.1145/37401.37422.

[38] N. Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1:99–108, June 1995. ISSN 1077-2626. doi: 10.1109/2945.468400. URL http://dl.acm.org/citation.cfm?id=614258.614298.

[39] D. P. Mitchell and A. N. Netravali. Reconstruction filters in computer-graphics. *SIGGRAPH Comput. Graph.*, 22:221–228, June 1988. ISSN 0097-8930. doi: http://doi.acm.org/10.1145/378456.378514. URL http://doi.acm.org/10.1145/378456.378514.

[40] T. Möller, R. Machiraju, K. Mueller, and R. Yagel. A comparison of normal estimation schemes. In *Proceedings of the 8th conference on Visualization '97*, VIS '97, pages 19–ff., Los Alamitos, CA, USA, 1997. IEEE Computer Society Press. ISBN 1-58113-011-2. URL http://dl.acm.org/citation.cfm?id=266989.267004.

[41] N. Nacereddine, L. Hamami, and D. Ziou. Thresholding techniques and their performance evaluation for weld defect detection in radiographic testing. *MG&V*, 15:557–566, January 2006. ISSN 1230-0535. URL http://dl.acm.org/citation.cfm?id=1375858.1375890.

[42] A. Neubauer. *Virtual Endoscopy for Preoperative Planning and Training of Endonasal Transsphenoidal Pituitary Surgery*. PhD thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, June 2005. URL http://www.cg.tuwien.ac.at/research/publications/2005/neubauer-2005-vir/.

[43] L. Neumann, B. Csébfalvi, A. König, and E. Gröller. Gradient estimation in volume data using 4d linear regression. Technical Report TR-186-2-00-03, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, Feb. 2000. URL http://www.cg.tuwien.ac.at/research/publications/2000/Csebfalvi-2000-GELR/.

[44] Österreichisches Gießerei Institut. Pilot foundry, . URL http://www.ogi.at/en/labors/versuchsgiesserei.php. Accessed: 2011-10-17.

[45] B. Preim, C. Tietjen, W. Spindler, and H. O. Peitgen. Integration of measurement tools in medical 3d visualizations. In *Proceedings of the 13th conference on Visualization '02*, VIS '02, pages 21–28, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7803-7498-3. URL http://portal.acm.org/citation.cfm?id=602099.602101.

[46] S. Raman and R. Wenger. Quality isosurface mesh generation using an extended marching cubes lookup table. *Comput. Graph. Forum*, 27(3):791–798, 2008. URL http://dblp.uni-trier.de/db/journals/cgf/cgf27.html#RamanW08.

[47] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl. Interactive volume rendering on standard PC graphics hardware using multi-textures and multi-stage rasterization. In *HWWS '00: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 109–118, New York, NY, USA, 2000. ACM Press. ISBN 1581132573. doi: 10.1145/346876.348238. URL http://dx.doi.org/10.1145/346876.348238.

[48] P. Rideout. Sprites and text, . URL http://ofps.oreilly.com/titles/9780596804824/index.html. Accessed: 2011-10-17.

[49] S. Roettger, S. Guthe, D. Weiskopf, T. Ertl, and W. Strasser. Smart hardware-accelerated volume rendering. In *Proceedings of the symposium on Data visualisation 2003*, VISSYM

'03, pages 231–238, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association. ISBN 1-58113-698-6. URL http://portal.acm.org/citation.cfm?id=769922.769948.

[50] A. Rosenfeld and J. L. Pfaltz. Sequential operations in digital picture processing. *J. ACM*, 13:471–494, October 1966. ISSN 0004-5411. doi: http://doi.acm.org/10.1145/321356.321357. URL http://doi.acm.org/10.1145/321356.321357.

[51] H. Scharsach, M. Hadwiger, A. Neubauer, S. Wolfsberger, and K. Bühler. Perspective isosurface and direct volume rendering for virtual endoscopy applications. In B. S. Santos, T. Ertl, and K. I. Joy, editors, *EuroVis*, pages 315–322. Eurographics Association, 2006. ISBN 3-905673-31-2. URL http://dblp.uni-trier.de/db/conf/vissym/eurovis2006.html#ScharsachHNWB06.

[52] J. P. Schulze, R. Niemeier, and U. Lang. The perspective shear-warp algorithm in a virtual environment. In *Proceedings of the 12th conference on Visualization '01*, VIS '01, pages 207–214, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7803-7200-X. URL http://portal.acm.org/citation.cfm?id=601671.601703.

[53] M. Sezgin and B. Sankur. Survey over image thresholding techniques and quantitative performance evaluation. *J. Electronic Imaging*, pages 146–168, 2004.

[54] C. Sigg and M. Hadwiger. Fast third-order texture filtering. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, pages 313–329, 2005.

[55] S. W. Smith. *The scientist and engineer's guide to digital signal processing*. California Technical Publishing, San Diego, CA, USA, 1997. ISBN 0-9660176-3-3.

[56] T. Theußl, H. Hauser, and E. Gröller. Mastering windows: improving reconstruction. In *Proceedings of the 2000 IEEE symposium on Volume visualization*, VVS '00, pages 101–108, New York, NY, USA, 2000. ACM. ISBN 1-58113-308-1. doi: http://doi.acm.org/10.1145/353888.353906. URL http://doi.acm.org/10.1145/353888.353906.

[57] P. Thévenaz and M. Unser. Precision isosurface rendering of 3d image data. *IEEE Transactions on Image Processing*, 12:764–775, 2003. doi: 10.1109/TIP.2003.814240.

[58] P. Šereda, A. Vilanova Bartroli, I. W. O. Serlie, and F. A. Gerritsen. Visualization of Boundaries in Volumetric Data Sets Using LH Histograms. *IEEE Transactions on Visualization and Computer Graphics*, 12:208–218, March 2006. ISSN 1077-2626. doi: http://dx.doi.org/10.1109/TVCG.2006.39. URL http://dx.doi.org/10.1109/TVCG.2006.39.

[59] L. Westover. Footprint evaluation for volume rendering. *SIGGRAPH Comput. Graph.*, 24:367–376, September 1990. ISSN 0097-8930. doi: http://doi.acm.org/10.1145/97880.97919. URL http://doi.acm.org/10.1145/97880.97919.

[60] Wikipedia. Industrial CT scanning - cone beam rotation, . URL http://en.wikipedia.org/wiki/File:Ct_scan_cone_beam.png. Accessed: 2011-10-17.

[61] Wikipedia. Jolted steel fibre. URL http://de.wikipedia.org/w/index.php?title=Datei:Gestauchte_Form.svg&filetimestamp=20110929150501. Accessed: 2011-10-17.

[62] F. Y. Wu. The potts model. *Reviews of Modern Physics*, 54(1):235–268, Jan. 1982. doi: 10.1103/RevModPhys.54.235. URL http://dx.doi.org/10.1103/RevModPhys.54.235.

[63] Y. Zhou and A. W. Toga. Efficient skeletonization of volumetric objects. *IEEE Transactions on Visualization and Computer Graphics*, 5:196–209, July 1999. ISSN 1077-2626. doi: 10.1109/2945.795212. URL http://portal.acm.org/citation.cfm?id=614275.614432.

[64] Y. Zhou, A. E. Kaufman, and A. W. Toga. Three-dimensional skeleton and centerline generation based on an approximate minimum distance field. *The Visual Computer*, pages 303–314, 1998.