Interactive Volume Exploration for Feature Detection and Quantification in Industrial CT Data

Markus Hadwiger, Laura Fritz, Christof Rezk-Salama, Thomas Höllt, Georg Geier, and Thomas Pabel

Abstract— This paper presents a novel method for interactive exploration of industrial CT volumes such as cast metal parts, with the goal of interactively detecting, classifying, and quantifying features using a visualization-driven approach. The standard approach for defect detection builds on region growing, which requires manually tuning parameters such as target ranges for density and size, variance, as well as the specification of seed points. If the results are not satisfactory, region growing must be performed again with different parameters. In contrast, our method allows interactive exploration of the parameter space, completely separated from region growing in an unattended pre-processing stage. The pre-computed *feature volume* tracks a *feature size curve* for each voxel over *time*, which is identified with the main region growing parameter such as variance. A novel 3D transfer function domain over *(density, feature_size, time)* allows for interactive exploration of feature classes. Features and feature size curves can also be explored individually, which helps with transfer function specification and allows coloring individual features and disabling features resulting from CT artifacts. Based on the classification obtained through exploration, the classified features can be quantified immediately.

Index Terms—Non-Destructive Testing, Multi-Dimensional Transfer Functions, Region Growing, Volume Rendering.

1 Introduction

Non-destructive testing (NDT) is a scientific discipline which examines the internal structures of industrial components such as machine parts, pipes, or ropes without destroying them. It is an essential tool in construction engineering and manufacturing, especially in the automotive and aviation industry. In cast metal parts, for example, the processes during solidification may cause shrinkage cavities, pores, cracks, or inhomogeneities to appear inside the structure, which are not visible from the outside. NDT allows the assessment of such material defects which arise throughout the manufacturing process, as well as during use if the component is exposed to mechanical loads or corrosion. Furthermore, NDT is nowadays not only used for inspecting metal parts but for a variety of different materials such as plastics, wood, or concrete, as well as minerals in general. In recent years, 3D Computed Tomography (CT) has become common in NDT, which has created powerful new possibilities, but also new challenges for the inspection and testing process. Industrial CT volumes are generally quite large, with voxels commonly stored with 16 bits of precision, which leads to several hundred MB to one or more GB of raw data per scan. Real-time volume rendering has become an essential tool for visualizing these volumes, usually using bricking strategies [5] to cope with the large data sizes. However, for NDT practitioners visualization is just one part of the workflow, which includes a variety of processing tasks such as defect detection and quantification, computing statistical measures and properties such as material porosity, performing accurate measurements and comparisons, and many more.

The goal of our work is to help bridge the gap between the visualization of features and the quantification of defects. Feature detection is usually performed via some type of segmentation, which most commonly builds on region growing and filtering operations such as morphological operators. Segmentation results in one or several static segmentation masks, which can be visualized as part of the 3D volume and also form the basis of quantification. The segmentation, however, cannot be modified without re-computation. This decouples the detection of features from visualization and prevents working in a fully interactive manner. Most of all, it hampers interactively exploring the volume without already knowing beforehand what features are contained in it. Whenever the segmentation results for specified parameters are not satisfactory, the user has to modify the parameters and the entire segmentation has to be computed all over again. This is often time-consuming and tedious.

Unlike this standard approach, we propose a *visualization-driven* method for feature detection that allows features in the volume to be explored interactively without re-computing the segmentation. The basis for this is an unattended pre-computation stage that computes a *feature volume* and some additional data structures, which contain the result of feature detection over *parameter domains* instead of fixed parameters. This pre-computation has to be performed only once for a given data set and forms the basis of interactively exploring all contained features. In contrast to detection of a single type of features, we allow the user to explore all feature classes and decide interactively which classes of features are of interest, instead of specifying this information beforehand. This is particularly useful in the context of compound parts or complex materials such as minerals.

In a traditional workflow for detecting defects in cast metal parts, the parameters for the segmentation need to be specified such that the density of defects is below a certain threshold (assuming that air or gas comprises the interior of defects), their sizes are larger than a given minimum (very small defects are noise), and smaller than a given maximum (very big defects are not defects but, e.g., actual holes). Moreover, further parameters must be set for the region growing process, for example a maximum density variance in the region, or maximum standard deviation from the neighborhood of a seed voxel. The system might also require the user to manually specify seed voxels or set parameters for automatic seed determination. In contrast, our system computes and records the result of region growing for the entire density domain, all different sizes of features, and the entire domain of the most important region growing parameter (given a specific region growing algorithm) such as maximum variance. For generality, we refer to this parameter as the "time" parameter t throughout the paper. Together, these three 1D parameter ranges comprise the 3D *(density, feature_size, time)* domain, which is explored by the user via 3D transfer functions (TFs). In order to make TF specification tractable, a 2.5D metaphor is employed, which still provides the necessary flexibility.

2 Related Work

Region growing is a fundamental image processing technique for segmentation [18] based on a specified homogeneity criterion. It belongs to the class of non-uniform problems, whose run-time complexity is

- *Markus Hadwiger, Laura Fritz, Thomas Höllt are with the VRVis Research Center, Austria, E-mail: {msh|laura}@vrvis.at, thoellt@uni-koblenz.de.*
- *Christof Rezk-Salama is with the Computer Graphics Group, University of Siegen, Germany, E-mail: rezk@fb12.uni-siegen.de.*
- *Georg Geier and Thomas Pabel are with the Austrian Foundry Research Institute, Austria, E-mail: {georg.geier|thomas.pabel}@ogi.at.*

Manuscript received 31 March 2008; accepted 1 August 2008; posted online 19 October 2008; mailed on 13 October 2008.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

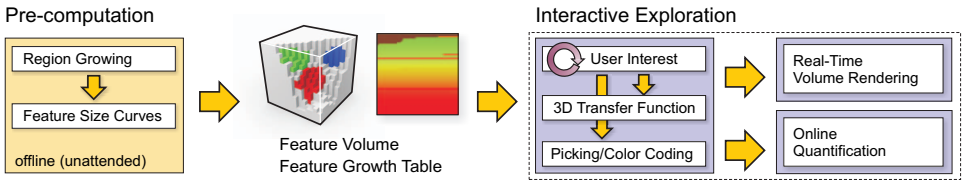

Fig. 1. Overview of our pipeline. The pre-computation stage computes feature size curves via multi-pass region growing, which are stored split up into a 3D feature volume and a 2D feature growth table. These are the basis for the subsequent interactive exploration stage.

strongly data dependent and cannot be determined at compilation time. Simple implementations are based on aggregation starting from a specified seed point in a recursive way. More efficient implementations use a split-and-merge strategy [8], and parallel implementations have been developed [4, 11]. The seed points and the homogeneity metric are essential for region growing techniques. Many research groups have suggested methods to simplify and automatize their specification. In [1], the homogeneity criterion is omitted by simultaneous evaluation of several different seed points. The volume seedlings approach [3] represents an interactive technique for specifying seed points to select regions of interest. However, since this approach is working in screen space, it is restricted to a static viewpoint. Other techniques derive the homogeneity criterion from statistical information about the local neighborhood of the seed point, mainly mean value and variance. In 3D, such techniques are closely related to automatic iso-surface extraction. Tenginakai et al. [16] propose a method for detection of salient iso-surfaces based on higher-order statistical moments. Techniques such as contour-trees [17] and -spectrum [2] evaluate information about topology, area and enclosed volume of iso-surfaces. This information is then used for feature classification.

For the visualization of 3D scalar fields, powerful real-time volume rendering techniques are available [6]. In recent years, usability aspects have become more and more prominent in visualization systems. In their seminal paper, Kindlmann and Durkin describe a semi-automatic TF design [12]. Although their TF was still one-dimensional, the gradient magnitude and the second-order derivative of the scalar field were taken into account. True multi-dimensional TFs were introduced by Kniss et al. [13]. Here, the derivatives were pre-computed and a 3D TF was applied for classification. The authors also proposed a user-interface based on interaction in both the spatial domain and the feature space of the TF (dual domain interaction). The original idea of tracing path lines along the gradient field described in [12], was expanded by Sereda et al. [14]. They employ a LH (low-high) histogram for selecting regions of interest in feature space. Each voxel with a gradient magnitude larger than a small threshold is considered a boundary voxel and a short path line is traced along the gradient field in order to determine two tissue types at the boundary.

Huang and Ma [9] integrate the region growing technique into volume visualization systems. Besides full segmentation, their technique may perform region growing on a partial data range in order to define a 2D TF for volume rendering. Such a visualization, however, will not be exact compared to the full segmentation. Although their visualization is fast and effective, modifying the seed points at run-time will also require re-computation. Huang et al. also demonstrate an application of their region growing technique for non-destructive testing of CT data [10], which is effective, but underlies the same limitations for interactive exploration. Like the approach by Huang and Ma [10], our visualization technique is based on multi-dimensional TFs. The feature volume we use, however, is different. Unlike their approach, our technique allows us to interactively explore the volume by selecting feature size, density, and the main region growing parameter in real-time. Region growing is performed only as a pre-processing step.

3 Pipeline Overview

An overview of our pipeline for exploration and quantification of features is illustrated in Figure 1. As a pre-requisite, for a given CT volume additional information must be pre-computed (Section 4), which constitutes the basis of interactive feature exploration. We employ a multi-pass region growing approach (Section 4.4) that conceptually computes a *feature size curve* over "time" t (which corresponds to the main region growing parameter) for each voxel in the volume (Section 4.1). For memory efficiency, these curves are stored split up into a 3D *feature volume* (Section 4.2), and a corresponding 2D *feature growth table* (Section 4.3).

When the feature volume and growth table are available, the data set can be explored interactively for features of interest in the *exploration* stage (Section 5). Feature exploration builds on the specification of a 3D transfer function (TF) in the *(density, feature_size, time)* domain (Section 5.1), which is constituted by the CT density volume, the feature volume, and the feature growth table. TF specification is not only the means by which the user determines the visualization, but also how features to be quantified are selected. Features can also be explored individually using a graphical feature view or picking in orthogonal slice views (Section 5.2), which can also be used to remove specific features from rendering and quantification that are artifacts from the CT acquisition process. During exploration, the current feature classification is displayed using real-time volume rendering (Section 5.3).

From the feature classification specified by the user during the interactive exploration phase, the *quantification* stage (Section 6) automatically computes statistical measures such as feature count, volume, and surface area for features that have been selected in the exploration stage. That is, quantification is performed in a visualization-driven manner, where everything that is selected for feature visualization is included in the quantification. Performing quantification only for the feature classes found to be of interest during exploration empowers the domain expert to interactively control the final result. Both feature exploration and quantification can be performed as often as desired without requiring additional pre-computation.

4 Pre-Computation

Although exploration is conceptually the most important part of our pipeline, the basis for interactivity during exploration is a complex pre-computation stage, whose components are described in this section. However, no user input is required for this stage, and thus it is technically complex and important but decoupled from the exploration. The main goal for pre-computing additional information is to enable exploration of different classes of features with different parameters in such a way that, e.g., the main parameter used to steer region growing (e.g., maximum variance) can be changed interactively after actual region growing has been performed. In order to allow this, we perform region growing in multiple passes and track the progress for each voxel, which is recorded in *feature size curves*.

4.1 Feature Size Curves

In order to allow interactive exploration of region growing with different parameter settings, such as different variance thresholds, which would not be possible to change interactively, the result of region growing is tracked and recorded along the parameter axis in the pre-computation stage. That is, instead of using a single parameter value, we track features over an entire *parameter range*. In order to make the following description more general, we denote this parameter as "time" t, which is stepped from a start time t_0 (e.g., minimum interesting variance) to a maximum time t_n (e.g., maximum interesting variance) in a specified number of steps b. That is, the time (parameter) axis is sampled into b bins, e.g., $b = 16$, which allows the resulting curves to be stored in arrays with b entries. For each voxel, the size of the feature (region) it belongs to is recorded, resulting in a *feature size curve* for each voxel $\mathbf{x}$ along the time axis t: $f_s(\mathbf{x},t)$.

Figure 2 illustrates the feature size curves of four different example voxels. Voxel 0 (red) is the seed voxel of a feature that starts at time t_0 with a size of 30 voxels and grows in size several times as t increases. Voxel 3 (purple) is another voxel in this feature, but only becomes a part of it at time t_1. Voxel 2 (green) belongs to another feature, but merges with the feature containing voxels 0 and 3 at time t_2. Voxel 1 (blue) belongs to a feature with a size of 15 voxels, which stays at this size over time, i.e., does not grow any further.

Storing feature size curves: The major observation for storing feature size curves $f_s(\mathbf{x},t)$ with a manageable amount of memory is that all voxels in a feature exhibit very similar curves. In the beginning (time t_0), most voxels do not belong to any feature, i.e., $f_s(\mathbf{x},t_0) = 0$. Seed voxels that start a new feature at time t_i record the entire growth curve of this feature. In each pass, additional voxels may become a part of this feature, and when a voxel does so at time t_j, it from then on shares the feature size curve with the curve of the original seed voxel. Before a voxel becomes part of a feature, its "voxel-local" feature size is zero. That is:

$$f_s(\mathbf{x},t) = \begin{cases} 0 & t < t_j \\ f_s(\mathbf{x}_s,t) & t \geq t_j \end{cases} \quad (1)$$

for a voxel $\mathbf{x}$ that at time t_j becomes a part of a feature whose original seed voxel is $\mathbf{x}_s$. This fact makes it possible to store only a single feature size curve *per feature* in full. However, for each voxel, the feature ID that it will become a part of at time t_j must be stored, as well as storing t_j itself. This *per voxel* information is stored in a 3D *feature volume* (Section 4.2), whereas the curves themselves are stored in a 2D *feature growth table* (Section 4.3). Figure 3 illustrates the relationship between the feature volume, the feature growth table, and feature size curves, which is described in detail below.

4.2 Feature Volume

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

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

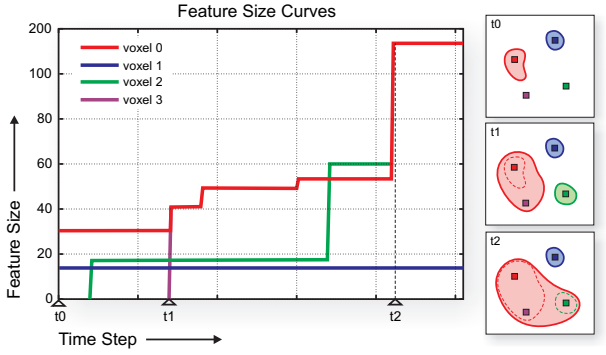

Fig. 2. Feature size curves of four example voxels. Seed voxels belong to a feature from the time where it is created (voxels 0, 1, 2), whereas other voxels may join a feature at a later time (voxel 3; t_1). Features may merge over time (t_2), which implies that the feature size curves of all contained voxels are the same after the merge (voxels 0, 2, and 3).

4.3 Feature Growth Table

As described in the previous sections, the feature volume itself does not store actual feature size curves. Instead, we store one representative curve for each feature, which is the feature size curve of the feature's seed voxel $\mathbf{x}_s$, in a 2D *feature growth table*. The feature growth table contains one row per feature and b columns, where b is the number of bins for sampling the parameter t. Each entry in this table is a $(f_s(\mathbf{x}_s,t_i), ID(t_i))$ pair, which in each row collectively represent the sampled growth curve of the feature over time t: $f_s(\mathbf{x}_s,t)$, as well as mapping time to *current feature ID*: $ID(t)$. The latter is necessary in order to be able to handle the merging of features over time, where feature birth ID and current feature ID are not necessarily the same because the ID can change when a feature merges into another one. Handling the merging of features is described below. The feature size curve for a given voxel $\mathbf{x}$ can be reconstructed for any time t_i by using Equation 1 with $t_j = t_{birth}(\mathbf{x})$, and indexing the feature growth table in row $ID_{birth}(\mathbf{x})$ and the column corresponding to the bin of t_i, to obtain $f_s(\mathbf{x}_s,t_i)$ when $t_i \geq t_j$. This is a very simple and efficient scheme, which can easily be evaluated in a GPU fragment shader during ray-casting, illustrated by the pseudo code in Section 5.3.

In principle, the feature growth table is stored in a 16-bit two-channel 2D texture, e.g., a Luminance-Alpha texture in OpenGL. However, since this texture has to contain one row per feature and there can be thousands of features, hardware texture dimension constraints often make it impossible to use a single 2D texture for this purpose. If the hardware supports 2D texture arrays (e.g., NVIDIA GeForce 8 or higher), we split up the feature growth table into a texture array with m rows and $\lceil n/m \rceil$ layers, where m is the maximum allowed dimension of a 2D texture as reported by the hardware, and n is the number of features. If texture arrays are not supported, the feature growth table is split up in a similar way, but layers are stored in the depth dimension of a 3D texture, which is functionally almost identical. However, accessing a 3D texture is slower than accessing a 2D texture array, and it might be necessary to pad the depth dimension to a power-of-two.

Merging features: Features can merge as the parameter t increases, e.g., when two or more small disjoint but close features grow in size over t until they finally touch and thus merge. When this happens at time t_k, they are treated as a single feature for all $t \geq t_k$. In order to do so, we assign a new *current feature ID* to the entire merged feature, using the ID of the (sub-)feature with the largest voxel population of the features that have merged. During rendering, the feature ID that is current at a time t_i needs to be determined for a given voxel (sample). Thus, we store the current feature ID corresponding to each t_i ($ID(t_i)$) in the feature growth table, as explained above. Every row in this table corresponds to a *feature birth ID*, i.e., the ID that was assigned on feature creation, whereas the *current feature ID* is retrieved from the ID entry in column t_i. Thus, the IDs stored in the feature volume are feature birth IDs instead of current IDs. The big advantage of this approach is that it allows the IDs stored in the feature volume to be left untouched by the merging of features. It is sufficient to know the feature birth ID for each voxel, i.e., the feature the voxel first belonged to. Everything else can be obtained from the feature growth table, i.e., for any t_i, the current ID and current size of the feature are retrieved from the feature growth table in row *feature birth ID* and column t_i.

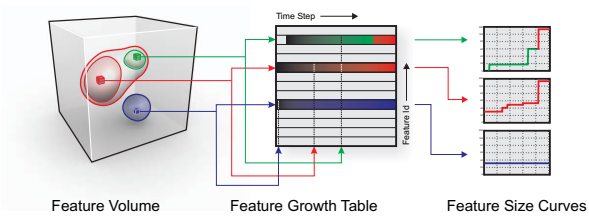

Fig. 3. A feature volume stores only the per-voxel information that is necessary in order to reconstruct the feature size curve of each voxel using the per-feature growth information stored in the feature growth table, where each row corresponds to one feature.

4.4 Multi-Pass Region Growing and Seed Selection

Both the feature volume and the feature growth table are computed using region growing in multiple passes, where each pass corresponds to a specific time step t_i. Figure 4 illustrates the overall region growing process for three consecutive time steps. The resulting feature size curves over time and the feature volume and feature growth table used to store them are illustrated in Figures 2 and 3, respectively.

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

Culling: In industrial CT parts a significant number of voxels are usually part of the background, i.e., air. In order to speed up the pre-computation stage, it is worthwhile to remove these parts of the volume from the potential seed candidates. We do this by allowing the user to specify an opacity TF, which is used to cull small sub-blocks of the volume (e.g., 32^3). The simplest TF for culling is a simple windowing function. In our system, the default setting is a window over the entire density range, which disables culling, but can be changed by the user before pre-computation is started. Culling determines an active block list, and the voxels contained in active blocks are considered as potential seeds.

Region growing: Given the active block list determined by culling, which might contain the entire volume, selection of seed candidates for region growing proceeds by processing block after block, considering each contained voxel in turn.

In each pass, for each seed candidate, a region is grown as far as possible given the current parameter t_i. In order for a region to become a feature, its size must be both larger than a given minimum (not *too small*), to avoid spurious features of only a few voxels due to noise, and smaller than a given maximum (not *too big*), to avoid turning entire structures such as holes into features. These two size thresholds are specified globally and are set to very conservative values by default, which is usually sufficient and thus no specification by the user is necessary. If a region satisfies these two criteria, a new feature is created from it. All voxels comprising this feature will subsequently not be considered as seed candidates. Region growing then continues by considering the next seed candidate. Furthermore, in all passes after the first one, in addition to starting completely new features, existing features have to be grown further if allowed for by the increased region growing parameter t_{i+1}. The distinction between first and subsequent passes, as well as handling multi-pass region growing efficiently, is described below.

In the first pass, $i = 0$, with a corresponding region growing parameter $t_i = t_0$, every voxel that is not yet part of any feature is considered as seed candidate for starting a completely new feature. Note that since seed candidates are considered sequentially, even in the first pass many voxels may already have been assigned to features when a given candidate is processed. In the next pass, $i+1$, and all subsequent passes, (1) as yet unassigned voxels are considered as new seed candidates, possibly starting a new feature at time t_{i+1}; and (2) already existing features are grown further from their boundary voxels, if possible given the new region growing parameter t_{i+1}. If an existing feature grows further at time t_{i+1}, its feature size increases, i.e., $f_s(\mathbf{x}, t_{i+1}) > f_s(\mathbf{x}, t_i)$, see Figures 2 and 3.

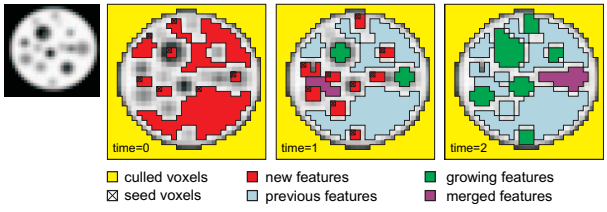

Fig. 4. Region growing is performed in multiple passes. In each pass, new features can be created, previously existing features may grow (except in the first pass), and features may merge. Optionally, background voxels can be culled in order to improve pre-processing performance.

The following approach allows handling all cases efficiently and treating the first and all subsequent passes as uniformly as possible. We maintain two bit masks with one bit per voxel for tagging and thus removing voxels from further consideration for either growing a new feature or extending an existing feature. A voxel is considered tagged when its bit is set in either one of the two masks (or both).

The goal of the *checked mask* is to remove voxels from consideration for growing a new feature (where it would be a seed voxel), or extending an existing feature (where it would be a voxel in the feature's boundary), both in the current and all subsequent passes. This mask is cleared only before the first pass and then updated from pass to pass. A bit in the checked mask is set when either (a) the voxel is *inside* a feature, i.e., all its neighboring voxels also belong to the feature (e.g., using the 26-neighborhood); or (b) because the voxel is part of a region that became *too big* (see above).

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

During region growing, a voxel is a candidate for either growing a new feature or growing an already existing feature further when neither its bit in the checked mask nor in the visited mask is set. The distinction between these two cases is done according to the corresponding entry in the feature volume. The entry there contains a valid birth feature ID if the voxel already belongs to a feature, and thus is a candidate for growing the feature further. It contains an invalid ID when the voxel does not yet belong to a feature, and thus is a seed candidate for growing a completely new feature.

4.5 Region Growing Criteria

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

Region growing method A: We are using a variant of seeded region growing [1] that is also able to include a region's boundary. Region growing is performed in two distinct phases:

1. *Grow the homogeneous "core" of a region.* A voxel is added to the region when the difference of its density to the average density of the whole region is below a given threshold ε: $|v - v_r| < \varepsilon$, where v is a voxel's density, and v_r is the current region's average density. After a new voxel is added, v_r is updated accordingly.

2. *Expand the region by including its boundary.* For every voxel adjacent to the region, we either check a gradient magnitude criterion in order to decide whether this voxel should be included in the region, or use the voxel's LH value [14].

The time parameter t determines the current homogeneity criterion: $t = \varepsilon$. Further, it is possible to separate these two phases in such a way that they can be distinguished later on during interactive exploration. By advancing the time parameter t between the two phases, *even* bins of t correspond to region cores without their boundary, and *odd* bins correspond to regions including their boundary. This allows to select regions with or without their boundaries in the 3D TF.

Region growing method B: Huang et al. [10] are using a combination of region growing based on the standard deviations of density and gradient magnitude, respectively. They determine both standard deviations for a fixed neighborhood of each seed voxel. The main parameter of their method is a global scale factor $k > 0$:

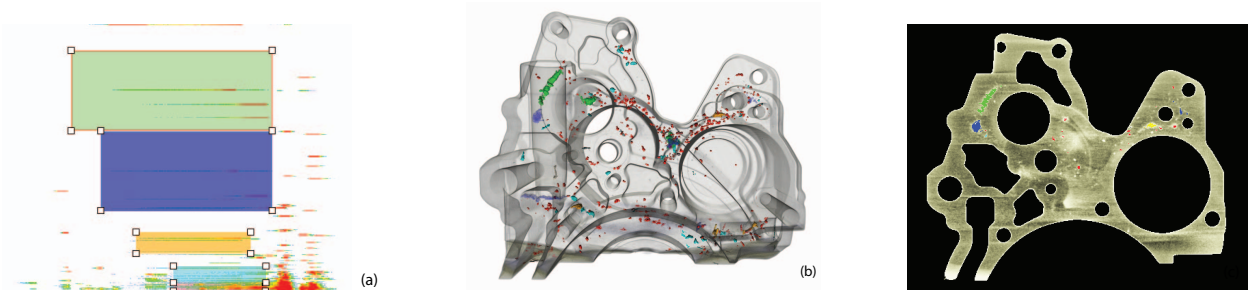

Fig. 5. Apart from inspecting individual features, the feature classification space can be explored through a stack of 2D histograms spanning the 3D domain of *(density, feature_size, time)*. (a) 2D slice through the domain with histogram and transfer function widgets (x axis: *density*, y axis: *feature_size*); (b) Volume view generated with GPU-based real-time ray-casting; (c) Orthogonal slice views can also be used for picking features.

$$fca = \frac{|v - v_s|}{k\sigma_v}, \quad (2)$$

$$f_{cb} = \frac{|g - g_s|}{k\sigma_g}, \quad (3)$$

$$fcc(p) = pfca + (1-p)f_{cb}, \quad (4)$$

where v is a voxel's density value, v_s the density of the seed, g a voxel's gradient magnitude value, g_s the gradient magnitude of the seed, and σ_v and σ_g are the corresponding standard deviations of the seed neighborhood, respectively. The factor p can be set to a constant value but is set by default to $p = \frac{\sigma_g}{\sigma_v + \sigma_g}$. We employ these region growing criteria in our framework by setting $t = k$ for tracking the main parameter k.

5 Exploration

For the user, the exploration stage is the most visible part of our pipeline. The complexities from the previous pre-computation stage are hidden to a large extent. During exploration, the current classification is shown in real-time in a 3D volume view (Figure 5 (b)) and three orthogonal slice views (Figure 5 (c)). In order to explore and classify features and feature classes, the corresponding regions in the volume can be mapped to color and opacity using one of two different means: (1) via a 3D TF in the *(density, feature_size, time)* domain (Section 5.1; Figures 5 (a) and 6), which maps entire feature classes; or (2) directly via picking individual features in one of the slice views or the *graphical feature view* (Section 5.2; Figure 7). This special view allows the user to pick features, inspect their feature size curves, and set their color and opacity individually, which can also be used for disabling features that stem from CT artifacts by setting their opacity to zero. Picked features are immediately highlighted in all views.

An important concept during exploration is the handling of the time axis t. Showing all time steps simultaneously is only supported by the graphical feature view, where feature size curves can be inspected along the time axis in a function plot, and the dense color-coding of feature IDs shows their evolution over time (due to creation and merging of features), which constitutes the main part of the view (Figure 7). All other views, i.e., the 3D volume view, the three orthogonal 2D slice views, and also the TF panel depict only a single time step. This *current time step* t_{cur} is specified globally for exploration and can be modified by the user at any time via a simple slider.

5.1 Exploring Feature Classes

The main goal of classification is to explore feature classes, instead of requiring the user to inspect individual features. Features are classified by specifying a TF in the 3D domain of *density* (from the CT volume), *feature size* (retrieved from feature size curves), and *time* (the changes of features according to the main region growing parameter). Although this is a 3D domain, we use a 2.5D metaphor to make the manual specification of TFs manageable. The 2D $(density, feature\_size)$ subdomain can be viewed in its entirety for any given time t_{cur} in the TF panel. This corresponds to choosing a specific time of interest and then exploring features according to their size and density distribution. Figure 6 illustrates the 3D TF domain, highlighting two selected 2D subdomains and the widgets intersecting them.

Feature histograms: The background of Figure 5 (a) is a 2D histogram plotting voxel density (x axis) against feature size (y axis). The number of voxels with a given $(density, size)$ combination is color-coded (red corresponds to a large number of voxels). For each time step t_i, a corresponding 2D histogram is computed, which are together maintained as a stack of 2D histograms that collectively span the entire 3D domain. In order to gain insight into the distribution of feature sizes, densities, and their occurrence in time, the time axis is explored using the slider for t_{cur}, which specifies the current time of interest.

3D TFs and 2.5D widgets: The 3D TF in the *(density, feature_size, time)* domain is specified using 2.5D widgets, which result from extending some of the well-known regular 2D TF widgets such as boxes, tents, or Gauss blobs [13] into the third dimension by assigning a time range $[t_a, t_b]$ to each widget using a range slider. This range determines in which time steps this widget is active, i.e., 2D widgets are extruded into 3D from time t_a to time t_b. The actual widget shape is 2D, e.g., a Gauss blob is extruded into a cylindrical shape in 3D. The reason for this is that opacity ramps are very useful in the $(density, size)$ subdomain, but gradually changing the opacity classification over time is not meaningful because the time axis is in fact not continuous (it is not only sampled, but also corresponds to the impact of fixed increments in the main region growing parameter on the evolution of regions, not actual time). Thus, a widget is either fully present at a time t_i with $t_a \leq t_i \leq t_b$, or not present at time t_i at all. In many cases, in order to determine a specific feature class the user explores the time domain until a time step is found that depicts the features of this class well. In this case, widgets for this class are set to be active only in this particular time step, i.e., $t_a = t_b$. Example TFs are also shown in Figures 8, 9, and 10.

5.2 Exploring Individual Features

In addition to exploring whole classes of features, it is important to also allow the user to pick and inspect individual features. Features can be picked with the mouse in either (1) one of three orthogonal slice views, which retrieves the current feature ID at the picked location; or (2) in the graphical feature view. The graphical feature view

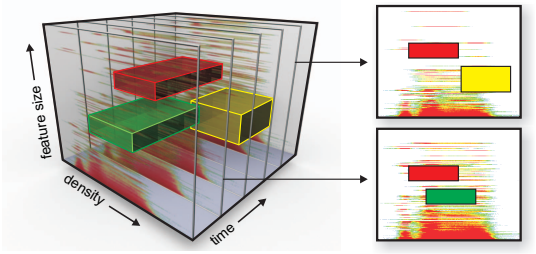

Fig. 6. TF with 2.5D widgets in the 3D *(density, feature_size, time)* domain. A stack of 2D *(density, feature_size)* histograms, one for each time step, helps with TF specification.

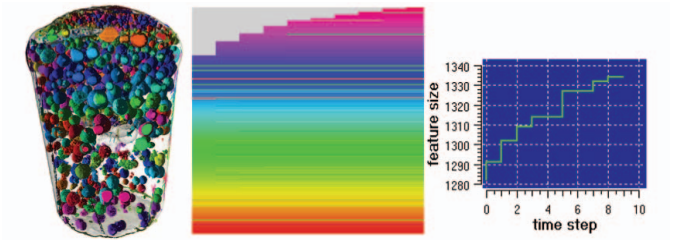

Fig. 7. Graphical feature view, where each row (middle image) corresponds to a feature and IDs are color-coded. The horizontal axis is time t. A plot of the feature size curve of the currently picked feature is also shown (right image). The color coding of IDs can be used during volume rendering for immediate inspection of the result of region growing for a given time step t_{cur} without specifying a 3D TF (left image).

(Figure 7) has two main components, a visualization of all features with their feature IDs color-coded and depicted over time (Figure 7, center), and a plot of the representative feature size curve of the currently picked feature (Figure 7, right). The former visualization contains one row for each feature, with the vertical coordinate corresponding to the feature ID (increasing from bottom to top). The horizontal axis is time t, where horizontal changes in color indicate the merging of features and thus a change in feature ID. The gray areas (top left) correspond to features that are only created at later time steps, i.e., who have no valid feature ID before their *feature birth time*. This view depicts the feature growth table described in Section 4.3 as a color-coding of the $ID(t)$ channel from the $(f_s(\mathbf{x}_s,t_i),ID(t_i))$ pairs stored in the table. As such, it is a visual representation of the behavior of all features over time with respect to their creation and merging with other features. When features merge, their ID changes (except for the feature with the largest voxel population of the merging features, which is kept, see Section 4.3). This shows up as color changes within a row in this view. This color-coded view does not allow detailed analysis but provides a good overview at a glance whether a lot of features are merging or not, and at what time steps a lot of merges occur. Detailed inspection is then possible by picking a feature (row), and looking at the corresponding plot that shows all details of the feature's size curve.

- **Feature picking:** When a feature is picked, a specific individual color and opacity can be specified, which is then stored per feature by overriding the corresponding entry in the 1D color ramp TF described in the next paragraph.
- **Feature color coding:** All features can automatically be shown in different colors, by mapping feature IDs to colors and opacity via a 1D transfer function, which is filled with a color ramp by default. This is the same color ramp used in the graphical feature view (Figure 7, center). Figure 7 (left) shows this color mapping applied in the 3D volume view, which is useful to gain a quick overview before transfer functions are specified.
- **Removal of artifacts:** Picking features is also very useful for removing erroneous features that in fact are artifacts from the scanning process, such as reconstruction/Feldkamp artifacts or center/circular artifacts. When an artifact is picked, its individual opacity can be set to zero, which removes it from both rendering and quantification.

5.3 Volume Rendering

Volume rendering is performed by using ray-casting in the fragment shader [15]. Since industrial CT volumes are quite large and a feature volume is required in addition to the density volume itself, we employ bricking in conjunction with single-pass ray-casting [7], keeping only the visible subset of the entire volume in GPU memory. The following pseudo-code (similar to GLSL) illustrates the main steps that need to be done in order to determine the color and opacity (without shading) of a given sample, as RGBA tuple in the `vec4` variable `out`:

```
float density = texture3D(density_volume, sample_coord3);
vec2 feat_vox = texture3D(feature_volume, sample_coord3);
float birthID   = feat_vox.x;
float birthTime = feat_vox.y;
if ((birthID == ID_NONE) || (birthTime > T_CUR)) {
  out = texture1D(tf1D, density);
} else {
  vec2 fsmap = texture2D(growth2D, vec2(T_CUR, birthID));
  float curSize = fsmap.x;
  float curID   = fsmap.y;
  if (curID == PICKED_ID) {
    out = pickingColor;
  } else {
    out = texture1D(selection1D, curID);
    if ( (out.a > 0.0) && !use_color_ramp_1D )
      out = texture3D(tf3D, vec3(density, curSize, T_CUR));
    if (out.a == 0.0)
      out = texture1D(tf1D, density);
  }
}
```

which can then be composited during ray-casting, or simply displayed in an orthogonal slice view. The volume is rendered for a specific time step, i.e., the global current time t_{cur}, which is denoted as `T_CUR` in the code. For a sample with volume coordinates `sample_coord3`, the density volume (`density_volume`) and the feature volume (`feature_volume`) are sampled at that position, which yields the density and the feature birth ID (`birthID`) and time (`birthTime`). When no feature is present at that location given the current classification, i.e., no feature exists there at all (`ID_NONE`), or the feature does not yet exist at time t_{cur}, or is mapped to zero opacity in the feature TF (`tf3D`), the regular 1D density TF (`tf1D`) is used. The feature growth table is `growth2D`. For color-coding features or using individual colors and opacity (see Section 5.2), a 1D table is used (`selection1D`), which overrides the feature TF when `use_color_ramp_1D` is set.

6 Quantification and Results

In order to assess the quality of materials or the whole casting process itself, it is necessary to quantify the features contained in a data set, e.g., their number, volume, surface area, as well as global statistical measures such as average volume and standard deviation. The focus of our system is to provide the basis for interactively specifying *what* should be quantified, as a basis for a variety of actual quantification options. Our system computes and displays quantities corresponding to feature classes selected via the 3D TF, individual features, as well as overall information computed in the pre-computation stage.

6.1 Feature Quantification

In contrast to rendering during exploration, quantification does not primarily consider individual voxels (samples), but whole features with all their voxels. Therefore, quantification does not need the feature volume but relies mainly on the feature growth table (Section 4.3), which contains the representative feature size curves for every feature, as well as information computed during region growing that is not needed for rendering, such as lists of voxels comprising individual features. For a feature f, the representative feature size curve $f_s(\mathbf{x},t)$ is the feature size curve of the feature's seed voxel $f_s(\mathbf{x}_s,t)$, see Section 4.1. The size of the feature in voxels at time t_i can be determined directly from this representative feature size curve, which is stored in the feature growth table. However, this considers only the region growing process itself, not the classification done via the TF which can exclude features and whole feature classes from quantification. During the region growing process (Section 4.4), a list of contained voxels is incrementally constructed for each feature. In order to quantify a feature, these voxels have to be visited, and their density, together with the feature's size at time t_i, must be used to perform a lookup in the 3D feature TF. The resulting opacity determines whether this voxel should be included in the quantification or not. In addition to using the opacity, feature classes can be quantified individually by either quantifying

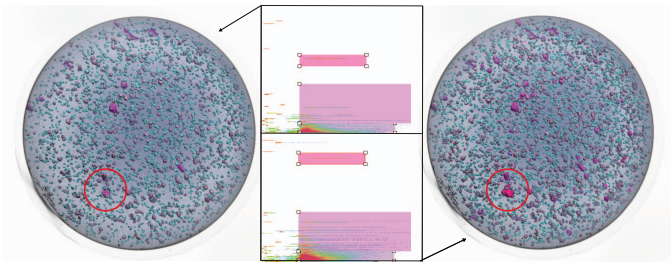

Fig. 8. A golf ball, reinforced with dense particles. Two different time steps and the corresponding 2D section of the 3D TF are shown. In the earlier (left) time step, the indicated particle is still small because the region growing parameter has not advanced far enough yet. In the later (right) time step, it has become a big feature that does not grow further since it has reached its maximum (actual) size. See also Table 1.

each widget's classification separately, or combining the classification of several widgets that collectively classify a single feature class.

We compute the most common measures such as the volume of features in voxels, or mapped to cubic millimeters via the reference size of a single voxel in x, y, z, the surface area of a feature, density average and standard deviation, as well as global statistical measures such as average feature size and standard deviation. However, additional measures can be added easily.

6.2 Results

Table 1 illustrates exemplary quantification results for two of the data sets we have used in this paper, see also Table 2.

Figure 5 depicts a part of a cast housing for the automotive industry. The part is produced of an AlSi-type alloy in a die-casting process. As it carries fluids during operation, impermeability of the housing is one point of specification. Therefore a characterization of voids in the casting has to be performed. Table 1 (top) gives quantification results for different void sizes (feature classes) contained in this data set, including the number of features in each class, their average volume in voxels, and the percentage of voxels in the class with respect to the number of voxels in the whole part (excluding the surrounding air).

Figure 8 shows a 2-piece construction golf ball. The inner piece is reinforced with dense particles. To evaluate the quality of these reinforcements, their distribution and size were checked using industrial CT. The two different time steps in Figure 8 clearly show the influence of the region growing parameter (here, k of region growing method B, Section 4.5) on the quantification result at the end of the pipeline. A 3D TF was used to obtain an optimal result for the complex structure of the sample. Table 1 (bottom) contrasts the quantification results of two selected time steps. The user visually determined that the earlier time step (values in parentheses) corresponded to incomplete results, whereas the later one resulted in a plausibly complete detection of features. For example, the particle indicated in Figure 8 corresponds to an agglomeration of smaller particles during the production of the golf ball. It also appears in the quantification as a singular particle of large size (last row of Table 1).

Data set	class	feature count	avg.vol. [voxels]	% of part vol.
Cast	small	337	158.1	0.041
Housing	med.	16	717.2	0.088
(Fig. 5)	large	6	1629.7	0.075
	XL	6	5181.3	0.024
	XXL	3	10678.7	0.026
Golf	small	6131 (4375)	70.2 (83.5)	0.044 (0.037)
Ball	med.	1875 (948)	209.1 (224.2)	0.040 (0.022)
(Fig. 8)	large	78 (26)	1152.2 (1029.8)	0.092 (0.027)
	XL	1 (1)	9472 (3441)	0.0097 (0.0035)

Table 1. Example quantification results. The results for the golf ball are for the time step selected by the user as the "complete" one (Fig. 8, right), and an earlier, "incomplete" time step in parentheses (Fig. 8, left).

Figure 9 shows a "reduced-pressure-test sample" (RPTS), which is used in the casting industry to evaluate the gas content of an aluminum or copper melt. Therefore about $30cm^3$ ($1.83in^3$) of the melt is solidified at a pressure of 8000 Pa (1.16 psi), which causes the gas in the melt to form pores in the metallic volume. Furthermore, the shrinkage of the metal during solidification causes shrinkage cavities to be formed in the center of the upper regions of the sample. Therefore, these samples are ideal test pieces for the evaluation of feature detection, as they are virtually full of different pore sizes and shrinkage cavities. The evaluation of those basic types of features is of great relevance to the casting industry as they are the most common defects in metallic cast parts beside non-metallic inclusions. The distinction between these defects is of special interest for the casting expert, as they are both voids but have a completely different origin, and therefore have to be treated differently in the casting process. For this application, the time step t_{cur} in our system was chosen interactively, such that the rendered features comply with the actual position and size of the different defects. In this case, a (2D) feature TF for the time step t_{cur} was sufficient in order to classify the gas pores according to their size, and the shrinkage cavities can be separated by their morphologies.

The asphalt drilling core depicted in Figure 10 with a diameter of $100mm$ ($3.9in$) is used to characterize the quality of asphalt. It is composed of three main phases: the mineral phase, the bitumen binding phase, and pores. The mineral phase can be composed of different minerals in different grain sizes. This highly complex composition concerning density profile and dimensional range makes a reliable evaluation especially difficult. Such samples ideally show the advantages of interactive feature detection. Due to the fundamentally different behavior and composition of the employed phases, a 3D TF can be used to separately characterize the different phases. Figure 10 shows the result of two different feature TFs at a specific time t_{cur}.

6.3 Performance and Memory Usage

Table 2 gives typical numbers for pre-computation times and volume rendering frame rates for the data sets used in this paper. Performance has been measured on an Athlon 64 2.4GHz, 4GB, XP64, and a GeForce 8800 GTX, 768MB. The first region growing step always consumes the most time. It has to compute the additional data required, such as mean and standard deviation of the seed voxel neighborhood, considers the most voxels as seeds, and grows and discards all regions that become *too big*. All following time steps are then much faster. The bit masks maintained during region growing ensure that many voxels are visited only once over time, which reduces processing time for further iterations. Volume rendering is fast, as only a few additional operations compared to regular volume rendering have to be executed per fragment, see the pseudo code in Section 5.3.

Table 2 also lists the memory usage of the major additional data structures computed, i.e., the feature volume (first value) and the fea-

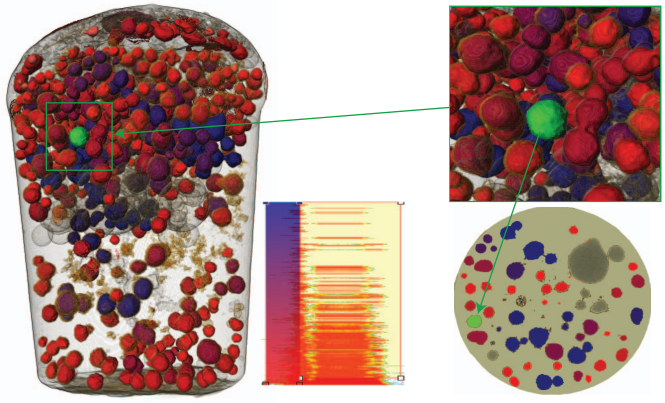

Fig. 9. Reduced-Pressure-Test Sample (RPTS). Lower densities are the interior of pores (red-blue), higher densities their boundaries (yellow). Feature size is mapped with a color gradient from red (small) to blue (large). Very large features are set to transparent. An individual picked feature is highlighted in green, here of size 7939 voxels and $9.02mm^3$.

Data Set	Resolution	Feat.-Mem.	Pre-comp. (1st, 2nd, 3rd..*n*th time step, overall for 16 steps), Method A||B								Rendering
Cast Housing	667x465x512	606MB+128KB	600 s	8.7 s	7-8 s	12.1 min	421 s	0.84 s	0.5-0.9 s	7.2 min	16-22 fps
Golf Ball	512x512x256	256MB+448KB	198 s	3.8 s	3-4 s	4.2 min	447 s	7.2 s	5-6 s	8.9 min	22-28 fps
RPTS	373x377x512	275MB+192KB	280 s	64.6 s	45-66 s	20.5 min	260 s	146 s	29-85 s	15.1 min	20-23 fps
Asphalt Core	512x512x256	256MB+1.3MB	253 s	27.4 s	17-21 s	9.4 min	450 s	24 s	10-15 s	10.9 min	20-40 fps

Table 2. The data sets we have used in this paper, with typical pre-computation times (first and second time step, range for 3rd+; and overall time for 16 time steps), and typical volume rendering frame rates (viewport 512x512). The left four columns of pre-processing use region growing method A, and the right four columns method B (Section 4.5). The first step is the most expensive; after the second step, computation times decrease rapidly.

ture growth table for 16 time steps (second value). For rendering, only a subset of the entire feature volume needs to be resident in GPU memory due to texture bricking, whereas the feature growth table is always resident in texture memory in its entirety.

7 Conclusions and Future Work

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

We believe that the concept presented in this paper is very powerful, but it is also only one step toward driving defect and feature detection by visualization and bringing visualization methods and segmentation closer together. There are many possibilities that can be explored in the future. We have used two different simple region growing methods as a proof-of-concept of our general framework, and would like to explore additional options in the future. We would also like to investigate adaptive sampling schemes of the parameter (time) domain. Extending the basic concept further, higher-dimensional parameter spaces would enable exploration of a wider variety of possible segmentations. Semi-automatic transfer function generation in our new 3D TF domain or similar domains would also be a worthwhile venue of future research. Finally, we are also planning to extend the possibilities for quantification, specifically with respect to global measures such as porosity.

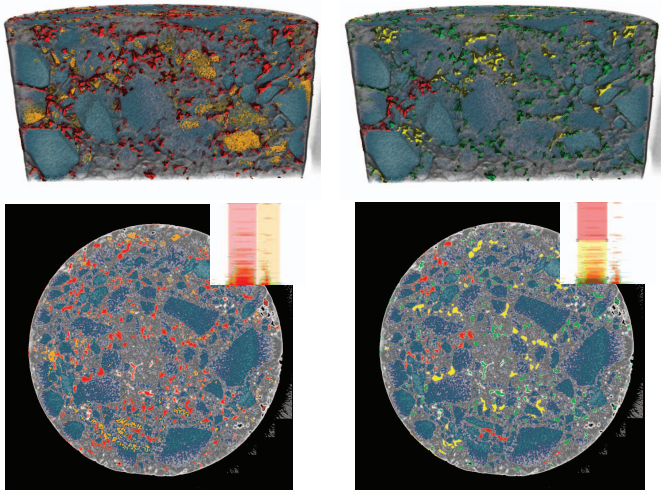

Fig. 10. Asphalt drilling core with different material components. Region growing yields features of two clearly distinguishable density ranges. The higher density range (orange widget in left column) corresponds to mineral components of higher density incorporated in the coarse fraction of the asphalt, which in this case are undesired features. Mapping them to completely transparent (right column) removes them. The phases between the coarse mineral components of lower density (red widget, left column) can be further distinguished according to feature size, giving different constituents of the fines (right column): small features (green), medium-sized features (yellow), and large features (red).

Acknowledgements

We would like to thank the anonymous reviewers, Johanna Beyer, Daniel Habe, Siegfried Schider, Raphael Fuchs, Sebastian Zambal, Jiří Hladůvka, and the Austrian funding agency FFG.

References

[1] R. Adams and L. Bischof. Seeded region growing. *IEEE Trans. Pattern Anal. Mach. Intell.*, 16(6):641–647, 1994.

[2] C. Bajaj, V. Pascucci, and D. Schikore. The contour spectrum. In *Proceedings IEEE Visualization '97*, pages 167–ff, 1997.

[3] M. F. Cohen, J. Painter, M. Mehta, and K.-L. Ma. Volume seedlings. In *Proc. ACM Symp. on Interactive 3D Graphics*, pages 139–145, 1992.

[4] N. Copty, S. Ranka, G. Fox, and R. V. Shankar. A data parallel algorithm for solving the region growing problem on the connection machine. *Journal of Parallel and Distributed Computing*, 21(1):160–168, 1994.

[5] B. Eckel. *OpenGL Volumizer Programmer's Guide*. Silicon Graphics, Inc., 1998.

[6] K. Engel, M. Hadwiger, J. M. Kniss, C. Rezk-Salama, and D. Weiskopf. *Real-Time Volume Graphics*. A K Peters, Wellesley, Mass., 2006.

[7] M. Hadwiger, C. Sigg, H. Scharsach, K. Bühler, and M. Gross. Real-time ray-casting and advanced shading of discrete isosurfaces. *Computer Graphics Forum*, 24(3):303–312, 2005.

[8] S. Horowitz and T. Pavlidis. Picture segmentation by a directed split-and-merge procedure. In *Proc. Pattern Recognition*, pages 424–433, 1974.

[9] R. Huang and K.-L. Ma. RGVis: Region growing based visualization techniques for volume visualization. In *Proc. Pacific Graphics*, 2003.

[10] R. Huang, K.-L. Ma, P. McCormick, and W. Ward. Visualizing industrial CT volume data for nondestructive testing applications. In *Proceedings IEEE Visualization 2003*, pages 547–554, 2003.

[11] G. N. Khan and D. F. Gillies. Parallel-hierarchical image partitioning and region extraction. In *Computer Vision and Image Processing*, pages 123–140, 1992.

[12] G. Kindlmann and J. W. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *Proc. IEEE Symposium on Volume Visualization '98 (VolVis '98)*, pages 79–86, 1998.

[13] J. Kniss, G. Kindlmann, and C. Hansen. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *Proceedings IEEE Visualization 2001*, pages 255–262, 2001.

[14] P. Šereda, A. V. Bartroli, I. W. Serlie, and F. A. Gerritsen. Visualization of boundaries in volumetric data sets using LH histograms. *IEEE Transactions on Visualization and Computer Graphics*, 12(2):208–218, 2006.

[15] S. Stegmaier, M. Strengert, T. Klein, and T. Ertl. A simple and flexible volume rendering framework for graphics-hardware-based raycasting. In *Volume Graphics*, pages 187–195, 2005.

[16] S. Tenginakai, J. Lee, and R. Machiraju. Salient iso-surface detection with model-independent statistical signatures. In *Proceedings IEEE Visualization 2001*, pages 231–238, 2001.

[17] M. van Kreveld, R. van Oostrum, C. Bajaj, V. Pascucci, and D. Schikore. Contour trees and small seed sets for isosurface traversal. In *Proc. ACM Symp. on Computational Geometry*, pages 212–220, 1997.

[18] S. Zucker. Region growing: Childhood and adolescence. *Computer Graphics and Image Processing*, 5:382–399, 1976.