FAKULTÄT FÜR !NFORMATIK

Classification and Visualization of Volume Data Using Clustering

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Computergraphik & Digitale Bildverarbeitung

eingereicht von

Andreas Opitz
Matrikelnummer 0226294

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Betreuer: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
Mitwirkung: Dipl.-Inf. Dr.techn. Peter Kohlmann

Wien, 18. 11. 2009

(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

Erklärung zur Verfassung der Arbeit

Andreas Opitz
Gugitzgasse 6/12/3, 1190 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)

Abstract

The increasing prevalence and improvement of imaging techniques like computed tomography or magnetic resonance imaging enhances the resolution and thereby the amount of the acquired data. This development leads to a rising demand for means which are efficient and easy to use in order to evaluate the desired information. As a volumetric data set may contain various objects and materials, a transfer function is used to steer visibility and coloring of the different structures based on certain properties. Standard transfer functions which rely on scalar values only, rapidly come up against limiting factors because of their coarse selectivity and liability to noise. The usage of more complex methods leads to increased demands concerning the user's knowledge and requires some practice in order to obtain good results. This thesis provides an overview of volume visualization and techniques for the creation of transfer functions. It also deals with various approaches to clustering and presents an intuitive method for the design of transfer functions based on Mean Shift Clustering and LH histograms.

Zusammenfassung

Mit der zunehmenden Verbreitung und Weiterentwicklung von bildgebenden Verfahren, wie zum Beispiel der Computertomographie oder der Magnetresonanztomographie in den Gebieten der Medizin und der Industrie, steigen nicht nur die Auflösung und somit die Datenmenge, sondern auch die Anforderungen an eine möglichst effiziente Auswertung der gesuchten Informationen. Da in einem Volumendatensatz verschiedenste Objekte und Materialien enthalten sein können, wird eine sogenannte Transferfunktion verwendet um Sichtbarkeit und Farbe der verschiedenen Strukturen zu kontrollieren. Eine einfache, nur auf Skalarwerten basierende Transferfunktion stößt aufgrund ihrer groben Selektivität und Anfälligkeit gegenüber fehlerhaften Messwerten schnell an ihre Grenzen. Bei ausgefeilteren und komplexeren Methoden steigen jedoch die Anforderungen an den Benutzer, erfordern zumindest Grundkenntnisse der Thematik und einiges an Übung um gute Ergebnisse erzielen zu können. Diese Arbeit gibt einen Überblick über die Materie der Volumenvisualisierung und der Transferfunktionen. Ebenso werden verschiedene Ansätze von Clustering betrachtet und ein möglichst intuitive Methode zum Entwurf von Transferfunktionen basierend auf Mean Shift Clustering und LH-Histogrammen präsentiert.

Acknowledgments

I want thank Dr. Peter Kohlmann and Prof. Eduard Gröller for their help, advice, and the supervision of my thesis as well as Dr. Markus Hadwiger and Dr. Petr Sereda for their hints and ideas.

I want to thank my friends and especially my family for their support and encouragement throughout my work.

Contents

1 Introduction

This thesis deals with the classification and visualization of different objects or materials contained in a volumetric data set. Such a data set can be created using imaging techniques like computed tomography (CT) or magnetic resonance imaging (MRI). Especially for complex structures it is desirable to have a three-dimensional representation which can be analyzed and interacted with more intuitively than by simply looking at individual slice images.

The following section gives an overview of volume rendering in general and presents different volume rendering techniques. Chapter 2 deals with various approaches to transfer function design whereas Chapter 3 focuses on clustering techniques which can be used to automate this process. Chapter 4 explains the implementation of the automatic classification and visualization of the data. In Chapter 5 visualizations created with the new technique are shown, and the performance of the algorithm is evaluated. Chapter 6 summarizes this thesis.

1.1 Volume Data

In general, a volume is considered as a continuous, three-dimensional field of data values. However, for the practical implementation of volume rendering techniques, a discrete representation is used. Analogous to raster images where a picture element (pixel) stores the color for a specific raster position, a volume element (voxel) stands for the scalar value in a discrete volumetric data set. There are two possible interpretations of such a voxel concerning its basic structure. Either it can be seen as a single point or as a cube. In the first case, the scalar value is only defined at the grid points, and the

region between two voxels is considered to be empty. Values in between are obtained by using interpolation methods. In the second case a voxel is treated as a cube which contains a single data value. The decision which of the two interpretations should be used, depends on the chosen rendering technique. Other considerations about the access of the data concern which kind of grid type is used. It is possible to store the values in uniform grids where the cuboid cells are aligned parallel to the coordinate axes or in unstructured grids, like tetrahedral grids [SML97]. Unstructured grids are commonly used, for example in computational fluid dynamics (CFD) or computed electromagnetic fields. Although this approach provides more flexibility, uniform grids allow an easier and more efficient access to the data in computer memory for volume rendering applications. As a consequence it is not uncommon that simulation data is mapped to a uniform grid for visualization purposes only.

Basically, there are three sources for the data. The first one is the measurement of physical phenomena using different scanning methods, such as CT [Bus00]. It has its origins in medical imaging and has been adapted for the usage in industrial applications of non-destructive testing (NDT). The acquisition is based on X-rays which are emitted onto the object of interest. The radiation interacts with the different materials, leaves the object and is measured by a sensor on the opposite side of the emitter. By repeating this procedure from various angles, a slice image of the object can be reconstructed. When shifting the object through the scanner at small steps, a series of those images can be combined to form a volumetric data set. Another medical imaging technique, MRI, uses a strong magnetic field to align the spins of atomic nuclei, as described by Markisz and Aquilia [MA96]. This alignment is modified by a radio frequency pulse, and when the spins return to their previous orientation they emit energy which is measured and used for reconstruction of the volume.

1.2 Interaction of Volume and Light

Volume rendering is based on the modeling of the interaction of light with the media it passes through, and the radiance that reaches a virtual camera is used for the visualization. When traversing the volume, light can be emitted, absorbed, or scattered. As a

consequence, an equation for the transfer of light can be created which represents this physical model. The complete equation is not applicable for practical purposes because the solution would not be possible in reasonable time. Therefore, simplified variants which consider only parts of the model have been introduced. The most common one is the emission-absorption model which does not take scattering or indirect illumination into account. This leads to the volume rendering integral

$$I(D) = I_0 e^{-\int_{s_0}^{D} \kappa(t)dt} + \int_{s_0}^{D} q(s) e^{-\int_{s}^{D} \kappa(t)dt} ds \tag{1.1}$$

as described by Engel et al. [EHK+06]. The initial radiance I_0 at entry position s_0 is attenuated by the volume during traversal using the absorption coefficient κ. Additionally, the emission is represented by $q(s)$ and contributes to the radiance $I(D)$ at exit position D for the remaining distance along the ray. Due to the computational complexity of an analytical solution, which is possible only under certain circumstances, a discretized variant of the equation is used instead. A numerical solution can be achieved by splitting the integration domain into a finite number of intervals and evaluating it as a series of summations and multiplications. To simplify this procedure even further, compositing schemes such as front-to-back (FTB) compositing are introduced where accumulated color c_i and opacity α_i are expressed as

$$\begin{aligned} c_i &= c_{i-1} + (1 - \alpha_{i-1})\alpha(x_i)c(x_i) \\ \alpha_i &= \alpha_{i-1} + (1 - \alpha_{i-1})\alpha(x_i) \end{aligned} \tag{1.2}$$

with c_{i-1} being the accumulated color and α_{i-1} the accumulated opacity of the previous step. The final values are obtained by stepping along the ray starting at the position closest to the camera until the end of the volume has been reached. At each step the color $c(x_i)$ and the opacity $\alpha(x_i)$ for the current position, which have been defined in a transfer function, are weighted and incorporated into the result. Basically, a transfer function is a mapping which assigns optical attributes to each position in the volume. A common method is the definition of a lookup table where these attributes are stored for each scalar value. There are also other compositing schemes such as back-to-front (BTF) compositing where the ray is traversed from the back, instead of starting at the

side of the volume facing the camera. As a consequence it is not necessary to accumulate the opacity in a separate variable:

$$c_i = \big(1 - \alpha(x_i)\big)c(x_{i-1}) + \alpha(x_i)c(x_i) \tag{1.3}$$

When using maximum intensity projection (MIP) [WMLK89] the final color is obtained using the maximum of all values encountered when stepping along the ray.

$$c_i = max\big(c_{i-1}, c(x_i)\big) \tag{1.4}$$

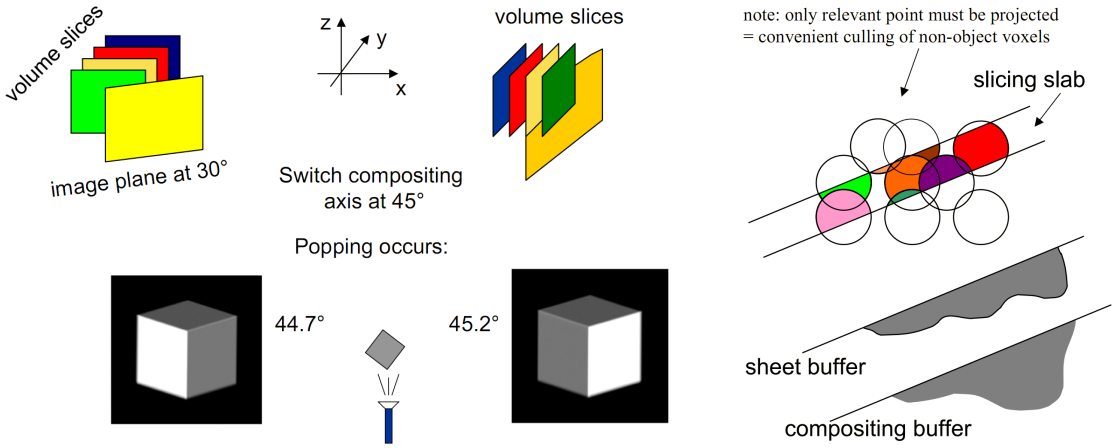
This method is independent of the direction of traversal and is frequently used, for example, for the analysis of magnetic resonance angiography (MRA) or the visualization of positron emission tomography (PET).

1.3 Volume Rendering

Volume rendering techniques can be categorized based on the method which is used for the traversal of the data and are assigned to either image-order or object-order algorithms. Object-order techniques operate in object space and project the voxels onto the image plane where the compositing is performed. As a consequence, one voxel influences several pixels around its center of projection. Image-order algorithms start from a single pixel and find the voxels which contribute its color and opacity values.

1.3.1 Splatting

Splatting is a representative of the object-order algorithms and was developed by Westover [Wes89]. It is based on the assumption that the volume consists of basis functions where each point of the grid modulates the kernel by its value, as described by Hansen and Johnson [HJ04]. When picking an arbitrary position in the volume, the corresponding result can be determined by finding the modulated kernels which contribute to the



(a) Axis-aligned sheet buffer (b) Image-aligned sheet buffer

Figure 1.1: Splatting using different compositing techniques [Mue05].

sample due to their overlap. They are then combined by evaluating the sum of their contributions. The integration of the basis function is independent of the viewing direction when assuming a radially symmetric kernel. As a consequence, it can be pre-integrated, and the result is stored as a so-called footprint in a lookup table. The footprints are splatted onto the screen and thereby cover a certain area. If the kernel itself is a Gaussian, then its footprint is a Gaussian too, and it can be calculated analytically. When using orthographic views, it is necessary to calculate the footprint function only once for a single view because it stays the same for each sample apart from an image space offset. Hansen and Johnson differentiate three different types of splatting. The composite-only method [Wes89] traverses the voxels in a front-to-back or back-to-front order and assigns colors and opacities according to the specified transfer function. After splatting the voxels onto the image plane, they are composited with the existing ones. Since the kernels have been pre-integrated, artifacts like sparkling or color bleeding may be visible in animated visualizations.

The second technique, the axis-aligned sheet-buffered method [Wes90], uses the slices of the volume which are most parallel to the image plane, as can be seen in Figure 1.1a. Instead of compositing them directly, the values are simply added to color and opacity buffers. The subsequent buffers are then composited which helps to avoid the color bleeding artifacts from the previous technique. Instead, popping artifacts may occur

when the view direction changes the order of compositing at certain angles.

The third method, the image-aligned sheet-buffered splatting approach [MC98], uses a compositing sheet which is parallel to the image plane, shown in Figure 1.1b. It shifts a slab through the volume and uses its intersections with the kernels for the projection. This has the consequence that a single voxel is visited more than only once. On the one hand, it is the most accurate method amongst the three presented techniques and avoids the bleeding and popping artifacts. On the other hand, it has the highest computational complexity because a single voxel is considered multiple times.

1.3.2 Texture Slicing

Texture slicing is an object-order technique and uses two-dimensional slices which are placed in the volume in order to apply a sampling of the data, as described in [EHK+06]. In the axis-aligned version of this algorithm (also referred to as object-aligned texture slicing) three stacks of slices are created which sample the volume along the the object's main coordinate axes (see Figure 1.2). It is necessary to use three stacks in order to avoid artifacts which would occur if the angle between the slice normal and the viewing direction converged toward 90 degrees. As a consequence, the observer would see through the gaps between the slices. Instead, the stacks are switched when the angle reaches 45 degrees. The advantages of texture slicing are its simplicity and rendering speed when implemented on graphics hardware. The drawbacks, however, are the correlation between sampling rate and the number of slices. The image quality can be improved by using a two-dimensional multitexture-based approach. Here, a trilinear interpolation is

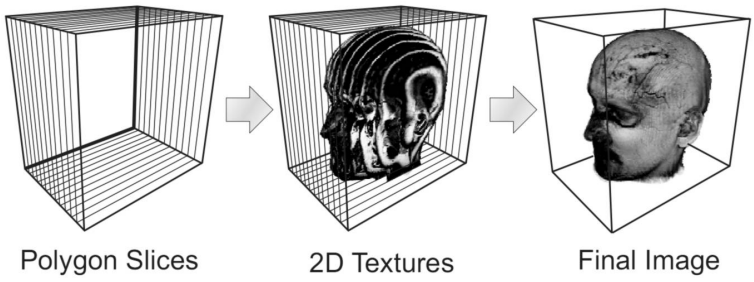
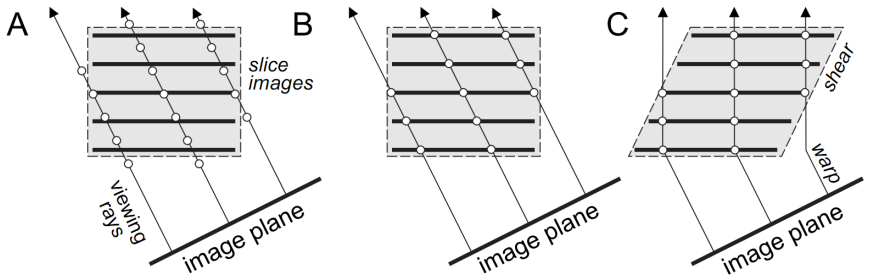


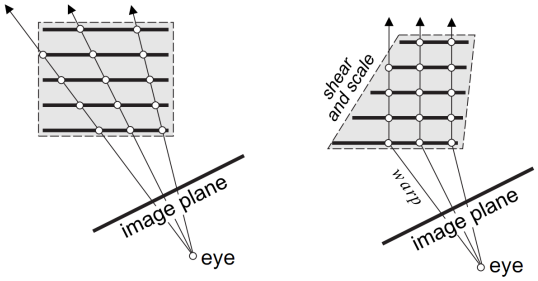
Figure 1.2: Texture slicing using axis-aligned slices [EHK+04].

achieved by using a combination of the standard bilinear interpolation of two neighboring slices and a linear interpolation between these two values. The trilinear interpolation could also be performed directly on a 3D texture. The undesired property that the sampling rate depends on the viewing angle can be compensated by making the distance between two textures dependent on the angle. Another improvement is the usage of slices which are parallel to the viewport instead of aligning them with the object's main axes. This helps to prevent the flickering artifacts which occur when switching between two different stacks.

1.3.3 Shear-Warp Volume Rendering



(a) orthographic projection



(b) perspective projection

Figure 1.3: Shear warp volume rendering [EHK+04].

Shear-Warp volume rendering, developed by Lacrout and Levoy [LL94], is another object-order technique. It is similar to texture slicing where each slice is transformed (warped) individually at first and the compositing performed afterwards. The shear-warp technique reverses this order of traversal [EHK+06]. For an orthographic projection (Figure 1.3a) it applies a shearing to the slices in the beginning and additionally

a scaling if perspective projection (Figure 1.3b) is used. An intermediate base plane is introduced additionally to the final image plane and is aligned parallel to two of the object's main axes. The slices are sheared and can be projected directly onto this base plane. This projection corresponds to the resampling of a two-dimensional image because the slices and the base plane are parallel. When the compositing of the slices has been finished, the image is transformed accordingly. The advantage of this algorithm is its performance when implemented on a CPU using run-length encoding and early ray termination. Due to the non-uniform access patterns it is not the first choice when an implementation on a GPU is required.

1.3.4 Ray Casting

Image-order techniques, like ray casting introduced by Levoy [Lev88], use the pixels of the image plane as starting position for sampling the volume (see Figure 1.4). During ray traversal the volume rendering integral is repeatedly evaluated for each sampling position using equations 1.2. As Weiskopf [Wei07] states, it is common to use uniform Cartesian grids which are sampled at equidistant steps using trilinear interpolation, typically in a front-to-back compositing scheme. At the beginning of the traversal, the initial sampling position has to be determined. This is done by calculating the first intersection of the viewing ray with the volume boundary. Additionally, the direction of the ray is needed for the subsequent sampling steps. The second intersection of the ray with the volume boundary marks the stopping position of the traversal.

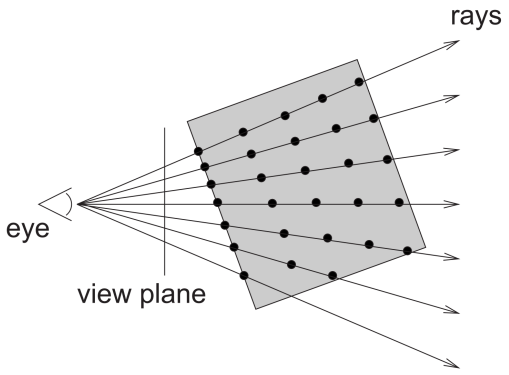


Figure 1.4: Ray casting [EHK+04].

There are several optimization techniques for speeding up this process, for example early ray termination. Additionally to the inside/outside test, the traversal can also be stopped when the opacity has reached its maximum. Subsequent sampling positions would no longer contribute to the result when using front-to-back compositing and are therefore ignored. Another method called empty-space skipping or empty-space leaping by Krüger and Westermann [KW03] completely leaves out empty regions. The authors use an octree structure to store information about the child nodes like the minimum and maximum scalar values. Additionally, a two-dimensional texture which is accessed using pairs of minimum and maximum values is created. It indicates the existence of at least one entry in the subvolume corresponding to the octree cell that will be assigned an opaque color value after the application of the transfer function. The ray is traversed using a coarse resolution as long as the two-dimensional texture, which is accessed using pairs of min/max values, indicates empty space. When a non-empty region is detected, the step size is decreased. Apart from these two optimization techniques, there are several other strategies for increasing the rendering speed which are described by Engel et al. [EHK+06].

The inherent parallelism of the ray casting algorithm due to its independent viewing rays makes it the ideal candidate for the implementation on a GPU. This is done by assigning the operations of a single ray to one of the multiple, parallel processing pipelines of the graphics card. Additionally to the speed gained by parallelizing the ray traversal, the access time of the data is decreased by making use of the fast graphics memory for storing the volume.

1.3.5 GPU-based Volume Rendering

The progress in the development of consumer graphics hardware enabled high quality volume visualizations at interactive frame rates. The advantage of the execution on the graphics card is the high degree of parallelization which cannot be achieved even with today's multi-core CPUs. In 3D graphics polygonal models are used for the visualization of virtual objects and scenes. The sequence of individual steps required for the conversion from polygons to a displayable raster image is called the graphics pipeline. Initially, the graphics cards were based on a fixed-function pipeline which could not be

altered, but over time they became more flexible, configurable and programmable. The pipeline consists of several stages but can be reduced to three stages for explanatory purposes [EHK+04]:

Geometry Stage

- operations like translation, rotation or scaling are performed on the individual vertices in the local object coordinate system and in world space
- per-vertex lighting is calculated based on the specified light sources and material properties
- the scene is transformed from world space to view space of the specified camera
- the projection from view space into the two-dimensional screen space is performed

Rasterization Stage

- geometric primitives like points, lines and polygons are disassembled into a raster image representation where each resulting fragment corresponds to a single pixel
- colors and texture coordinates are interpolated for each fragment

Fragment Stage

- the information of each fragment like color, transparency or depth information is used for the calculation of the final pixel color

With the introduction of programmable shaders in 2001 [LKM01], there are at least two types of programs which can be executed on the graphics card: Vertex programs apply transformations described in the geometry stage, whereas fragment programs process the rasterized result. Volume rendering techniques operate directly on the scalar values without creating intermediate geometric representations of the data as done in surface rendering algorithms. Although there is no polygonal model, the vertex program is required for the setup of a proxy geometry. It is used for the transformation of the data

set and serves as a basis for the compositing stage which is performed in the fragment program.

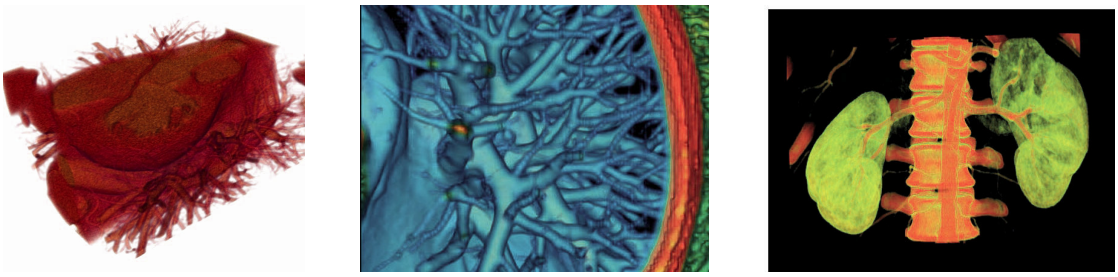
According to Engel et al. [EHK+04] there are several reasons for the success of slice-based volume rendering techniques implemented on a GPU: The graphics card's high rasterization performance, the fast transfer of data from the texture memory to the rasterization unit, the built-in interpolation methods for the sampling of the data, and the simplicity of the implementation. The authors also state that output sensitivity should be the most important aspect for algorithms in the field of computer graphics. Therefore, the main disadvantage of slice-based techniques is the direct relationship between the number of slices and the complexity of the volume data. According to Krüger and Westermann [KW03], only a few percent of all fragments of the examined data sets contribute to the final image. Since many volume rendering techniques focus on the visualization of borders, the majority of the voxels is set to transparent and therefore not required for the generation of the final image. Further disadvantages of slice-based algorithms are their dependency on the rasterization performance, their inflexibility, and their limited suitability for acceleration techniques. As a consequence, ray casting is the preferred volume visualization technique in this thesis because of its flexibility and performance.

2 Transfer Functions

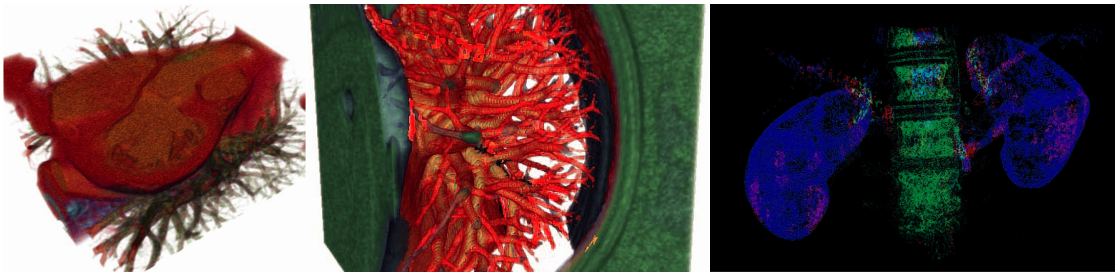
A volumetric data set, for example obtained from a CT scan, contains density values only, but it does not store information about the materials contained within, let alone their colors and opacities. For the visualization it is essential though to have a mapping from intensities to optical attributes. As a consequence, it is up to the user to perform the classification which means the identification of certain features. As transfer functions evolved, this process is no longer limited to the specification of certain density values. The transfer function design has moved from the one-dimensional domain of density values to more abstract, multi-dimensional representations of the data. As a result, structures of interest within a data set can be visualized in a higher quality. In most cases, the process still requires the user to have knowledge about the data and the structures contained within, as well as experience with the setup of a transfer function in general. The following sections will give an overview of the different approaches and also shows some example visualizations.

2.1 Transfer Functions Based on Statistical Properties

Caban and Rheingans [CR08] introduce a technique which is based on texture analysis by combining first-, second- and high-order statistics. This helps to discriminate structures even when they have similar density values. The authors capture structural and geometrical properties by using histogram statistics, co-occurrence and run-length matrices. First, a data set is divided into overlapping subvolumes which are then analyzed



(a) density-based (b) gradient-based (c) opacity- & gradient-based.



(d) size-based (e) size-based ($\rho = 2$ $\sigma = 4$ $k = 3$) (f) size-based ($\rho = 2$ $\sigma = 4$ $k = 3$)

Figure 2.1: Comparison of visualizations created with traditional methods (top row) and with texture based transfer functions (bottom row) [CR08].

separately. These partitions have to be large enough to capture textural statistical properties and small enough to include only local features of the surrounding voxels. There exist several properties which can be analyzed, for example, the probability of the occurrence of a certain intensity value inside one of these subvolumes. Other local features which can be extracted by first-order statistics are mean, variance, kurtosis, skewness and deviations. Second-order statistics capture the likelihood of observing two different intensity values at a certain distance and are calculated by using co-occurrence matrices. These matrices are created for different angles and are then averaged in order to provide rotation-invariant properties such as energy, inertia, entropy, etc. Regions with similar properties are determined by using run-length matrices of high-order statistics. Such a matrix holds the frequency of a certain number of points with a specific intensity in one direction. It offers the possibility of analyzing the amount of short runs, long runs and the uniformity of a run. All in all, there are 20 textural metrics which get pre-computed and combined into a multi-dimensional vector.

The transfer function is set up by specifying the number of different structures within

the data set. With this input an inequality-based fast k-means algorithm is used to detect clusters which are then used for setting up the transfer function. It is also possible to manually define thresholds for individual metrics. Another way for rendering certain structures is to select parts of them with the help of a three-dimensional widget. The descriptors contained within this region are then combined and averaged to form a new descriptor. The distance between the textural properties of the voxel and the new descriptor are examined during the rendering phase. If the value is below a certain threshold, then a different color and opacity is assigned to the voxel under consideration. In order to separate two different structures, parts of them have to be selected with a three-dimensional widget. If the corresponding metrics are similar, then they are assigned lower weights. Otherwise, they are emphasized by using higher weights. For a 128^3 data set with a region size $\rho = 2$ voxels and an overlap $\sigma = 4$ voxels the pre-processing part of the algorithm takes about 30 seconds and the clustering approximately 6 seconds. Figure 2.1 shows some examples in comparison to traditional methods. The very recent approach of moment curves by Patel et al. [PHBG09] is also based on statistical properties of a data set for visualizing distinct features with similar density values. Instead of dividing the data set into static subvolumes, they use dynamically changing neighborhoods. By doing so, they try to find optimal sets of voxels for the statistical analysis. For each voxel the neighboring density values inside a sphere with an initial radius r are used for calculating statistical properties, and the procedure is repeated for increasing radii. The resulting triples of radius, mean and variance can be interpreted as coordinates of a three-dimensional feature space and combined to form a curve, the moment curve. After evaluating this procedure for every voxel of the data set, materials can be classified by using specific groups of moment curves. These groups can be obtained through brushing in the visualization of the curves, as shown in Figure 2.2.

2.2 Curvature-Based Transfer Functions

Kindlmann et al. [KWTM03] use the curvature of structures contained in a volumetric data set as a criterion for the transfer function design and thereby continue the work of Hladůvka et al. [HKG00]. They use an algebraic framework of differential invariants

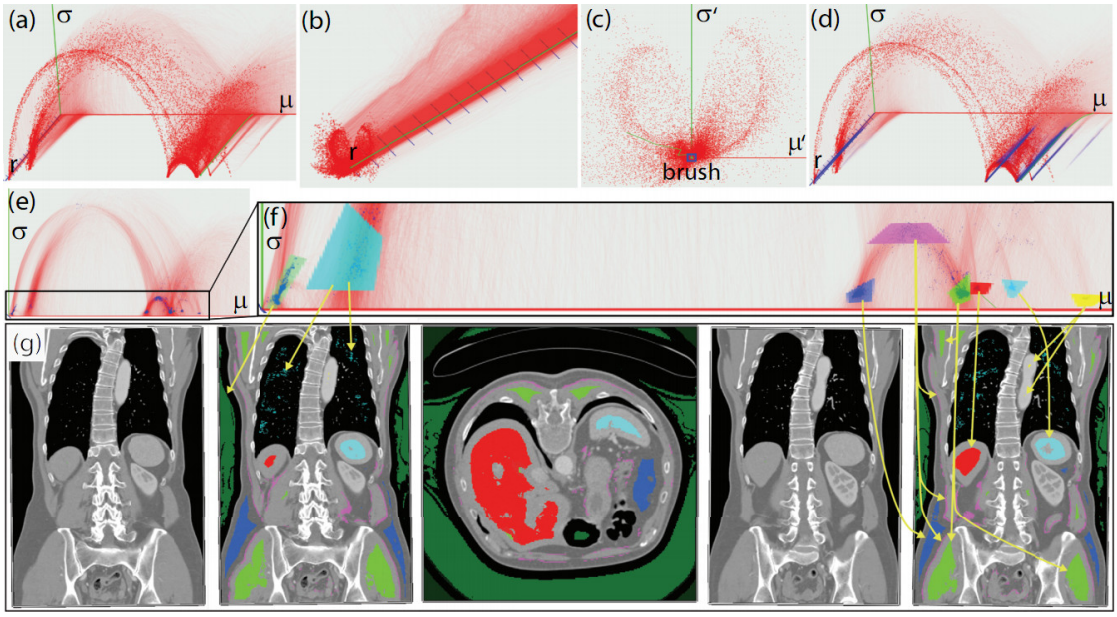


Figure 2.2: Transfer function setup and visualizations using moment curves [PHBG09].

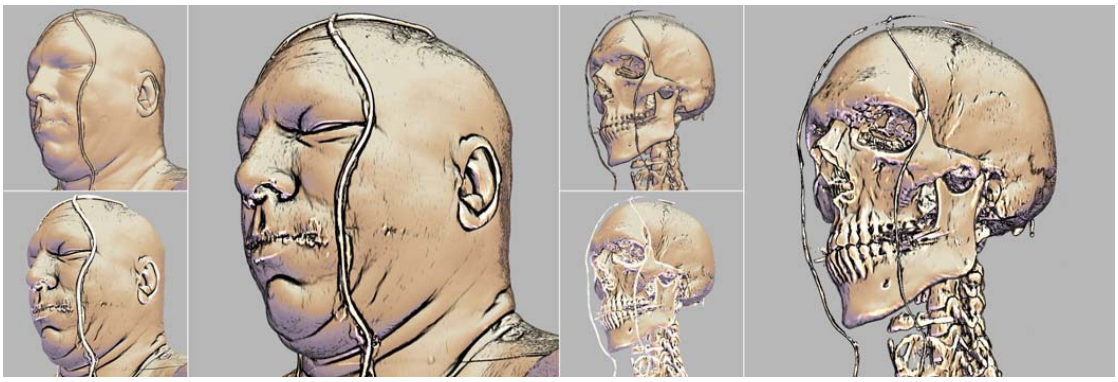


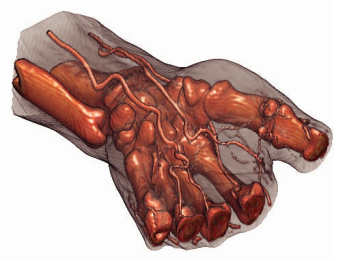
Figure 2.3: Visualizations using curvature-based transfer functions for emphasizing the silhouette and smoothing of the surface [KWTM03].

for calculating the curvature. One difficulty they are facing is the increasing amount of noise when using higher order derivatives. A convolution based measurement scheme enables them to determine the ideal tradeoff between accuracy, continuity, and filter size. The curvature information is then used for creating non-photorealistic renderings. The disadvantage of previous approaches for the emphasis of the silhouette of an object was that the visible width of the contour was not constant. The authors solved this problem by incorporating curvature into the transfer function. Isotropic surface smoothing is achieved through the minimization of the surface integral of total curvature. This technique is similar to the blurring of an image with a Gaussian filter. Noise can be removed with a variant of this energy function while preserving important features. A visualization created with these techniques is shown in Figure 2.3.

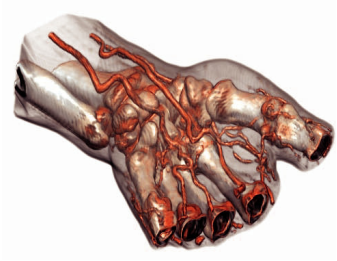
2.3 Size-Based Transfer Functions

Another approach for separating structures which have similar or the same intensity values is to use a transfer function which is based on the relative size of a feature. Correa and Ma [CM08] have developed a technique that uses scale fields. In such a field every voxel represents the local scale or size of the feature which contains this voxel. Scale-space theory has its origins in computer vision but it is also used for diffusion-based smoothing of three-dimensional volumes. Previous approaches to multi-scale analysis have used pyramid representations to enhance the classification which leads to discrete and disperse representations of scale. Another drawback is that they are hardly capable of finding small variations in size. These disadvantages can be avoided when computing scale fields which are based on continuous scale-space theory in combination with a set of scale detection filters.

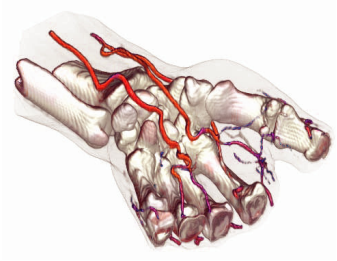
The first step of the procedure is the computation of the scale field. This could be achieved by using a convolution, which is quite expensive from a computational complexity point of view. Instead, the approach of forward Euler integration of the diffusion equation is the preferred technique. The minimum and maximum detectable sizes can be specified by the user as parameters for the diffusion process. The scale space can then be used, for example, to detect blobs at multiple scales by searching for maxima of



(a) 1D based on intensity:
scalar value → color
scalar value → opacity



(b) 2D based on intensity and gradient.



(c) Size-based:
size → color
scalar value & size → opacity

Figure 2.4: Comparison of traditional and size-based transfer functions for an unsegmented hand data set [CM08].

the normalized second derivative. It can happen that two distinct objects merge into a single blob in the diffusion process before they can be separated into different objects. Therefore, the signed forward differences are used as an approximation of the gradient. By doing so, a filter can be created which focuses on regions of a certain homogeneity and prevents diffusion over edges. Similar results could be achieved by a progressive application of smoothing and erosion filters. The result of the scale detection procedure is a set of discrete points which represent the most salient scales of the volume. In order to create a smooth transfer function, it is necessary to find a continuous representation for these points. Therefore it is assumed that all voxels within a certain radius t of an extremum in both space and scale can be described with a size t. The continuous representation can then be achieved with the help of scattered data interpolation using Shepard's interpolation. Since the scale-space is based on diffusion, it is inherently robust to noise. Features that have different intensities but similar sizes are represented similarly due to the usage of the Laplacian.

The scale field can now be used to create a transfer function by combining it with the original data. A comparison of traditional methods and the size-based approach can be seen in Figure 2.4. Vessels and bones cannot be separated very well with a transfer function based on density values only. The results slightly improve when a two-dimensional transfer function based on densities and gradients is used. With the help of the size based transfer functions, thin structures such as veins can be emphasized while hiding other structures like skin.

2.4 Transfer Functions Based on Semantic Models

With the increasing complexity of the transfer function itself, it gets more difficult to create good visualizations in a short time. Even for experienced users the result of modifying certain parameters is not always predictable, and the transfer function setup corresponds to a trial and error process. In order to provide an interface which can be used in a more direct way to visualize certain features of a volume, Rezk-Salama et al. [RSKK06] introduced a layer of abstraction into this process. Figure 2.5 shows an example using a single slider for the adjustment of the visibility of the vessels and the effect its change has on the visualization. This approach allows the creation of a semantic model which maps semantic parameters to instances of transfer functions based on the requirements of the user. Therefore, the structures of interest in the volumetric data sets, called entities, have to be defined in advance. As the implementation is designed for a specific field of application, e.g. medicine, the number of entities, such as skin or bone, is limited. The first step is the adjustment of a transfer function template for a certain number of training data sets. The parameter vector of each individual transfer function is then used as input for a principal component analysis (PCA). The result of this operation, a set of semantic parameters, is adapted according to the individual requirements of the visualization. The authors state that in practice the first principal component is dominant and can be used for editing the transfer function with only one parameter. It is also possible to introduce additional semantics which do not control the selection of the entities but properties like the contrast between them or their visibility. This is done by adapting the training data sets using the basic parameter. The first principal component from the difference of the parameter vectors is used as a semantic parameter for the contrast.

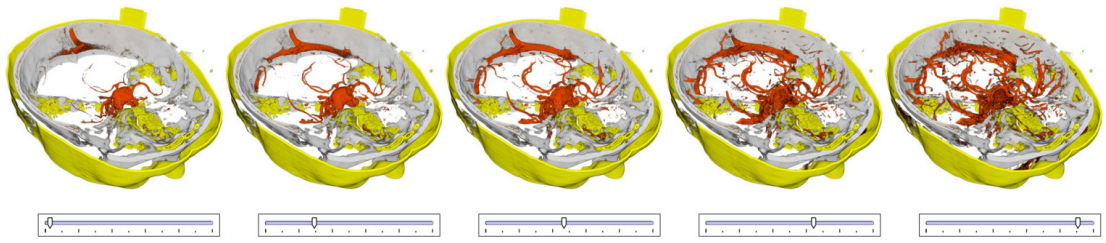


Figure 2.5: The visibility of blood vessels is controlled by a single parameter using a transfer function based on semantic models [RSKK06].

2.5 Transfer Functions Based on Derivatives

Instead of using a one-dimensional transfer function space that is based on the density values only, Kindlmann [Kin99] creates a two-dimensional space which incorporates the gradient magnitude as the second dimension. By doing so, material boundaries in a data set can be interpreted as arches in a new transfer function domain, as can be seen in Figure 2.6. Both ends of an arch correspond to homogeneous regions of two different materials within a data set that have a gradient magnitude of zero. As one ascends an arch on the left side, the corresponding position inside the data set is moved along the gradient direction. The top of the arch corresponds to the border between the two materials where the gradient magnitude is at its maximum. The descent of the arch stops on the other side in the homogeneous region of the second material because its gradient magnitude has become zero. A selection of specific materials and boundaries in this transfer function domain can be made using certain transfer function widgets. Basically these widgets are polygonal shapes which can be modified in order to fit to the arches, especially the regions on top with a high gradient magnitude.

An inherent problem of transfer functions based on density values and gradient magnitudes is that arches can overlap, as can be seen in Figure 2.8a. A possible approach for circumventing this effect is to incorporate the second directional derivative along the gradient direction as a third dimension into the transfer function design, as done by Kniss et al. [KKH01].

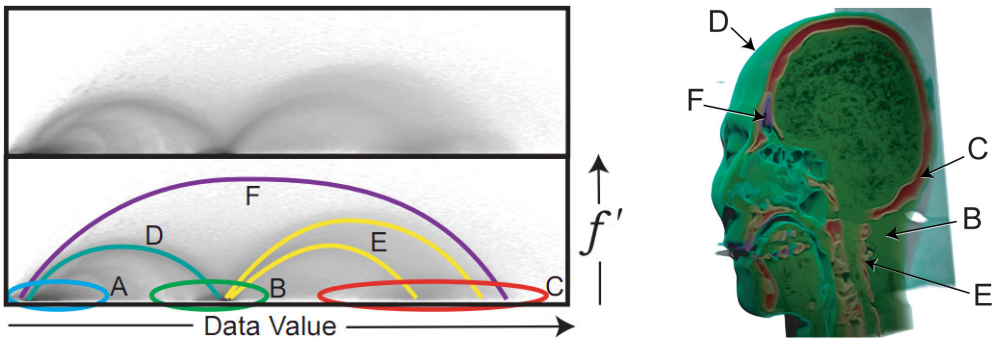
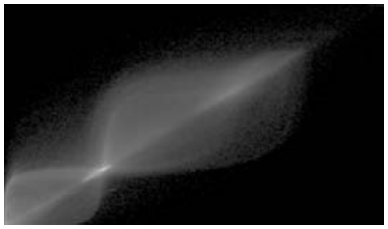
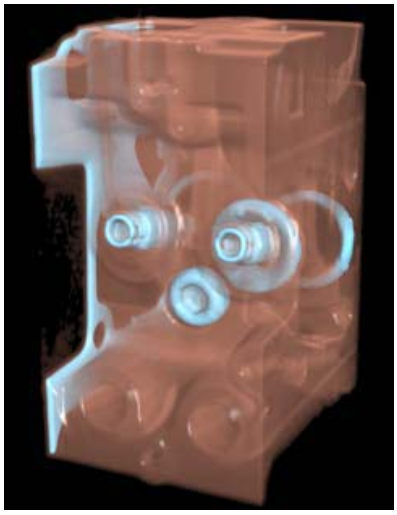


Figure 2.6: The rendering of the Chapel Hill CT Head data set is the result of a transfer function based on a 2D histogram of density values and gradient magnitudes [KKH02].



(a)

(b)



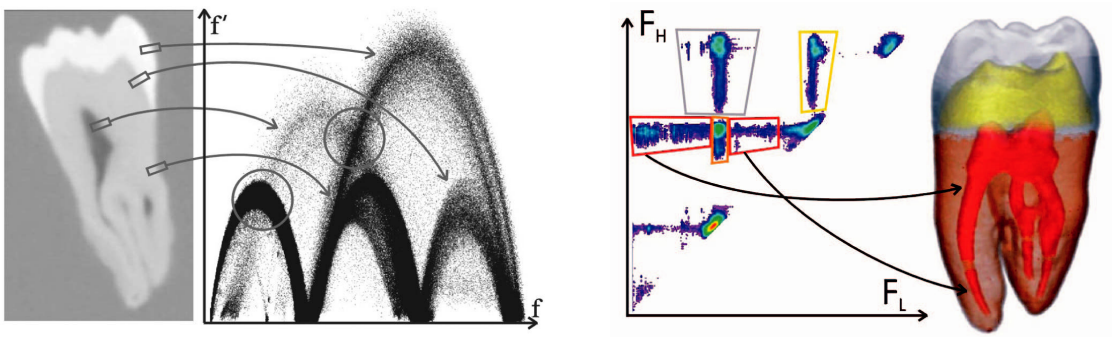
(c)

Figure 2.7: Instead of using the standard histogram consisting of values obtained from sampling the positive and negative gradient direction (a), parallel axes (b) serve as the domain for the transfer function design to create visualizations (c) [LM04].

The technique of using the two-dimensional transfer function domain of density values and gradient magnitudes still does not solve the problems of noise, partial volume effect and bias along the boundary. These influence factors result in a blurred histogram with multiple shifted or scaled copies of a single arch. An approach inspired by the technique of the arches was developed by Lum and Ma [LM04]. They pick the two density values on both sides of the border by using the gradient for taking the samples. The basic idea is that these two values will stay constant for a single border and therefore can be used to set up a transfer function. The distance between those two sample positions has to be defined individually for each data set because the width of the border varies with the amount of blur of the data. Lum and Ma state that a distance of one voxel in each direction would be sufficient. The setup of a transfer function is not as intuitive as for the two-dimensional space consisting of density values and gradient magnitudes. Plotting the density on two separate axes produces a histogram shown in Figure 2.7a. Therefore, a different representation was created which uses two parallel axes for the density values. The intensities obtained from sampling the positive gradient direction are plotted on the bottom axis whereas the ones that originate from the other side of the material boundary are plotted on the top axis. The value pairs are then connected by a

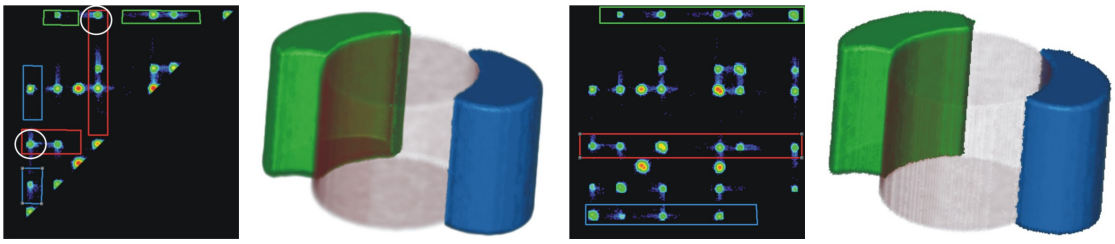
line where the brightness corresponds to the number of pairs with the same combination of scalar values, as can be seen in Figure 2.7b. The diagonals correspond to boundary regions whereas vertical lines represent the homogeneous regions of the data set. A transfer function can be created by simply brushing the diagonals.

Sereda et al. [SBSG06] also aimed at finding the materials located at both ends of the arches. They have developed a new transfer function space based on LH histograms which have been introduced by Serlie et al. [STF+03]. Instead of trying to find the density values of the two materials which form the border by taking samples at a fixed width, they integrate the gradient field in both the positive and the negative gradient direction. The integration continues as long as the gradient magnitude is above zero. This procedure is executed for every voxel of the data set, and the resulting low (F_L) and high (F_H) density values are used for constructing the two-dimensional histogram. By definition, F_H will always be greater than or equal to F_L. This has the consequence that only the region above the diagonal of the histogram will be populated with values. For voxels with a gradient magnitude of zero, the values for F_L and F_H can be set to the voxel's intensity because it is then inside a homogeneous material. In this case the value projects onto the diagonal of the LH histogram. When comparing this type of transfer function to those based on density values and gradient magnitudes, it turns out that the different boundary regions can be separated more clearly. There are less ambiguous regions due to the overlapping of the arches, as can be seen in Figure 2.8.



(a) (b)

Figure 2.8: Boundaries highlighted in the slice view project onto the marked positions in the two-dimensional histogram based on density values and gradient magnitudes (a). A visualization using LH histograms is shown in (b) [SBSG06].



(a) (b) (c) (d)

Figure 2.9: Visualizations of an artificial data set using standard LH histograms (a) and (b) and mirrored LH histograms (c) and (d) [SVG06b].

It is possible that different objects in a volumetric data set which are separated in space have similar properties and therefore cannot be distinguished by a transfer function. In such a case, the technique of region growing is a good means for discriminating these materials. Starting at a seed position, the current voxel is labeled, and the neighbors are examined. According to a certain cost function, the most similar one is selected, labeled, and then the procedure is repeated for the other unlabeled, neighboring voxels. This cost function is designed not to grow outside the current boundary, into another boundary, into areas of constant intensities or into small noisy boundaries. Therefore, several measures are incorporated into the calculation. The first criterion is the object distance which represents the similarity to those voxels which have already been labeled. It is calculated by using the Euclidean distance in the LH space. The neighbor distance is used as the second criterion and comes into play when dealing with biased or thinning boundaries. The third criterion is the neighbor coherence. It evaluates the directional coherence of the gradients and is scaled by the average strength of the neighbors. The boundary strength, which is used as the last criterion, facilitates the growth of strong boundaries with higher values for the distance between F_L and F_H. After calculating these measures, they are combined by using certain coefficients which need to be adapted accordingly for each data set.

One difficulty which arises with LH histograms is that two neighboring materials share the same boundary. Therefore, an unambiguous selection is not possible. Figure 2.9b shows a visualization where it can be seen that part of the green material boundary is missing because it belongs to the red, translucent material too. Based on their previous work, Sereda et al. have developed a modification of their LH histograms: the mirrored

LH histograms [SVG06b]. The boundary is divided into two separate parts by estimating the exact position of the edge during the integration of the gradient field. This can be done with the help of the second derivative by assuming that it is located at the zero crossings. The histogram is then created as follows: If the intensity of the voxel is greater than the one of the voxel which is located at the edge, then it projects above the diagonal as it is done in the standard LH histogram. If the voxel's intensity is lower, then it projects below the diagonal which is done by exchanging F_L and F_H. The histogram region below the boundary looks like a mirroring of the standard LH histogram because it contains approximately the same number of voxels as the region above the diagonal. Partial boundaries which belong to the same material are horizontally aligned and can be selected more easily. It is also possible to project the histogram values onto the F_H axis to create a one-dimensional histogram. This makes it easier to select the homogeneous regions and the boundary regions of a specific material at once. In the visualization shown in Figure 2.9d the hole in the boundary of the green material has disappeared.

2.6 Transfer Functions Based on Clustering

Tzeng and Ma [TM04] present a method for creating a transfer function based on material classes which are automatically extracted using the Iterative Self-Organizing Data Analysis Technique (ISODATA). Instead of applying the algorithm to the entire volume, only a subset of randomly chosen voxels is used as input to reduce the required processing time. Features used for the clustering are intensity, gradient magnitude, second directional derivative and neighboring values. The algorithm itself is an adapted version of the k-means clustering where the user has to specify the number of clusters in advance. ISODATA additionally merges clusters when the number of classes grows too large or when the clusters are too close to each other. Splitting is performed when there are too few clusters or when the samples contained within a cluster are very dissimilar. Because only a part of the entire data set has been classified, the remaining voxels have to be processed based on the classified ones. This is done by using the minimal distance between the feature vector of a voxel and the mean vectors of the clusters.

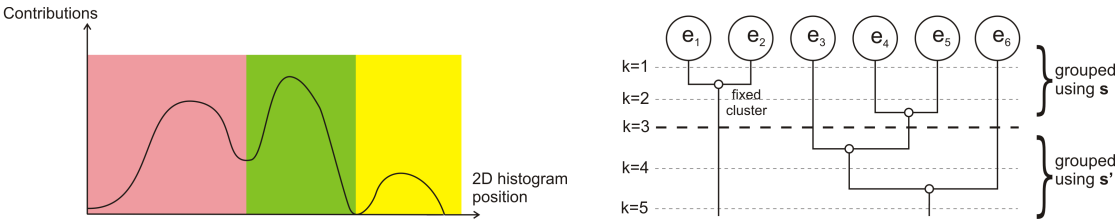


Figure 2.10: The contributions of a single row of an example LH histogram can be seen in (a). The initial clusters (colored regions) are determined through local extrema. In (b) a dendogram is shown which visualizes the interactive clustering procedure using different similarity measures. [SVG06a]

The result is then mapped to a two-dimensional transfer function based on density values and gradient magnitudes. This visualization is created to provide an interface for the refinement of the clustering and the manipulation of the colors and opacities assigned to the different material classes.

Based on their previous work about LH histograms, Sereda et al. [SVG06a] aimed at automatically creating transfer functions with the help of clustering techniques. They have developed an approach which consists of two similarity measures: The first one is supposed to group similar boundaries by detecting clusters in the LH feature space, whereas the second one evaluates the spatial connectivity of these clusters. It is necessary to downsample the histogram data before the clustering due to the complexity of the generation of the hierarchy, which is $O(\frac{n^3}{2})$ where n is the number of clusters. Another preprocessing step applied to the histogram, is the blurring of the data with a two-dimensional Gaussian kernel. The value for σ needs to be as small as the size of a histogram bin. This eliminates small clusters which exist, for example, due to the presence of noise in the data. Another effect of the filtering is the establishment of a direct neighborhood relation between separated peaks. The initial clustering can now be calculated by detecting local peaks, as can be seen in Figure 2.10a. The detected peaks correspond to the material boundaries inside the volumetric data set. With this information at hand, several similarity criteria for joining the clusters are established. The first one is simply the distance between two clusters. It is based on the assumption that elements which are close to each other in the LH histogram, represent boundaries

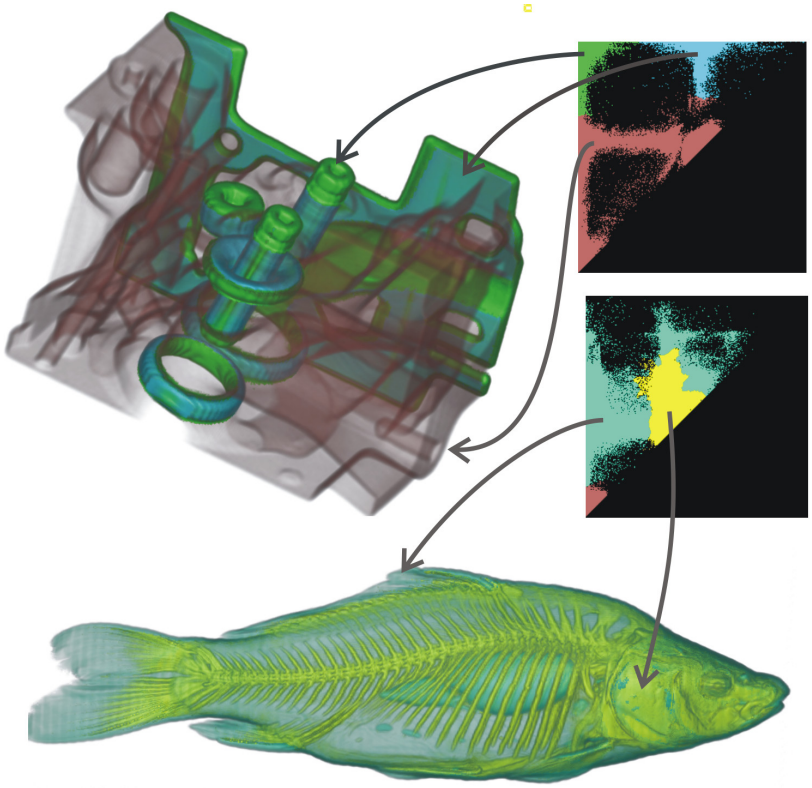


Figure 2.11: Visualizations of the engine and the carp data set based on the clustering of the LH histograms [SVG06a].

between similar tissues. Using the separation as a second criterion, helps to determine if two peaks correspond to different boundaries. This is done by looking at the depth of the valley between these peaks. The direction of elongation is used as the last criterion. It originates from the fact that clusters in the LH histogram are not necessarily compact and round. Instead, it is possible that the tissues which form the boundaries have varying density values and therefore lead to asymmetric clusters.

Now that the criteria have been determined, they are combined by using a technique which is based on the Bayesian decision theory. As a prerequisite the clusters are interpreted as bivariate two-dimensional probability density functions (PDF). When using a Bayesian classifier, the upper bound of the probability of a wrong decision can be estimated using the Bhattacharyya bound. This technique can also be used for interpreting this probability as the overlap of the PDFs and thereby as the similarity of the

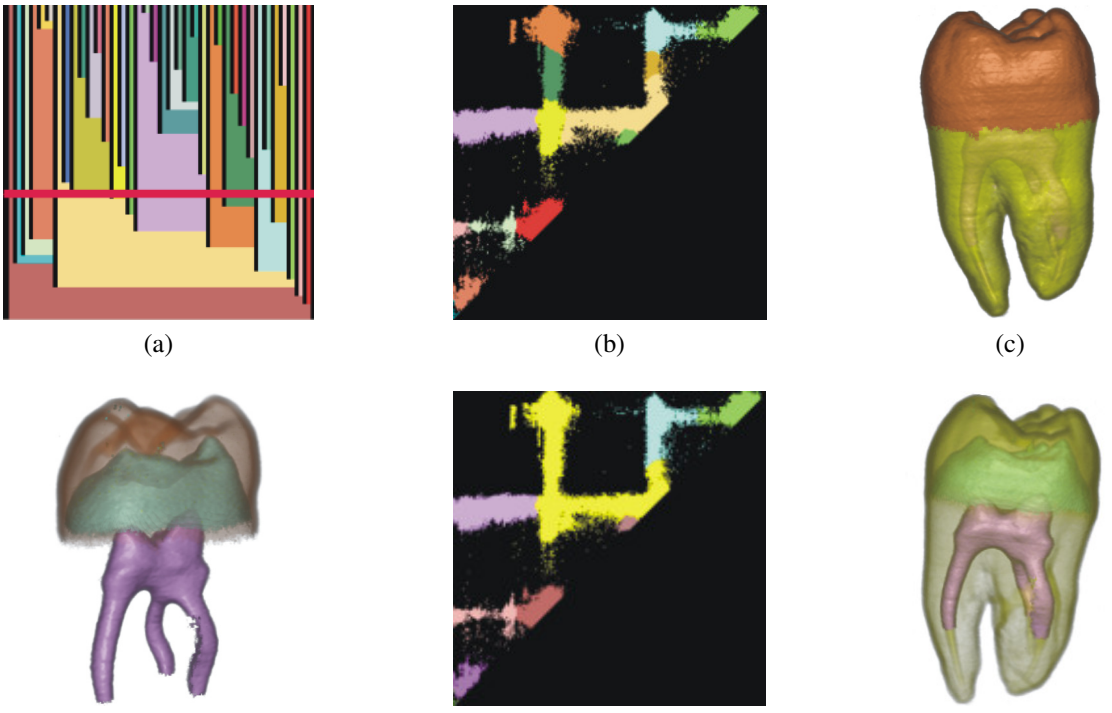


Figure 2.12: Visualizations of the tooth data set based on the clustering of the LH histograms [SVG06a]. The selection of the hierarchy level is indicated by a red line in the dendogram in (a). The corresponding selection of the clusters is shown in (b), whereas (c) is rendered with the resulting transfer function. It has been adapted manually in order to remove the cylindrical boundary and to fix the (yellow) dentine-air border. The visualization (d) is created by making the outside boundaries of the tooth semi-transparent. The grouping of several clusters in (e) leads to the rendering shown in (f).

clusters. The second similarity measure is supposed to agglomerate clusters corresponding to boundaries which are connected in the volumetric data set. This is achieved by calculating the number of direct neighborhood relations between the clusters. For a 3^3 neighborhood the relations between the voxels are weighted using the directional coherence [SBSG06] and normalized to make it independent of the cluster size.

The authors state their framework enables a real-time interaction with the cluster hierarchy and offers the possibility of selecting objects at different hierarchy levels, as shown in Figure 2.10b. In this example the clusters e_1 and e_2 are fixed at level 2, taken out of the hierarchy and are therefore no longer split or joined with other clusters when

changing the hierarchy level. The other clusters are grouped using a similarity measure s at the initial level. At level 3 this measure is modified and the grouping of the clusters is performed with the new measure s'. Figure 2.11 and Figure 2.12 show visualizations using the described technique.

3 Clustering

Kogan et al. define the task of clustering as follows: “Clustering techniques are used to discover natural groups in data sets and to identify abstract structures that might reside there, without having any background knowledge of the characteristics of the data.” ([KNT06, p. VII]) or as Das et al. describe it: “Cluster analysis means the organization of an unlabeled collection of objects (or patterns) into separate groups based on their similarity. Each valid group, called a “cluster”, should consist of objects that are similar among themselves and dissimilar to objects of other groups.” ([DAK09, p. VII])

From the point of view of machine learning, clustering could be interpreted as the process of unsupervised learning of a hidden data concept. A huge number of algorithms have been developed in the past which approach the task form different domains such as graph theory [Zah71], statistics [For65], fuzzy set theory [Jai00], evolutionary computing [Bon03] [Fal98] or neural networks [Koh95] [MJ95] [PBT93].

According to Kogan et al. [KNT06], it depends on the field of application which characteristics of the algorithms are the most important ones:

- type of attributes
- scalability to large data sets
- ability to work with high-dimensional data
- ability to find clusters of irregular shape
- handling of outliers
- time complexity
- data order dependency
- type of labeling (strict/fuzzy)
- dependence on user-defined parameters and a priori knowledge

The following sections will give an overview of different classical approaches to the task of clustering and analyze some of the above mentioned properties.

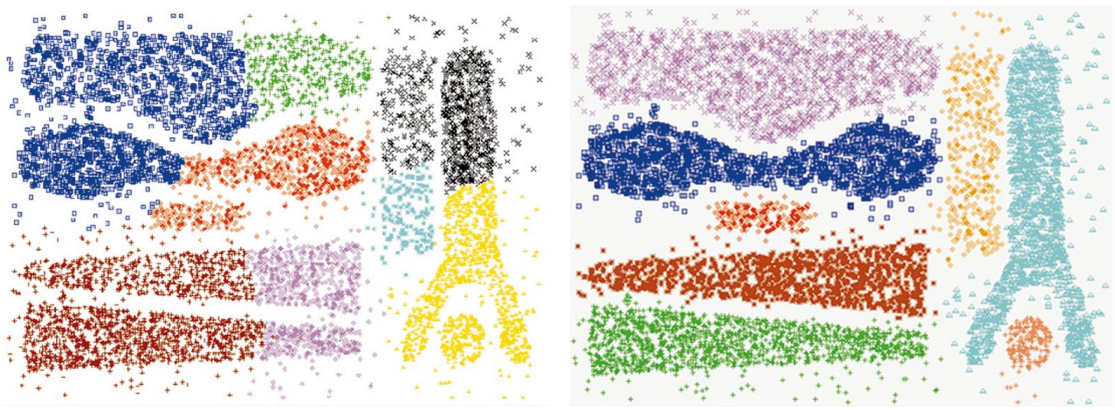
3.1 Hierarchical Clustering

In general, there are two types of hierarchical clustering algorithms: agglomerative and divisive. The agglomerative variant starts at the lowest hierarchy level where each cluster contains only a single object. When ascending the hierarchy, the number of clusters decreases at each step by fusing them. The divisive variant starts at the highest level where there is only one clusters which contains all objects and subsequently splits these clusters into smaller groups. This procedure can be visualized with a special kind of tree diagram, a so called dendogram, which can be seen in Figure 2.10b.

There are three linkage or distance metrics used to determine which clusters are fused:

- **Single-Link** uses the shortest distance between any member of the first and the second cluster, also referred to as connectedness or minimum method (e.g. SLINK [Sib73]) and produces elongated clusters.
- **Complete-Link** uses the greatest distance between any member of the first and the second cluster, also referred to as diameter or maximum method (e.g. CLINK [Def77]) and creates compact clusters.
- **Average-Link** uses the average distance between any member of the first and the second cluster (e.g. Voorhees' method [Voo86]) and results in cluster shapes somewhere between the single-link and the complete-link method.

One of the most popular hierarchical clustering algorithms is the "Balanced Iterative Reducing and Clustering using Hierarchies" (BIRCH) [ZRL96] [ZRL97]. It is based on a height-balanced tree, called Cluster Feature tree (CF-tree). This representation of the data is created by accumulating its zero, first, and second moments. When a new data point is added, it contributes to the closest CF leaf if the maximum number of elements for this leaf has not been reached. The statistics for the leaf and all its parent nodes up to the root are updated. If there is no space left, then a new CF is created. There are two important variables which determine the output of the clustering: The



(a) (b)

Figure 3.1: Comparison of the CURE (a) and the CHAMELEON (b) algorithm applied to a data set with 8000 points [KHK99].

branching factor limits the number of allowed children per node, whereas a dynamically updated threshold controls whether a point is assigned to an existing leaf or to a new one. The algorithm can be executed several times in order to improve the results of the previous runs. BIRCH is the first algorithm which can handle noise effectively and has a complexity of $O(N)$. Since the number of points a node is able to store is limited, the result does not always correspond to the actual clusters. Another drawback is that different orders of the input data lead to different clusters.

Linkage metrics tend to produce clusters with convex shapes since they are based on the Euclidean distance. Therefore, Guha et al. [GRS98] have developed an algorithm, "Clustering Using REpresentatives" (CURE), that has several advantages. It is an approach positioned somewhere between graph methods, which use all points and geometric methods, which use a single centroid. A cluster is represented by a certain number of points in its surroundings, and the single- and average-link metrics are replaced by the representatives' aggregate closeness. As a consequence, non-spherical shapes can be created and the impact of outliers is reduced. Although the complexity remains $O(N^2)$, the performance increases because N now refers to the number of samples.

There are also other hierarchical algorithms such as "RObust Clustering using linKs" (ROCK) by Guha et al. [GRS00] which is not used for numerical attributes such as CURE but for categorical attributes. A different technique from Karypis et al. [KHK99]

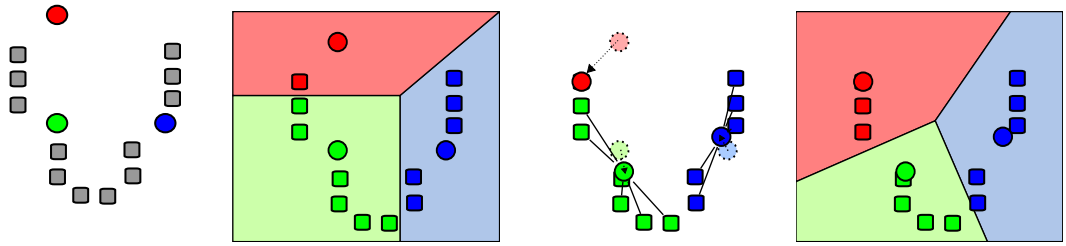


Figure 3.2: Example of the k-means clustering algorithm [Wik09]. The colored circles in (a) indicate the initial, randomly chosen centroids. In the first iteration shown in (b) each point is assigned to the nearest centroid. In (c) the mean values are updated and the centroid updated accordingly. Executing (b) and (c) until convergence results in the final clustering shown in (d).

called CHAMELEON uses a sparse-graph representation of the items and is based on the k-nearest-neighbor graph approach. Edges exist between two vertices only if they are both amongst their k most similar points. The clustering process consists of two stages. The output of the first stage are small partitions which are subsequently merged in the agglomerative process using relative interconnectivity and relative closeness as similarity measures. A comparison of the results obtained from CURE and CHAMELEON can be seen in Figure 3.1.

3.2 Partitional Clustering

Partitional clustering algorithms start with an initial partitioning of the data set and iteratively refine the classification by minimizing a certain dissimilarity measure within these clusters. The procedure stops when the position of the centroid no longer significantly changes, the quantization error is small enough, or the maximum number of iterations has been reached.

One of the most popular partitional clustering techniques is the k-means clustering algorithm [Mac67] [Har75] which is based on the calculation of the means. The first step is to specify the number of clusters which are supposed to be detected. Then the representatives of the clusters, the centroids, are initialized randomly or based on some a-priori

information. Each point is then assigned to its nearest centroid which are then recalculated. The two previous steps are repeated until a certain stopping criterion, e.g. no more changes, is met. This version of the algorithm, also known as Forgy's algorithm [For65], enables parallelization and does not depend on data ordering. A variation of the technique, the classic variant in iterative optimization, determines the impact of assigning a point to another cluster on the objective function in advance. Experiments have shown that the classic version produces better results than Forgy's version [LA99] [SKK00]. Another variant, the Iterative Self-Organizing Data Analysis (ISODATA), additionally applies a series of splitting and merging operations depending on the number of clusters. The advantages of k-means clustering in general are its simplicity and the complexity of $O(N)$ which enables its operation on large data sets. However, there are also some disadvantages: The number of classes has to be specified in advance, the performance is data-dependent, it is sensitive to outliers and the resulting clusters strongly depend on the initial configuration.

Algorithms based on k-medoids [TK03], where each cluster is represented by one of its points, are more robust to noise and outliers than k-means clustering. A medoid is a representative point of a cluster with its average dissimilarity to all other points being minimal. Clusters are therefore considered as groups of points which are close to those medoids. The implementation "Partitioning Around Medoids" (PAM) [NH94] randomly selects non-medoids and evaluates a cost function using a distance metric like the Euclidean distance. If the costs for the new candidate are sufficiently low, then it is exchanged with the current medoid. The procedure is iteratively repeated with new candidates until the medoids no longer change. Another variant of this algorithm "Clustering LARge Applications" (CLARA) [KR90] applies PAM to several subsets of the data and takes the best of these results. "Clustering Large Applications based on Randomized Search" (CLARANS) [NH94] goes even further. It interprets the search as a graph where every node corresponds to a possible solution (set of medoids) and an edge connects two neighbors if they differ by a single medoid. The algorithm iteratively compares a node to its neighbors and searches for a local minimum. If the corresponding node has been found, then CLARANS uses it as starting point for a new iteration. Otherwise, it restarts the search with a randomly selected node until a certain number of local minima has been found. The advantage of CLARANS, which has a complex-

ity of $O(N^2)$, over CLARA is that it uses a random search for finding the neighbors. In contrast, CLARA compares only few neighbors which correspond to a fixed, small sample.

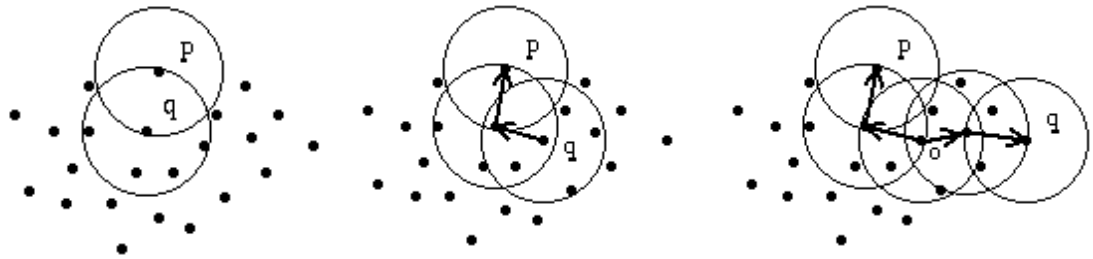
For data sets with overlapping clusters, a fuzzy technique can be used to determine the membership degree of the data points in the clusters. An example for this method is the fuzzy c-means (FCM) algorithm [Bez81] by Bezdek which is based on the fuzzy extensions of the least-square error criterion. Similar to the k-means clustering, the number of classes which are supposed to be found have to be specified in advance. Jain et al. [JMF99] state that the algorithm is better than k-means clustering at avoiding local minima, but the convergence to local minima of the squared error criterion does still exist.

Expectation Maximization (EM) algorithms [MK96] are used for finding maximum likelihood estimates of parameters in probabilistic models which depend on unobserved latent variables. The clustering is performed using Gaussian probability density functions (PDF). The algorithm is an iterative optimization technique consisting of an expectation and a maximization step. The first step determines the membership probability of a point in each cluster (fuzzy classification), whereas the maximization step uses the known and the expected values for generating a new estimate of the parameters. These two steps are repeated until convergence.

3.3 Density-Based Clustering

The idea behind density-based clustering is that clusters can be interpreted as dense regions of data which are separated by regions of lower densities. When it is seen as a connected dense component it can grow in any direction where the density leads. This approach enables the detection of clusters with other than the typical blob-like shapes known from k-means clustering by dividing the feature space into density-based grid units.

The algorithm “Density Based Spatial Clustering of Applications with Noise” (DBSCAN) developed by Ester et al. [EKSX96] is based on the conditions of (direct)



(a) (b) (c)

Figure 3.3: Assuming that *MinPts* equals 2, the border point p in (a) is directly density-reachable from q but not vice versa. In (b) p is density-reachable from q but not vice versa. In (c) p and q are density-connected to each other by o. [EKSX96]

density-reachability and connectivity. A few definitions have to be given before advancing to the algorithm:

- The ε-neighborhood of a point p from a data set D is defined by $N_\varepsilon(p) = \{q \in D | \overline{pq} \leq \varepsilon\}$.
- If the number of points contained within an ε-neighborhood of a point p reaches at least a certain minimum value $|N_\varepsilon(p)| \geq MinPts$, then p is a core point, otherwise it is a border point.
- A point p is directly density-reachable from a point q if p is in the ε-neighborhood of q and if q is a core point. This criterion is symmetric for pairs of core points but not for a combination of core and border points, as shown in Figure 3.3a.
- A point p is density-reachable from a point q if there is a series of points $p_1 \ldots p_n$ with $p_1 = q$, $p_n = p$ and p_{i+1} being directly density-reachable from p_i. As for the directly density-reachable points, this criterion is not symmetric for a pair of core and border points, as shown in Figure 3.3b.
- A point p is density-connected to a point q if there is a point o such that both, p and q are density-reachable from o (Figure 3.3c).
- A cluster C is a set of all points p and q which satisfy the following conditions:

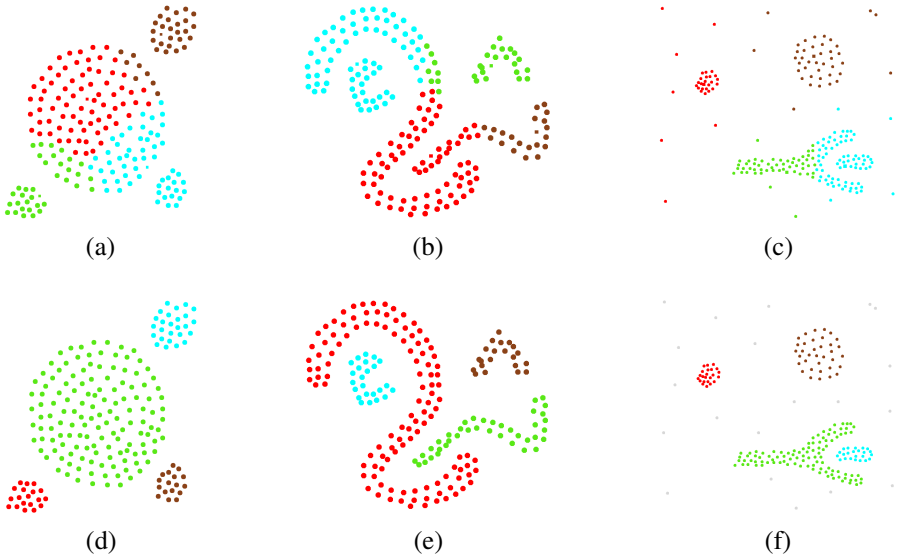


Figure 3.4: Clusters which have been found by CLARANS are shown in (a) - (c) whereas (d) - (f) haven been detected by DBSCAN [EKSX96].

  - maximality: $p \in C$ and q is density-reachable from p
  - connectivity: p is density-reachable from q

- All points which do not belong to any cluster are interpreted as noise.

The clustering process is started by checking the ε-neighborhood of an arbitrarily selected point p. If p is a core point, then a new cluster with all points q from $N_\varepsilon(p)$ is created. Each of these points is then processed by analyzing its ε-neighborhood. If it contains at least $MinPts$, then the neighboring points of q are a added to the cluster, and the procedure is executed for them too. Additionally, the algorithm may merge two clusters if they are close enough. The authors state that their algorithm has a computational complexity of $O(n \log n)$ when using spatial access methods such as R*-trees [BKSS90]. A direct comparison of the results obtained with DBSCAN and CLARANS based on the SEQUOIA 2000 benchmark data [SFGM93] can be seen in Figure 3.4. DBSCAN correctly classifies the clusters in each data set as well as the noise. The drawback of DBSCAN is that it is very sensitive to the definition of ε and $MinPts$. In contrast, CLARANS splits "natural" clusters apart, merges others over regions of low density and simply assigns noise to the nearest cluster centers.

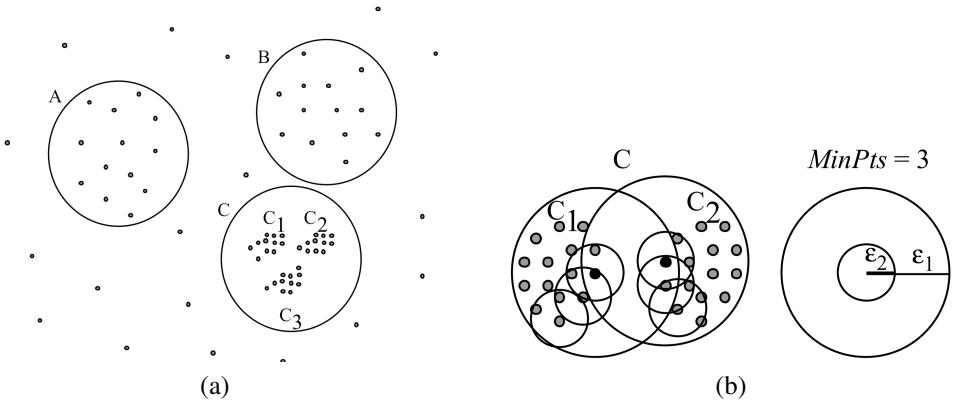


Figure 3.5: Algorithms which use global density parameters (a) fail to simultaneously detect clusters with different densities. Varying the ε-value in the OPTICS algorithm (b) solves this problem and finds clusters with different densities. [ABKS99]

A disadvantage of DBSCAN and other similar clustering algorithms is that they use global density parameters. This makes it impossible to simultaneously detect clusters with different densities in a data set, as can be seen in Figure 3.5a. Either the clusters $A - C$ or the clusters $C_1 - C_3$ but not A, B, $C_1 - C_3$ are found. The structures of the clusters simply cannot be represented by a single, fixed value for the density parameter. Hierarchical clustering algorithms address this issue, but the single-link method presented in Section 3.1 has two disadvantages. It does not split clusters apart which are connected by only few points having a small inter-object distance, and the resulting dendograms can be difficult to interpret when they consist of several hundred clusters. Alternatively, partitioning algorithms which are based on densities could be used with varying parameter settings. The difficulty with this approach is the infinite number of possible values for the parameters. Even when trying to cover a large range of values, the problem of analyzing the result still remains apart from the enormous amount of required secondary memory.

These difficulties were the motivation for Ankerst et al. [ABKS99] to develop the algorithm "Ordering Points To Identify the Clustering Structure" (OPTICS), which can be seen as an improved version of DBSCAN. It is based on the usage of a varying ε' with $\varepsilon' \leq \varepsilon$ for fixed values of ε and $MinPts$. Using a lower value for ε' may result in a higher number of denser clusters as can be seen in Figure 3.5b. The idea behind

Figure 3.6: Example for a reachability plot used in the OPTICS clustering algorithm [ABKS99].

OPITCS is to use these ε' values in parallel which requires a certain processing order for these runs to ensure the consistency of the results. Instead of storing memberships to the clusters, only the processing order of the objects and two distance parameters, the core- and the reachability-distance, are kept. The core-distance ε' of an object p is defined as the smallest distance between p and an object from its ε-neighborhood that still qualifies p as a core object. The reachability-distance is the smallest distance between a core object o and another object p with p being directly reachable from o. By defining a fixed $\varepsilon' \leq \varepsilon$, the resulting ordering can be used to generate a density-based clustering. This is done by using the core- and reachability-distance for assigning every point to a cluster or declaring it as noise. The ordering can also be visualized in a reachability plot which is a histogram with the ordered objects plotted on the first and the reachability-distances plotted on the second axis. For the resulting diagram, the exact values of the input parameters ε and $MinPts$ only have a small influence as long as they are high enough. Figure 3.6 shows the relationship between clusters of a two-dimensional data set and their correspondences in the reachability plot. Instead of manually defining ε-thresholds, a hierarchical clustering can be automatically obtained by introducing the following definitions:

- steep upward point: a point which is ξ% lower than its successor
- steep downward point: analog to steep upward point
- steep upward area: interval $[s,e]$ of two steep upward points s and e with each point in between being at least as high as its predecessor and which does not contain more than $MinPts$ in a row that are not steep upward points
- steep downward area: analog to steep upward area

Based on these definitions, a set of points is considered as a cluster if it satisfies the following conditions:

- the starting point is inside a steep downward area
- the ending point is inside a steep upward area
- it consists of at least $MinPts$ points (core condition of OPTICS algorithm)
- the reachability values of all points are at least $\xi\%$ lower than those of the first point of the downward area and the successor of the last point of the upward area

The result of the clustering can be controlled with the parameter ξ. A high value detects few but significant clusters whereas a lower value is the right choice if the result is too coarse. The authors state that their algorithm has a computational complexity of $O(n \log n)$ which is the same as for DBSCAN due to its structural similarity.

Hinneburg and Keim [HK98] have developed a clustering method "DENsity-based CLUstEring" (DENCLUE) which generalizes other clustering techniques like partitional, hierarchical or density-based methods. It uses an influence function, for example the parabolic, the square wave or the Gaussian function, which describes the impact of a data point on its neighborhood. This influence function is evaluated for each data point and the density of the whole data space is determined by creating the sum of the influence functions of every point. The clusters can then be identified by introducing so called density attractors which are local maxima of the general density function. They can be calculated by using hill-climbing techniques which are supported by the gradient of the general density function. The algorithm is capable of detecting arbitrarily shaped clusters and handling high amount of noise but depends on a good selection of the density parameter and the noise threshold. In order to improve the computational efficiency of the algorithm, the data set can be split into d-dimensional hypercubes where only those are stored in a tree structure which actually contain data. Using this optimization technique the complexity can be reduced to $O(n \log n)$.

The mean shift algorithm introduced by Fukunaga and Hostetler [FH75] and further adapted for various image processing applications by Comaniciu and Meer [CM02] is based on a gradient ascent approach. The neighbors of a point within a certain search window are used for the calculation of the mean either with a flat or a truncated Gaussian

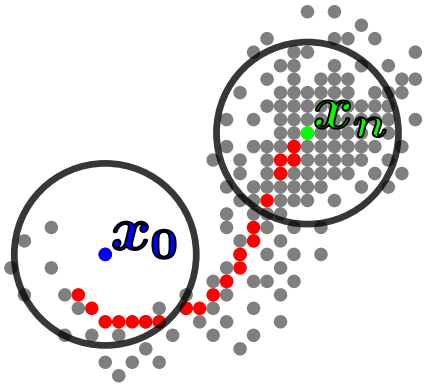


Figure 3.7: Example for the iterations of the mean shift clustering procedure. Starting at position x_0 (blue point) the search window is moved in the direction of strongest increase in density (red points) until convergence (green point).

kernel, as shown in Figure 3.7. For the next iteration this mean shift vector moves the kernel window in the direction of the highest increase in density, i.e. in the direction of the gradient. This sequence of calculating the mean and window shifting is repeated until the procedure converges towards a certain mode, the cluster center. In regions of low density the steps of the algorithms are big and decrease as the window is moved towards local maxima. The result of the algorithm mainly depends on one parameter, the bandwidth. It controls the size of the search window for the calculation of the local mean. If a small value is chosen, the algorithm will detect many small clusters, which may actually belong to the same bigger cluster. The worst case would be a bandwidth of the size of a single element. This would lead to every element being a cluster. In contrast, the result will contain only few clusters if the chosen value for the bandwidth is too high. If this is the case, then the algorithm will merge clusters, which in fact should stay separated. The extreme case in this scenario would be a single cluster which contains all elements of the data set. The algorithm is explained in more detail in Section 4.3.

4 Methods & Implementation

In the following sections the steps for the classification and visualization of a volumetric data set are explained. Figure 4.1 gives an overview of the individual stages. First, the preprocessing of the data is performed which is required for the generation of the LH histogram. This includes the calculation of the first partial and the second directional derivatives. The combination of the first partial derivatives leads to the gradient of the data which is used for both, the calculation of the LH values and the application of the lighting model during the volume rendering stage. Based on the preprocessed data, the LH values are obtained through the integration of the gradient and are used for the creation of the LH histogram. Afterwards, the mean shift clustering algorithm is applied to this histogram. The next step is the choice and construction of a transfer function. This can be done using different types which require more or less input. Finally, the setup and steps of the ray casting procedure for rendering the volumetric data set based on the selected transfer function are explained.

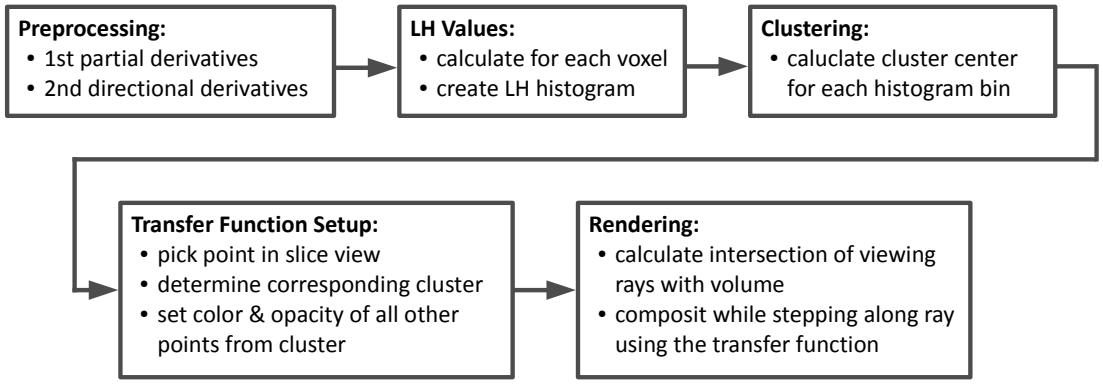


Figure 4.1: Processing pipeline for the classification and visualization of a data set.

4.1 Derivatives

The calculation of the LH histograms involves a number of preprocessing steps, i.e. the determination of the first partial derivatives, the gradients, and the gradient magnitudes. The generation of the mirrored LH histogram additionally requires the second directional derivatives.

4.1.1 First Partial Derivatives

The first partial derivatives need to be calculated for the generation of the gradients. By integrating the resulting gradient field in both, the negative and positive gradient direction, the procedure will converge to the L and H values respectively. There are several methods for estimating the derivatives which will be presented below. They are obtained by convolving the data set with certain derivative kernels. The process of the convolution of a volume with a one-dimensional kernel is illustrated in Figure 4.2. For each voxel the kernel (colored cubes) is aligned parallel to one of the coordinate axes with its center (red) being positioned over the current voxel. Then each element of the kernel is multiplied by the underlying value of the volumetric data set. The final result for this voxel is the sum of these multiplications. Due to the size of the kernel the voxels at the border of the data set have to be treated separately. The derivatives in this region are assumed to be zero.

Central Difference Method

The first kernel presented in this work uses the central difference:

$$\frac{1}{2}\begin{bmatrix}1 & 0 & -1\end{bmatrix} \tag{4.1}$$

The first partial derivate with respect to x, $(\frac{\partial f}{\partial x})$, for the density value at position (x,y,z) is calculated by multiplying the density values at position $(x-1,y,z)$, (x,y,z) and $(x+1,y,z)$ by $\frac{1}{2}$, 0 and $-\frac{1}{2}$, respectively and by adding up these values. This process is

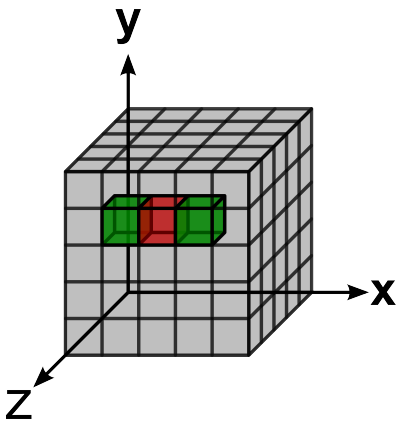


Figure 4.2: Example for a convolution of a 3D data set with a 1D kernel (colored cubes).

repeated for each voxel by subsequently shifting the filter through the volume. The derivatives $\frac{\partial f}{\partial y}$ and $\frac{\partial f}{\partial z}$ are calculated in the same way. Amongst the presented methods the central difference is the computationally most effective one: it has a complexity of $O(m)$ for each voxel where m is the size of the kernel ($m = 3$ in this case). Despite its high performance, the central difference method should only be used when the sampling rate of density values is well below the Nyquist limit [JHG99]. This is due to the Nyquist–Shannon sampling theorem which states that a signal can be reconstructed without the loss of information if the sampling rate is at least two times the highest frequency in the data.

Sobel Operator

This kernel combines the central difference method with a smoothing of the data:

$$\frac{1}{8}\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \tag{4.2}$$

It is considerably slower than the central difference method since it has a computational complexity of $O(m^2)$ for each voxel. The number of calculations for the convolution can

be reduced by using the separable version of the kernel [Hla01]. A kernel is separable if it can be composed of two multiplications:

$$\frac{1}{2}\begin{bmatrix}1\\2\\1\end{bmatrix} \otimes \frac{1}{4}\begin{bmatrix}1 & 0 & -1\end{bmatrix} = \frac{1}{8}\begin{bmatrix}1 & 0 & -1\\2 & 0 & -2\\1 & 0 & -1\end{bmatrix} \tag{4.3}$$

For the two-dimensional case the computational complexity can be reduced from $O(m^2)$ to $O(2m)$. For a three-dimensional kernel the usage of the separability has an even bigger impact because the complexity can be lowered from $O(m^3)$ to $O(3m)$.

The procedure for obtaining the derivatives of the three-dimensional data set using the Sobel operator is similar to the central difference method. The derivative $\frac{\partial f}{\partial x}$ is calculated as follows:

1. convolve the data set with the kernel $\frac{1}{4}\begin{bmatrix}1 & 2 & 1\end{bmatrix}$ in the z direction
2. apply the same kernel to the result of step 1 in the y direction
3. convolve the result from step 2 with the kernel $\frac{1}{2}\begin{bmatrix}1 & 0 & -1\end{bmatrix}$ in the x direction

The derivatives for the other directions are obtained in the same way.

Gaussian Derivative Kernels

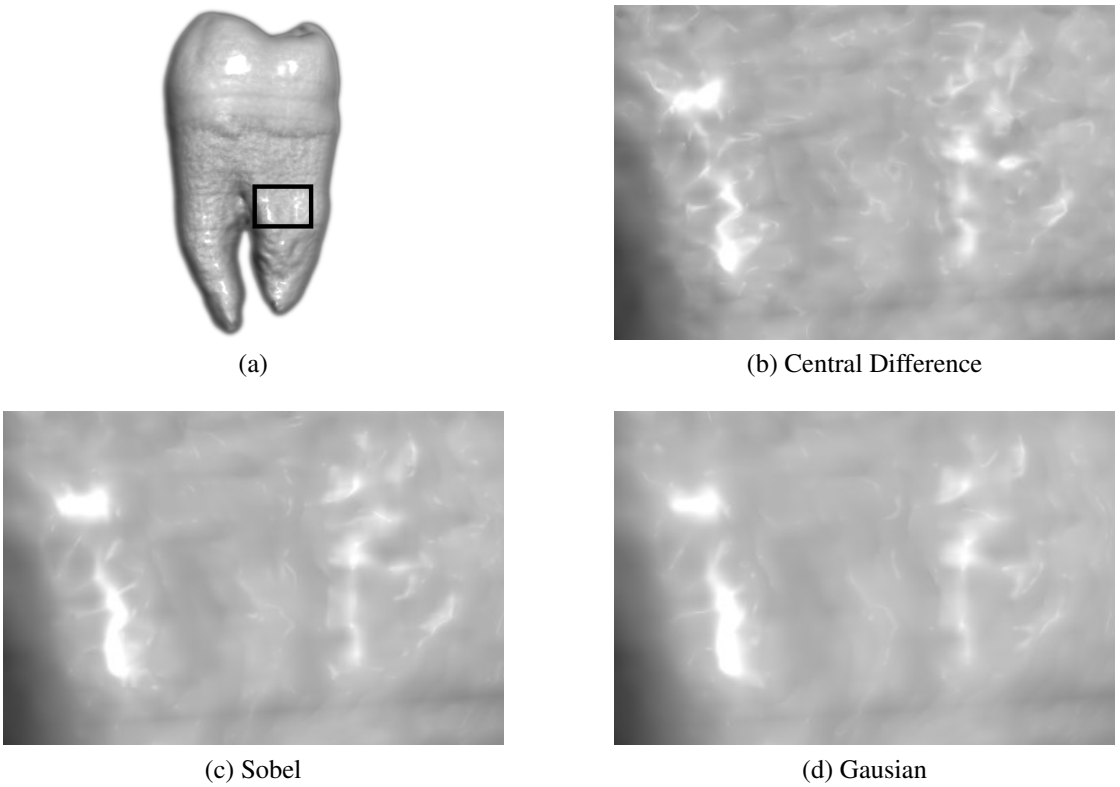
Gaussian derivative kernels exhibit even better differentiation properties than simple differences [JHG99]. Only the one-dimensional version of the Gaussian function needs to be sampled because the function itself and its derivatives are separable.

$$f(x;\sigma) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{x^2}{2\sigma^2}} \tag{4.4}$$

$$f'(x;\sigma) = \frac{-x}{\sigma^3\sqrt{2\pi}}\, e^{-\frac{x^2}{2\sigma^2}} \tag{4.5}$$

The authors suggest that the width of the kernel should be proportional to the standard deviation σ to avoid the early truncation of the Gaussian function, i.e. the radius of the kernel should be at least 4σ.

The separability has an increasing effect on the performance when bigger kernels are used: for a 3*x*3*x*3 kernel the saving is about 67% ($3 * 3$ multiplications instead of 3^3), for a 5*x*5*x*5 kernel about 88% and for a 7*x*7*x*7 kernel about 94%. A comparison of the results which have been obtained by applying the presented methods are shown in Figure 4.3.



(a) (b) Central Difference

(c) Sobel (d) Gausian

Figure 4.3: The consequence of choosing different derivative kernels can be visualized by applying lighting using the gradient as normal vector for the Phong shading model.

4.1.2 Second Directional Derivatives

Two different methods for the calculation of the second directional derivatives, described by Kindlmann [Kin99], have been implemented in this work. The first one uses the gradient of the gradient:

$$D^2_{\widehat{\nabla f}} f = \frac{1}{\|\nabla f\|} \nabla(\|\nabla f\|)\, \nabla f \tag{4.6}$$

The second algorithm involves the calculation of the Hessian matrix for each voxel:

$$\begin{aligned} D^2_{\widehat{\nabla f}} f &= \frac{1}{\|\nabla f\|^2} \nabla f^T\, Hf\, \nabla f = \\ &= \frac{1}{\|\nabla f\|^2} \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} & \frac{\partial f}{\partial z} \end{bmatrix} \begin{bmatrix} \frac{\partial f^2}{\partial x^2} & \frac{\partial f^2}{\partial x \partial y} & \frac{\partial^2 f}{\partial x \partial z} \\ \frac{\partial f^2}{\partial x \partial y} & \frac{\partial f^2}{\partial x \partial y} & \frac{\partial^2 f}{\partial y \partial z} \\ \frac{\partial f^2}{\partial x \partial z} & \frac{\partial f^2}{\partial y \partial z} & \frac{\partial^2 f}{\partial^2 z} \end{bmatrix} \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial z} \end{bmatrix} \end{aligned} \tag{4.7}$$

4.2 LH Histogram

The following sections describe the techniques which are required for the calculation of the LH values. Different integration methods are introduced as well as the choice of the optimal stopping criterion.

Gradient The first partial derivatives which have been described in the previous section are now combined to the gradient:

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right) \tag{4.8}$$

Gradient Magnitude The gradient magnitude is not only used for the calculation of the LH histogram but also during the rendering phase for enhancing the borders in the final visualization.

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2 + \left(\frac{\partial f}{\partial z}\right)^2} \tag{4.9}$$

4.2.1 Integration

The choice of the integration method which is used for stepping along the gradient has a significant effect on the quality of the LH histograms and the renderings. Three different iterative methods for the approximation of solutions of ordinary differential equations have been implemented. The Euler method

$$y_{i+1} = y_i + h\nabla f(y_i) \tag{4.10}$$

calculates the subsequent position y_{i+1} based on the gradient of the current position $f(y_i)$ using a certain step size h. It is the fastest but also least accurate amongst the presented techniques. It suffices for the development of the LH histograms but when it comes to applying the clustering in a subsequent step it is advisable to use more accurate algorithms like the two-stage explicit Runge-Kutta method

$$y_{i+1} = y_i + h\left(\tfrac{1}{4}\nabla f(y_i) + \tfrac{3}{4}\nabla f\left(y_i + \tfrac{2}{3}h\nabla f(y_i)\right)\right) \tag{4.11}$$

or Heun's method which is a two-stage second-order Runge-Kutta method.

$$y_{i+1} = y_i + \tfrac{1}{2}h\left(\nabla f(y_i) + \nabla f\left(y_i + h\nabla f(y_i)\right)\right) \tag{4.12}$$

It is important to normalize the gradient before using it for the integration. It turned out that using a step size of one voxel is a good balance between performance and accuracy. A smaller value did not noticeably change the LH histogram or improve the separation of objects in the renderings of the tested data sets. A comparison of the results using different integration methods and step sizes is presented in Chapter 5.

4.2.2 LH Values

After calculating the derivatives the algorithm for finding the LH values can be executed. For each voxel it tracks the gradient by using one of the integration methods described in the previous section. The H values are determined by following the path in the direction of the gradient whereas the negative gradient is used for finding the L values. The positions along the path do not necessarily correspond to a single voxel but can also lie in between voxels. Therefore, the volumes are accessed by using trilinear interpolation. The integration stops when one of the following stopping criteria is met:

- The density values are not strictly increasing (H values) or decreasing (L values). Alternatively it can be checked if the gradient magnitude is equal to zero. When noisy data is used it may be necessary to introduce a certain threshold in order to avoid an early and incorrect termination of the integration.
- The gradient magnitude starts to increase after it has been decreasing during previous steps along the gradient. This means that either a saddle point or an extremum has been passed. The case of the extremum should be covered by the previous stopping criterion if it is checked first.

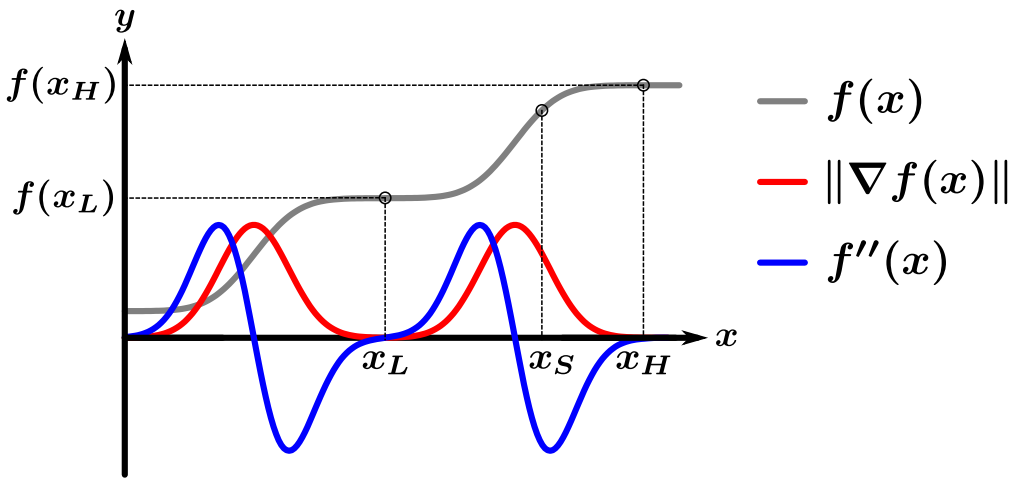


Figure 4.4: Density value profile $f(x)$ (black) while stepping along the gradient, the corresponding gradient magnitude $\|\nabla f(x)\|$ (red) and the second directional derivative $f''(x)$ (blue). The positions (x_L and x_H) of the LH values ($f(x_L)$ and $f(x_H)$) for an example starting position (x_S) are shown. Illustration inspired by [Kin99] and [SBSG06].

In the example shown in Figure 4.4 the integration starts at x_S. For finding the H value the algorithm continues to follow the gradient as long as the density values $f(x)$ are strictly increasing. At position x_H this is no longer the case because the profile has become constant. Therefore, the integration stops and the H value has been found.

If only the first stopping criterion would be used for finding the L value, then the algorithm would not stop at the position marked by x_L. Instead, it would continue until the density value profile becomes constant or when the boundary of the volume has been reached. This is the case because the absolute difference between the density values of two successive steps along the (inverse) gradient in Figure 4.4 is not zero around x_L. To avoid such long paths the second stopping criterion is used: At the starting position of the path (x_S) a flag which indicates if the gradient magnitude has been decreasing during previous steps is set to false. As soon as the maximum of the gradient magnitude $\|\nabla f(x)\|$ to the left of x_S has been passed, the flag is set to true and the second stopping criterion is checked. The integration stops at x_L because at this point the second stopping criterion is met. The number of the different combinations of LH values are stored in a two-dimensional histogram, the LH histogram.

By persistently storing the LH values and the second directional derivatives for each dataset, they need to be calculated only once. It is advisable to do the same for the first partial derivatives because they are also required for the Blinn-Phong shading model which is used in the rendering procedure.

4.3 Mean Shift Clustering

For a single point of the LH histogram its corresponding cluster center is determined as follows:

1. find the neighbors x_i of the current point x within a search window having the radius h (also denoted as bandwidth)
2. calculate the mean shift vector by evaluating Equation 4.13
3. shift the kernel window by the mean shift vector

These steps are repeated until convergence.

The mean shift vector is calculated as follows:

$$m(x) = \frac{\sum_{i=1}^{n} x_i g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)}{\sum_{i=1}^{n} g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)} - x \tag{4.13}$$

where

$$g(x) = -k'(x) \tag{4.14}$$

Possible kernel profiles $k(x)$ and their derivatives are the Epanechnikov kernel profile

$$k_E(x) = 1 - x \tag{4.15}$$

$$k'_E(x) = -1 \tag{4.16}$$

and the normal kernel profile

$$k_N(x) = \exp\left(-\frac{1}{2}x\right) \tag{4.17}$$

$$k'_N(x) = -\frac{1}{2}\exp\left(-\frac{1}{2}x\right) \tag{4.18}$$

These steps are executed for every element of the LH histogram. Apart from storing the resulting cluster center for all points, it is also important to create a list of distinct clusters and points which belong to them. This needs to be done in order to be able to select an entire cluster for the automatic creation of the cluster-based transfer function, as described in Section 4.4.3. Additionally, a distinct color is assigned to every cluster. It is also advisable to use a downsampled version of the LH histogram instead of applying the mean shift clustering to the original data. Because of the computational complexity of the algorithm the resolution should therefore be lower than 256×256. Apart from the size of the data itself the bandwidth is another factor which influences the duration of the procedure. Increasing the radius of the search window has two effects: On the one hand it reduces the number of resulting clusters because the probability that a second,

(a) LH Histogram (b) Initial clustering (c) Higher bandwidth (d) Merging thin clusters

Figure 4.5: The downsampled LH histogram (128×128) of the tooth data set is shown in (a). The result of the clustering exhibits some undesirable thin clusters (b). Increasing the bandwidth is not the ideal solution because this merges larger clusters too (c). The problem is solved by explicitly merging only thin clusters.

larger cluster exists in the surrounding of the current search window is higher. On the other hand the procedure takes more time to finish because more elements have to be included into the calculation at each step.

The performance of the algorithm also depends on how the elements of a single bin of the histogram are treated. Especially for data sets which have only few non-empty bins with a high number of contributing voxels the procedure will finish considerably faster if weights are used for these points. Instead of evaluating the function $g(x)$ for the same position m times, where m is the number of contributing voxels for the current bin of the histogram, the expression is evaluated only once and then multiplied by m. Another technique for speeding up the calculation is to create a separate two-dimensional array with the same size as the histogram. A list of visited positions is maintained while stepping along the mean shift vector. After the cluster center has been determined, a reference is stored in the array for every position of the path. As soon as the path of a subsequent pass intersects one of these positions the iteration can be stopped prematurely as the mean shift vector would converge towards the same, existing cluster center.

Depending on the data set and the chosen bandwidth it can happen that thin clusters are detected, as shown in Figure 4.5b. A possible solution for this problem would be to simply increase the bandwidth parameter. However, this approach does not necessarily merge the thin clusters only, but also some of those which should stay separate as they

correspond to different material boundaries (Figure 4.5c). Another way of addressing this problem is to check the bounding box of the clusters. If the length of the shortest side falls below a certain threshold, i.e. the minimum desired cluster size and if the nearest neighboring cluster is within a search window having the radius equal to the bandwidth parameter, then they can be merged (4.5d).

4.4 Implementation

The project has been implemented in C# and for rendering DirectX in combination with the High Level Shading Language (HLSL) has been used. During the development and the evaluation of the algorithm it proved to be very useful to persistently store the preprocessed data like the first partial and the second directional derivatives separately for each data set. By doing so, a lot of redundant computation time can be saved. It also comes in handy when comparing different integration methods or when varying the parameters for the calculation of the LH histogram.

The data sets which have been used for testing and evaluating the algorithm mainly originate from the volume library of the University of Erlangen [UEN09]. In the subsequent chapters some of the volumes have also been used in order to visualize the individual steps of the applied techniques. Table 4.1 shows some properties of the data sets like the resolution, the number of bytes per voxel, and whether the volume has been included in the benchmarks of Section 5.2

Some data sets which are stored using a 16bit/voxel accuracy nearly use the full range of possible values [0 65535]. These data sets have been scaled down to the interval [0 4095]. Due to memory limitations it is not possible to create a histogram as large as 65536×65536. Data sets with a resolution higher than $512 \times 512 \times 256$ have been scaled down as well. For the density values and the LH values unsigned integer accuracy is sufficient because the values are always greater than or equal to zero whereas the gradients do have a sign.

Data Set	X	Y	Z	Bytes/Voxel	Benchmark
Carp	256	256	512	2	✓
CT-Head	256	256	113	2	✗
Engine	256	256	256	1	✓
Hand	244	124	257	2	✗
Piggy Bank	512	512	134	2	✓
Sheep Heart	352	352	256	1	✗
Skewed Head	184	256	170	2	✓
Tooth	256	256	161	2	✓

Table 4.1: Properties of the data sets which have been used in this thesis.

4.4.1 LH Histogram

Before creating the LH histogram, the minima and maxima of the LH values need to be calculated. They are used to determine the size of the histogram and for being able to access the transfer function texture during the rendering phase later on. The LH histogram is stored in a two-dimensional array which has the size

$$s = \big(max\big(f_L(x)\big) - min\big(f_L(x)\big) + 1\big) \times \big(max\big(f_H(x)\big) - min\big(f_H(x)\big) + 1\big)$$

Data sets which use the full range of possible values of the 16 bit per voxel accuracy need to be downsampled at the beginning, as described in Chapter 4, due to memory limitations. The histogram is then constructed by determining the correct bin for each pair of corresponding LH values. Bins are equally sized sections of the histogram and contain an integer value which represents the number of the associated values.

LH Histogram Image

In order to judge the quality of the LH histogram or to manually create a transfer function by selecting parts of the LH histogram, a visual representation needs to be generated. A logarithmic scale is used to get a meaningful histogram image. If the values would be directly mapped to a range of grayscale values [255 0], then there would be

only few regions showing bins that are not empty. After the application of logarithm the result is normalized and the values are mapped to gray levels to produce the final image. Instead of using only gray levels for visualizing the underlying number of contributing voxels a color gradient as shown in Figure 4.6 can be used. Even after applying the logarithmic scale there are only few elements of the histogram image with the highest possible value. This is due to the fact that the histogram bins with the highest number of contributing voxels are the ones around the diagonal of the LH histogram representing the homogeneous regions of the data set. Depending on the ratio of the actual size of the histogram data and the desired histogram image size either upsampling or downsampling needs to be performed. Upsampling is applied to the histogram image after the operations mentioned above whereas downsampling is applied directly to the data before these operations by merging the bins of the histograms.

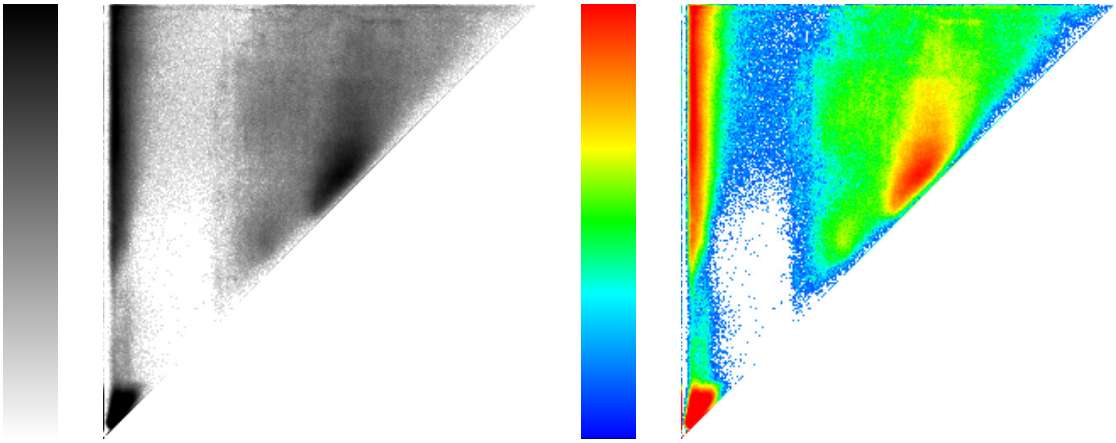


Figure 4.6: Examples of different color mappings for the LH histogram (sheep heart data set). The distribution is easier ratable when using colors instead of only gray values.

4.4.2 Mirrored LH Histogram

The mirrored LH histogram can be calculated as follows: For each voxel the second directional derivative described in Section 4.1.2 is examined. If it is greater than zero then the LH values of the volumetric data sets are exchanged, otherwise the algorithm proceeds to the next voxel. As a result the histogram based on these altered LH values is also populated below the diagonal as can be seen in Figure 4.7.

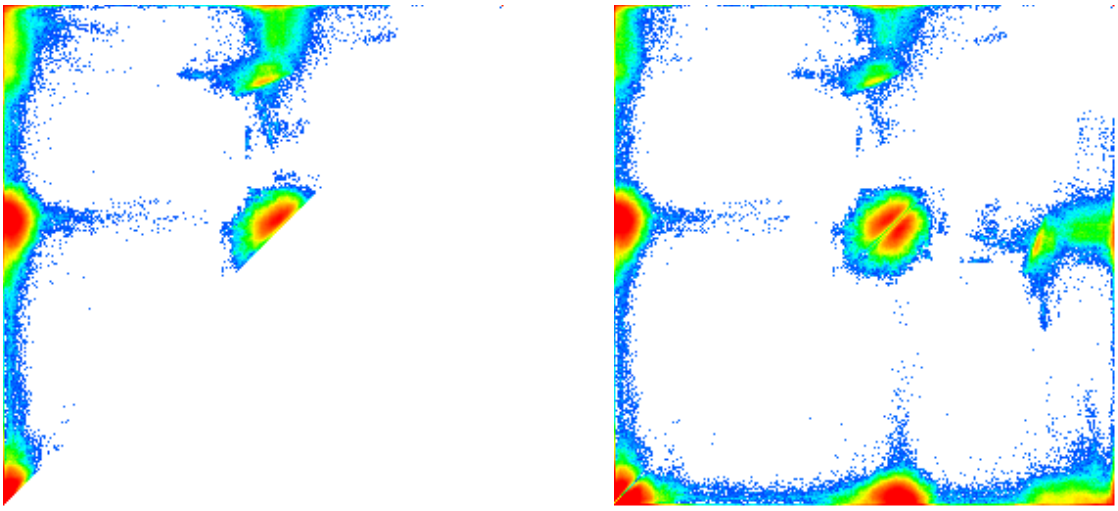


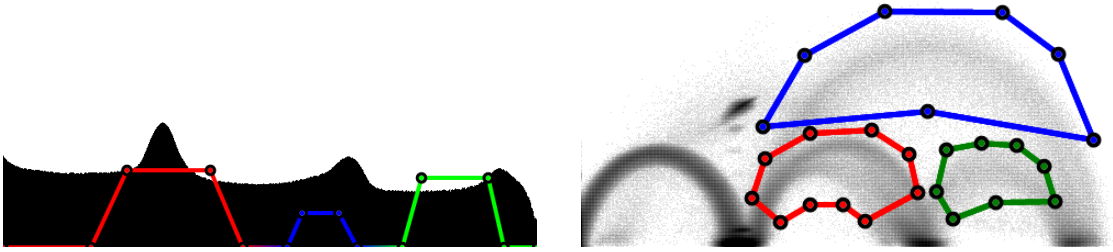
Figure 4.7: Example for a mirrored LH histogram using the engine data set.

4.4.3 Transfer Function Setup

The generation of the transfer function is a crucial part of the volume rendering procedure because small changes at this point result in big differences in the final rendering. Here the user decides which parts of the volume are going to be displayed and which parts will be hidden. Both, the automatic cluster-based approach and other manual transfer function design methods which require more user input and basic understanding of the underlying techniques have been implemented. For data sets where the automatic approach does not perform well the user can switch back to the manual design mode and modify the transfer function. It is also possible to combine the cluster-based with the manual technique to make changes only where necessary avoiding a complete manual setup of the entire transfer function.

4.4.4 Manual Design

The interaction widget for creating a one-dimensional transfer function which is based on the histogram of the density values is shown in Figure 4.8a. Control points are created at any position by clicking onto the desired location. The horizontal position corresponds to the density value whereas the vertical position determines the opacity. A



(a) 1D: based on density values

(b) 2D: based on density values and gradient magnitudes

Figure 4.8: Transfer function types and interaction widgets.

color can be assigned by using a standard color selection dialog. Colors for the density values which lie between two control points are interpolated.

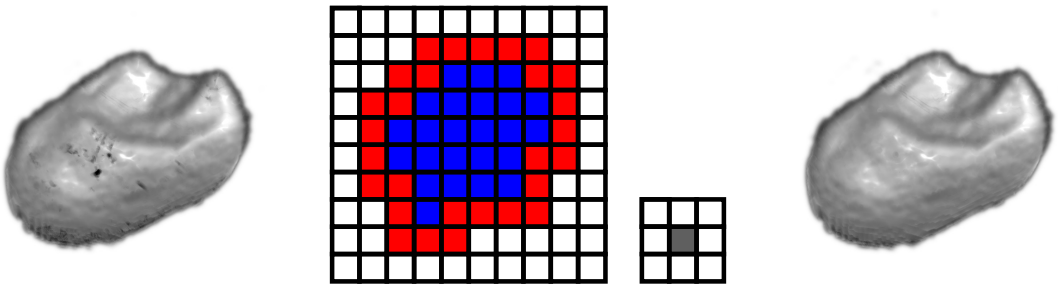
For the design of a two-dimensional transfer function similar tools are available, as shown in Figure 4.8b. This kind of histogram is based on density values (x-axis) and gradient magnitudes (y-axis). Polygons can be defined by generating control points like for the one-dimensional transfer function. In the two-dimensional case both color and opacity are selected in separate dialogs and the setup of appropriate shapes is more demanding.

4.4.5 Cluster-Based Approach

The interaction required by the cluster-based technique is a single click onto the (border) region of interest in the slice view. The following operations are then executed automatically by the application: First, the algorithm for finding the LH values described in Section 4.2.2 is applied for the selected voxel only. As the procedure is evaluated only for a single entry of the volumetric data set the operation takes only a fraction of a second. The LH values are then used as input parameters for the clustering algorithm. The mean shift procedure also terminates instantly because it is evaluated for a single point only. The result of the operation is the cluster center in the LH histogram which is then used to retrieve the list of points which belong to this cluster. For each point the corresponding position in the two-dimensional transfer function is highlighted using the cluster-specific color. The final result is a colored blob in the transfer function which has

the same shape and size as the selected cluster. As for the manual design of the transfer function the automatically assigned color and opacity can be replaced by custom values. The manual adaptation of the opacity is useful when the feature in the final rendering has a low contrast. This can happen when there is only a small number of voxels with a high gradient magnitude which are widely scattered.

Some points of the original LH histogram which lie close to the edge of the colored region may not be covered by the blob due to the downsampling of the histogram data described in Section 4.2.2. As a result cracks and holes can appear in the final visualization based on this transfer function as can be seen in Figure 4.9a. With the help of dilation [Pra01], a morphological operation used in image processing, these artifacts can be removed. Figure 4.9b shows an example based on artificial data (blue) using a 3×3 structuring element. The result of this operation is that the data highlighted in red is being added. So, instead of coloring only the current point of the LH histogram, its neighborhood which is defined by a structuring element gets covered by this color too. A visualization based on this dilated version of the transfer is shown in Figure 4.9c.



(a) Dentine-enamel boundary of the tooth data set without dilation.

(b) Example of a dilation.

(c) Dilation applied using a 3×3 structuring element.

Figure 4.9: Dilating the transfer function removes holes in the visualization.

4.5 Volume Rendering

Having prepared the required data, the last step for generating the visualization is the rendering. In order to provide interactive frame rates the ray casting is done on the

graphics card as described by Krüger and Westermann [KW03]. The following sections show which data structures are used for the volumetric data and the transfer functions, how the viewing rays for the ray casting procedure are calculated, and finally which steps need to be performed during the shading and compositing stage.

4.5.1 Texture Setup

In order to access the volumetric data, i.e. density values, gradients, gradient magnitudes, and LH values, it needs to be sent to the graphics card using textures. The volumetric data sets that have been used in this work contain density values requiring more than 8bit/voxel which is the typical accuracy for a single channel in typical 4-channel RGBA texture. Therefore 16bit unsigned single-channel textures formats (e.g. for the gradient) and 32bit signed single-channel floating point formats (e.g. for the derivatives) have been used to store the data. These textures have to be created only once during startup of the application.

For transfer functions that are only based on the density values a one-dimensional 32bit RGBA texture can be used. It stores the color and opacity values defined by the user and is accessed by using the normalized density value. Transfer functions which are based on density values and gradient magnitudes or LH values are stored in two-dimensional 32bit RGBA textures. The values obtained from the volumetric data sets need to be mapped to the interval [0, 1] too in order to be able to access the textures at the correct positions.

4.5.2 Calculation of the Viewing Rays

In the first two render passes of the ray casting procedure the entry and exit coordinates of the viewing rays which are cast through the volume are determined. They yield the starting positions and direction vectors used in the third rendering pass. In order to calculate these coordinates a proxy geometry representing the bounding box of the volumetric data set needs to be generated. This proxy geometry is then rendered with activated back face culling to determine the entry positions of the rays. In the vertex

shader the position of the current vertex is stored as a texture coordinate which is then passed to the pixel shader. The texture coordinates are interpolated for every pixel which is then shaded using the x, y and z values of the texture coordinates as r, g and b values. The result of this shader pass is not rendered directly to the screen but stored in a texture in order to be able to access it in the third rendering pass. The same procedure is repeated for the second render pass but this time front face culling is using instead of back face culling for obtaining the rays' exit coordinates from the volume. Again, the result is stored in a separate texture.

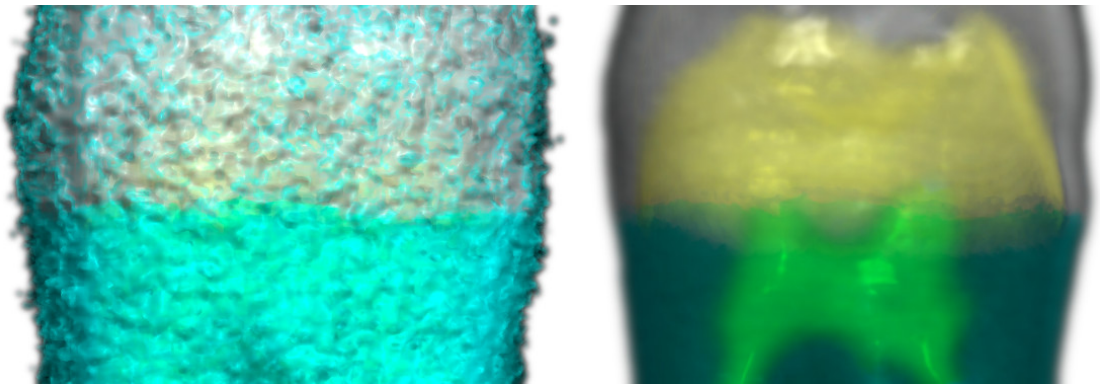
4.5.3 Shading and Compositing

The third rendering pass uses the entry and exit positions of the viewing rays to calculate the direction vectors of the viewing rays. They are used for sampling the volume using trilinear interpolation which is already implemented on the graphics card. Depending on the transfer function type one of the following texture lookups and calculations is performed:

1D based on density values: The normalized density value of the current voxel is used to access the one-dimensional density transfer function texture to obtain color and opacity for the current position.

2D based on density values and gradient magnitudes: Both, the density value and the gradient magnitude are normalized and then used for the texture lookup in the corresponding two-dimensional transfer function texture.

2D based on LH values: The procedure is the same as for the previous transfer function. Additionally, the gradient magnitude is used to scale the opacity of the result from the texture lookup. By doing so the border regions of the data set are enhanced as can be seen in Figure 4.10.



(a) Weights disabled (b) Weights enabled

Figure 4.10: Impact of the gradient magnitude used as weight for the opacity values.

After the application of the transfer function, the voxel is shaded using the Blinn-Phong reflection model [Bli77]. This algorithm requires the gradient as it is interpreted as the surface normal vector. There are two possibilities for obtaining the gradient: Either it is passed as a texture to the graphics card as described in Section 4.5.1, or it is calculated on the fly for every frame in the pixel shader. The first method requires less mathematical operations which improves the rendering performance. It also produces better renderings when more accurate derivative kernels have been used for the preprocessing. The second method on the other hand requires less texture memory. Finally the opacity is additionally scaled by the step size to avoid the occurrence of further artifacts when using lower sampling rates.

These operations need to be performed for every step along the ray. The final color of the pixel on the screen that corresponds to this ray is then calculated by using front-to-back compositing from Equation 1.2.

5 Results

In this section renderings are presented as well as the transfer functions and parameters which haven been used for creating them. Another important part is the performance of the technique which consists of a time-consuming preprocessing. This needs to be done only once for a data set which can then be displayed at interactive frame rates. The measurements and renderings have been created on a system consisting of an AMD Athlon64 X2 dual core CPU with 2GB of main memory using a Nvidia GeForce 9800 GT with 1GB of graphics memory.

The evaluation of the algorithm was done using several data sets from the volume library of the University of Erlangen [UEN09]. It turned out that mostly CT data sets were suitable for the design of transfer functions based on LH histograms. The histograms obtained from MRI scans did not exhibit unambiguously distinguishable regions in the LH histogram due to various image artifacts. The optimal value for the clustering bandwidth is about 20% of the histogram size and was obtained through experiments. When using lower values, the algorithm tends to converge towards local maxima of the histogram data. As a result, clusters which actually represent the same material are split apart. In contrast, higher values result in neighboring clusters being merged with the most dominant ones. This has the consequence that different materials which are close to each other in the LH histogram are assigned to the same class.

The visualizations in Section 5.1 are presented in combination with LH histograms which serve three purposes: First, they show the logarithmically scaled number of elements contributing to each of the histogram bins. This is done by varying the brightness of the corresponding pixel with white corresponding to the highest and black to the lowest number. Second, different colors are assigned to the clusters to distinguish them from each other in the histogram. Third, these colors are used directly for the transfer

function setup. The LH values for all data sets were calculated using Heun's method described in Chapter 4.2.1 with a step size of one voxel in combination with mean shift clustering using 20% of the histogram size as bandwidth parameter, unless denoted otherwise.

Section 5.2 shows the resulting visualizations using methods of different accuracy for the classification. Additionally, the performance is analyzed using benchmarks which compare the time required for the corresponding computations.

5.1 Visualizations

Figure 5.1 shows the tooth data set rendered with transfer functions based on the clustering of the LH histogram using different values for the bandwidth parameter. For each configuration two LH histograms are displayed. One which shows the clusters that have been used for the transfer function setup (Figure 5.1d - Figure 5.1f) and another one which shows all detected clusters (Figure 5.1g - Figure 5.1i). The additional histogram of all clusters is presented due to the ambiguous meaning of the color white (gray levels). The gray cluster shown in Figure 5.1g - Figure 5.1i was actually assigned this gray value for the transfer function setup. The clusters and the bottom (yellow, blue, orange, and red) were not assigned any color and did not contribute to the transfer function. In Figure 5.1a the bandwidth is too low, and as a result the pulp region is split apart, introducing a new, artificial material. In contrast, the consequence of choosing a bandwidth that is too high (Figure 5.1c) is that pulp and dentine are merged into a single material. The best result is obtained using the above mentioned value of 20% (Figure 5.1b). There is still a gap in the pulp region, but this effect cannot be avoided with the LH transfer function only, because it is the consequence of artifacts in the data set. As one follows the pulp from top to bottom, the L value of the corresponding pulp-enamel boundary increases until it reaches the level of the L value of the air-enamel boundary. For this region of the volume data set both boundaries project onto the same area in the LH histogram and therefore cannot be distinguished, neither by a manual, nor by an automatic clustering technique which is based on the LH histogram only. This is the reason

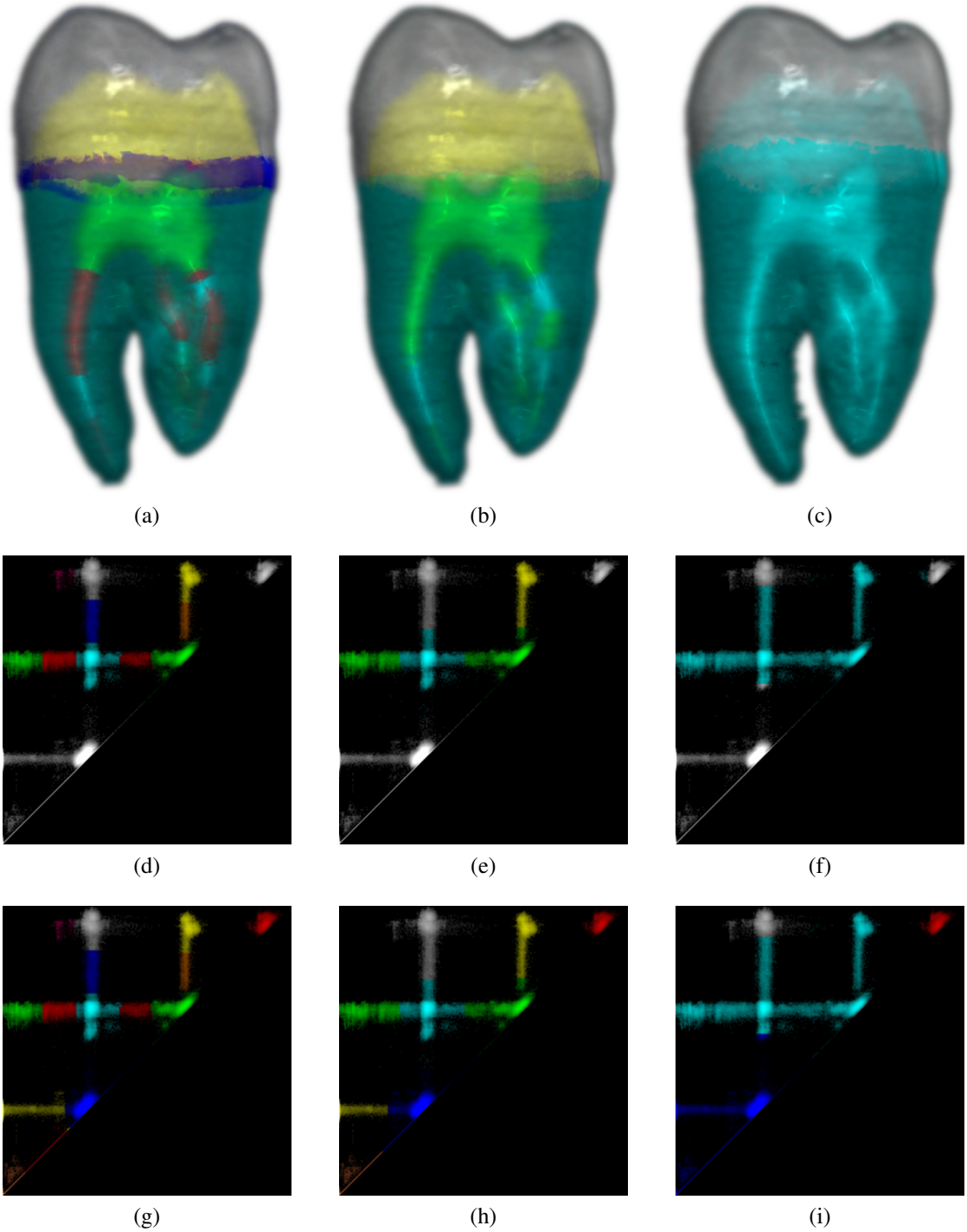
why both regions were assigned the same color (green) because they belong to the same boundary although they are separate clusters.

For the calculation of the LH values in Figure 5.2 different step sizes have been used. It can be seen that the influence of the artifacts which are responsible for the gaps in the pulp region decreases with lower step sizes. A drawback of the higher accuracy is the increased duration of the calculation of the LH values.

The clustering of the LH histogram of the hand data set shown in Figure 5.3 was also executed with different values for the bandwidth parameter. Similar to the tooth data set, the clusters are not compact but elongated. Using only 10% (a) splits the bone into 3 artificial materials, whereas 50% (c) is too much and merges the cluster on the diagonal (homogeneous tissue) with the cluster above (bone). A value of 20% (b) is the best choice although the extent of the yellow cluster does not reach far enough to the bottom of the region that corresponds to bone.

The LH histogram of the engine data set shown in Figure 5.4a exhibits compact clusters which can be easily detected by the mean shift algorithm. Varying the clustering parameter within the range of 20% - 50% does not significantly influence the result. For some data sets, including this one, it may occur that a material is surrounded by two other materials. The two separate clusters at the top actually represent the border region of the cylinders. They are bounded by air on one side and by the housing on the other side. Therefore, the integration of the gradient field leads to two different L values: the intensity of air and the intensity of the housing. In order to avoid this artificial separation, the two clusters were assigned the same color as can be seen in Figure 5.4c. Figure 5.4e shows the data set with the rendering based on the mirrored LH histogram. On the one hand the borders appear more crisp than in the standard LH histogram but on the other hand sampling artifacts start to appear because the ray caster only crosses few opaque border voxels.

The LH histogram of the carp data set shown in Figure 5.5 contains two dominant clusters which have been used for the visualization: the air-tissue and tissue-skeleton boundary. The impact on the rendering of using a higher step size for the calculation of the LH values can be seen in Figure 5.5c. The thin regions of the tailfin disappear because the integration converges to the density of the surrounding air on both sides.

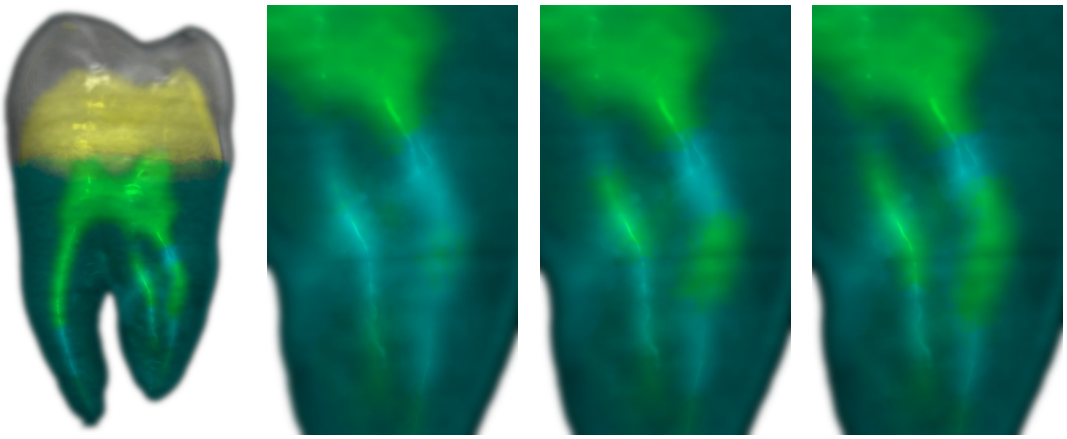


(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figure 5.1: Mean shift clustering applied to the LH histogram of the tooth CT data set ($256 \times 256 \times 161$). The renderings in the first row are based on the transfer functions shown in the second row whereas the third row shows all detected clusters. The values for the bandwidth used in the columns 1, 2, and 3 are 10%, 20% and 50% of the histogram size, respectively.



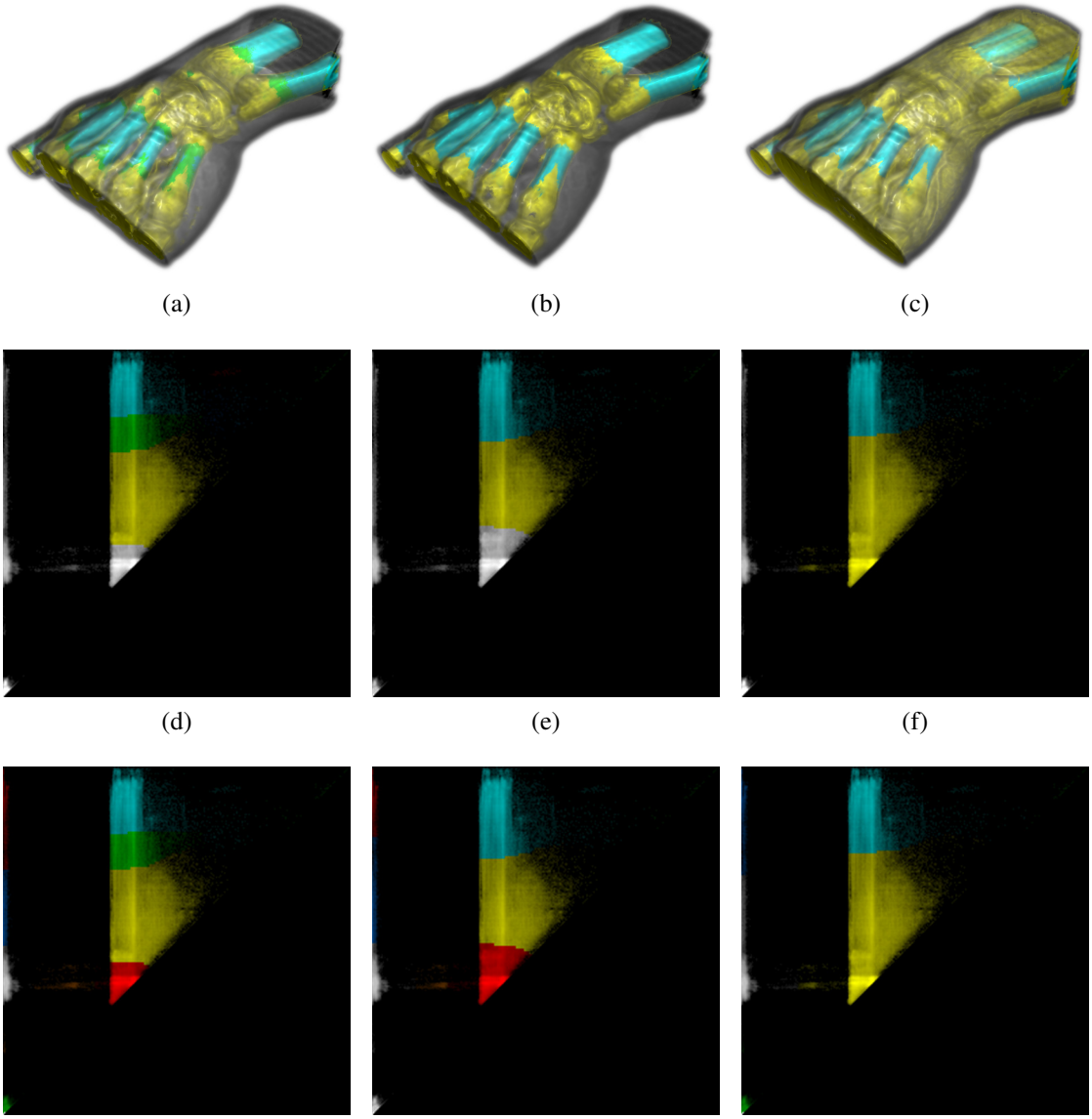
(a) Step size: 0.25 (b) Step size: 2.0 (c) Step size: 1.0 (d) Step size: 0.25

Figure 5.2: Comparison of different integration step sizes for calculating the LH values of the tooth data set.

Therefore, the LH values are the same and do not contribute to the voxels' opacities during rendering.

The LH histograms of the previous CT data sets contained clusters which could be visually distinguished easily. For the MRI data set of a sheep heart shown in Figure 5.6 this is no longer as obvious because of the presence of noise. Especially the tissue with the lower intensity (dark blue) is not separated very well. When using a lower bandwidth, parts of this tissue get classified correctly but additional artificial materials are introduced too. The manual setup of the transfer function through trial and error slightly improves the quality of the classification.

The piggy bank CT data set shown in Figure 5.7 can be clearly visualized using the automatically created transfer functions. Using 20% for the bandwidth parameter (Figure 5.7a) introduces some additional clusters especially at the border between the piggybank and the socket it is mounted on as well as in the middle of the hull. When a higher bandwidth of 30% is used (Figure 5.7b), some of these artificial materials are merged as well as the clusters which represent the coins and the socket because they are quite close in the LH histogram. By manually editing the transfer function (Figure 5.7c) these wrong misclassifications can be corrected.

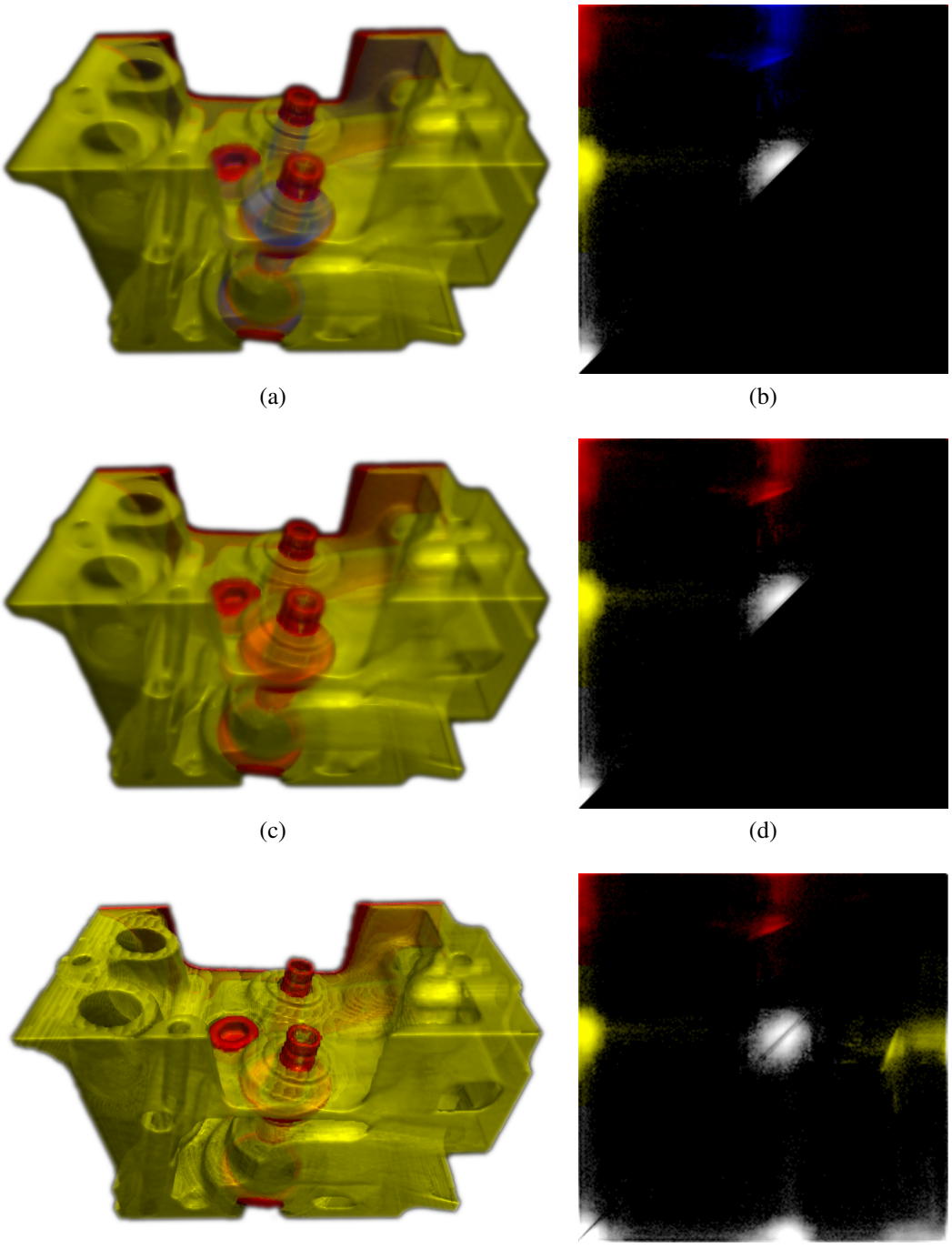


(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figure 5.3: Mean shift clustering applied to the LH histogram of the hand CT data set ($244 \times 124 \times 257$). The renderings in the first row are based on the transfer functions shown in the second row whereas all detected clusters are highlighted in the third row. The values for the bandwidth used in the columns 1, 2 and 3 are 10%, 20% and 30% of the histogram size, respectively.

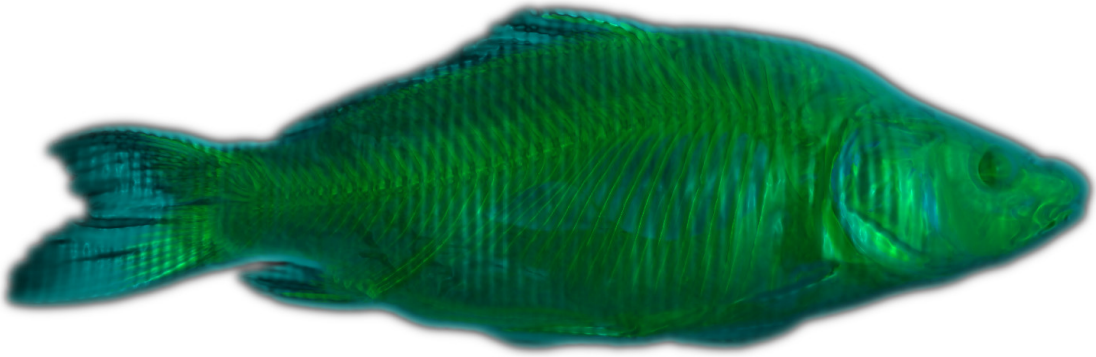


(a) (b)

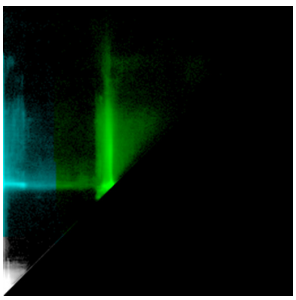
(c) (d)

(e) (f)

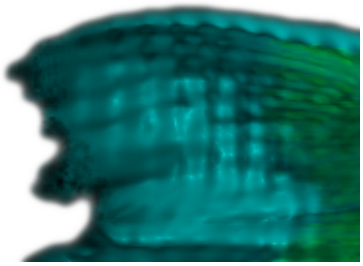
Figure 5.4: Mean shift clustering applied to the LH histogram of the engine CT data set ($256 \times 256 \times 256$). The separated red and blue clusters at the top (b) actually belong to the same object in the data set, they were assigned the same colors in (c) and (d). A rendering based on the clustering of the mirrored LH histogram (f) is shown in (e).



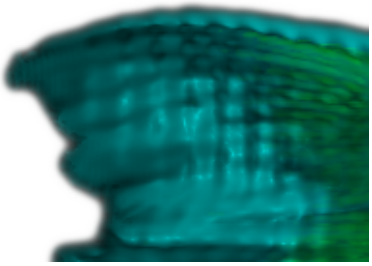
(a)



(b) Bandwidth: 30%



(c) Step size: 2



(d) Step size: 1

Figure 5.5: Mean shift clustering applied to the LH histogram (b) of the carp CT data set (a) ($256 \times 256 \times 512$). The images (c) and (d) show detail views of the tailfin based on LH values calculated with Heun's method using different step sizes.

For the skewed head CT data set shown in Figure 5.8 the automatic classification detects the important structures (bone, skin, and teeth) within the data set. The cheekbone close to the nose is not completely contained within the bone cluster though.

Similar results as for the skewed head are obtained for the Chapel Hill CT data set shown in Figure 5.9. The difficulty especially with this data set is the similar intensity of bone and teeth which compromises the clustering because both materials project into the same region in the LH histogram.

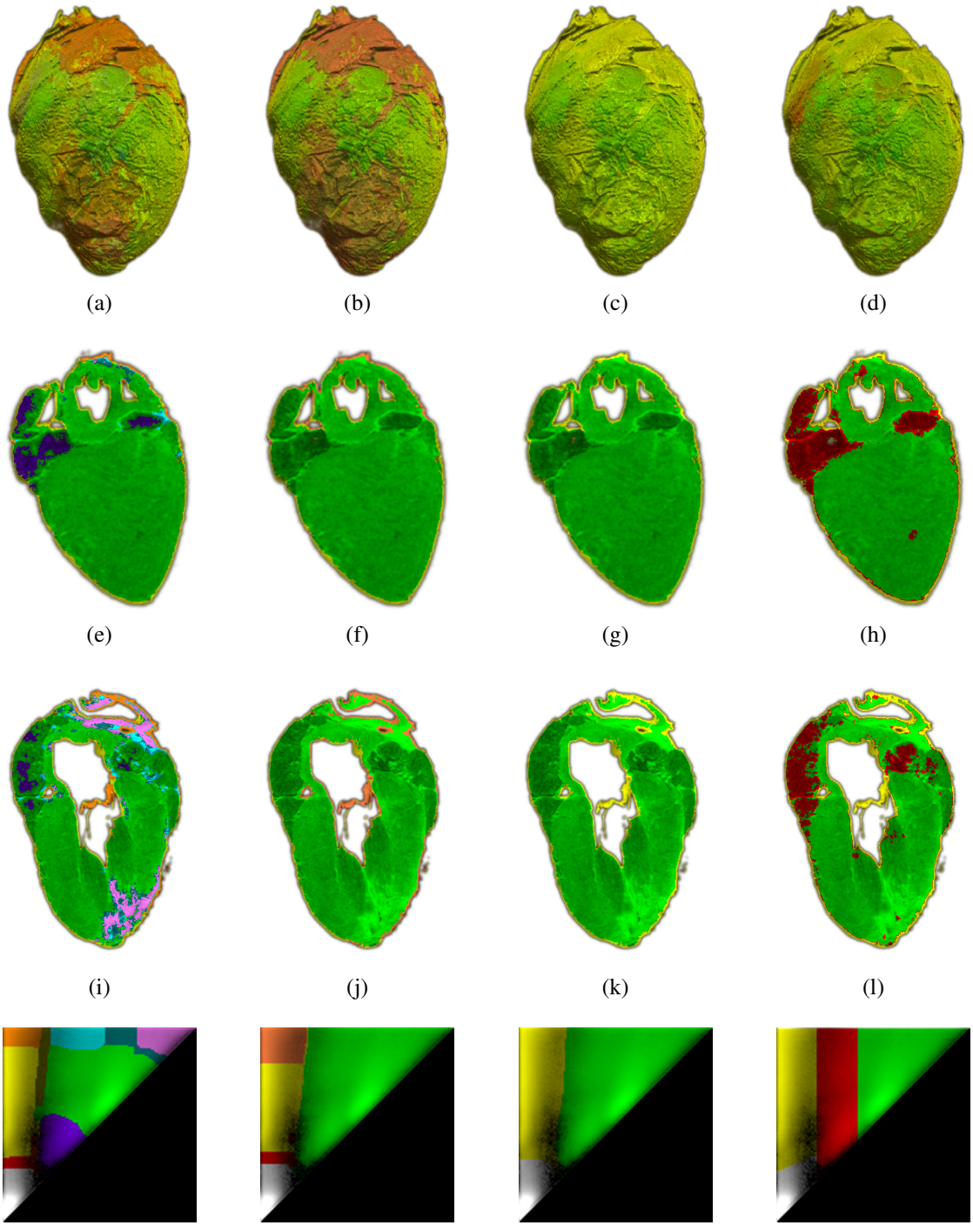
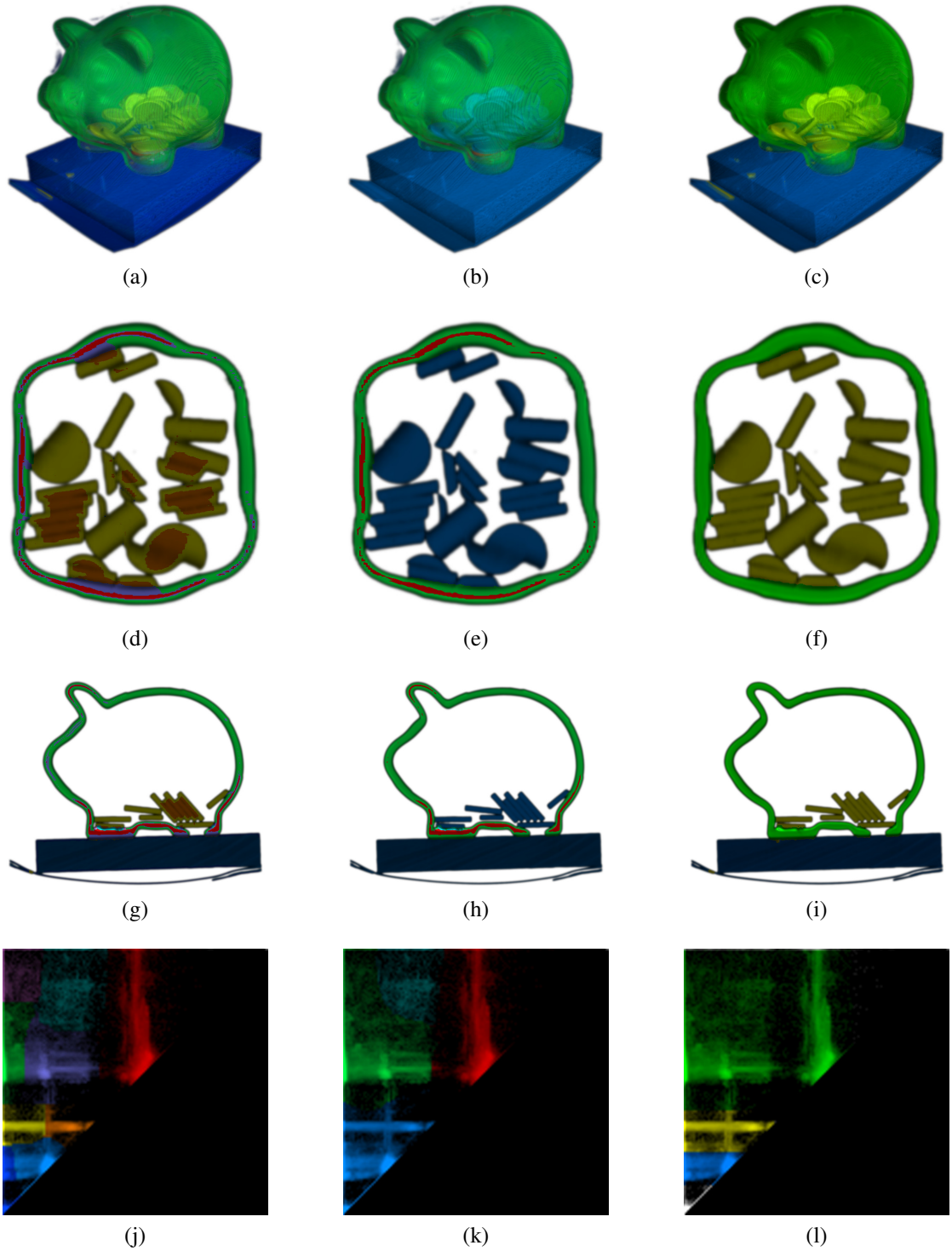


Figure 5.6: The renderings of the sheep heart MRI data set ($352 \times 352 \times 256$) in the columns one, two, and three have been created using the values 15%, 20%, and 30% for the clustering bandwidth parameter respectively, whereas the fourth column is based on a manual creation of the transfer function.



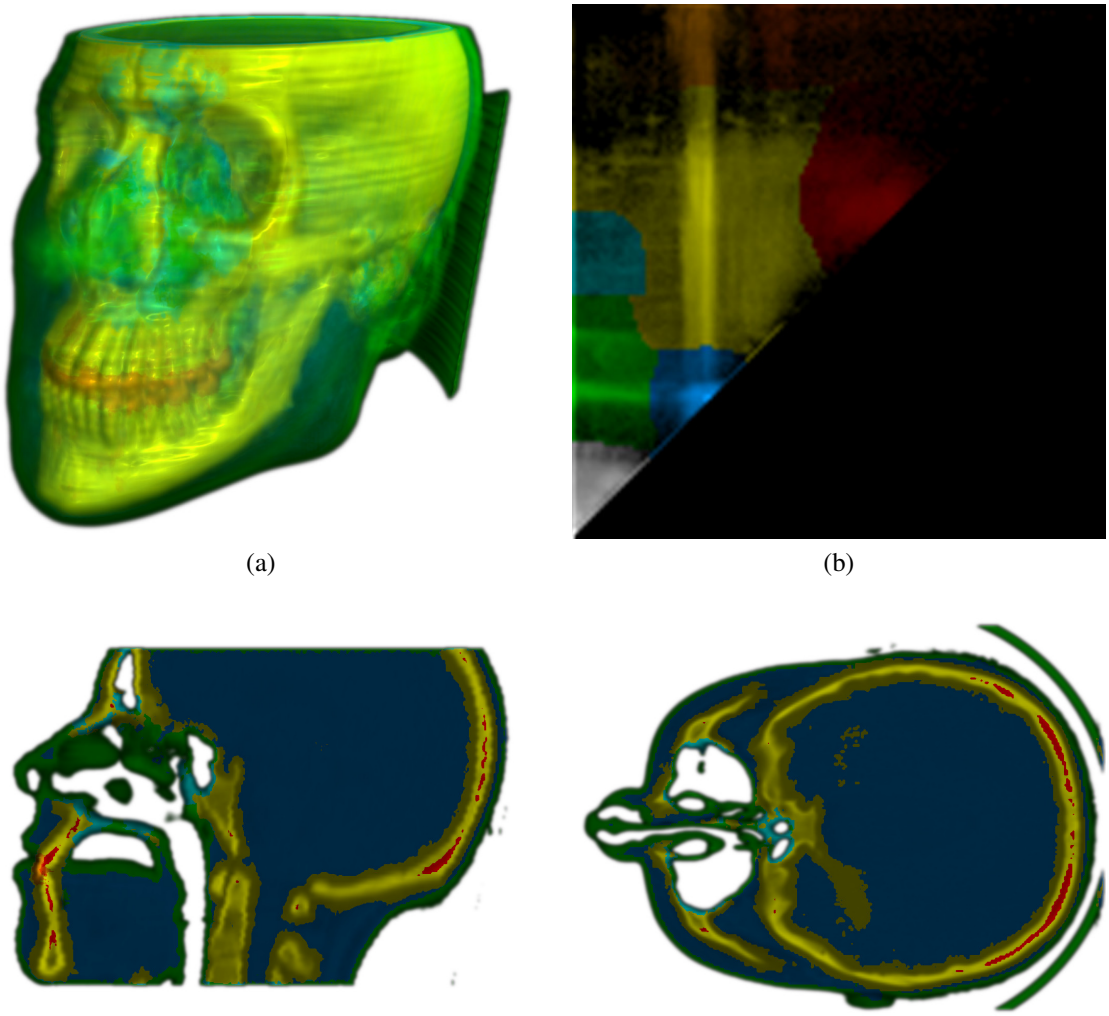
(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

(j) (k) (l)

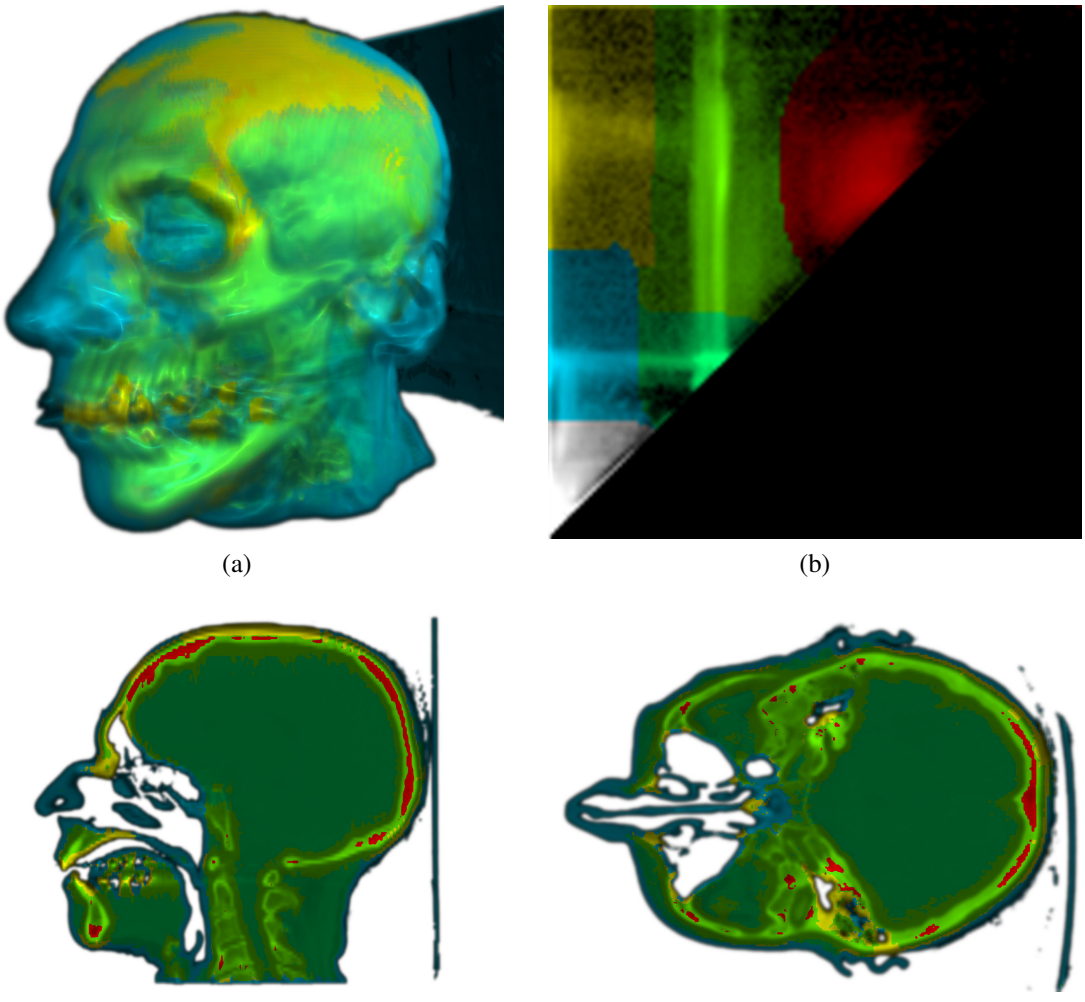
Figure 5.7: The renderings of the piggy bank CT data set ($512 \times 512 \times 134$) in the first two columns have been created using the values 20% and 30% for the clustering bandwidth parameter respectively, whereas the third column is based on a manual creation of the transfer function.



(a) (b)

(c) (d)

Figure 5.8: Mean shift clustering applied to the LH histogram (b) of the skewed head CT data set (a) ($184 \times 256 \times 170$) with (c) and (d) being slice views of the classified data.



(a) (b)

(c) (d)

Figure 5.9: Mean shift clustering applied to the LH histogram (b) of the Chapel Hill CT head data set (a) ($256 \times 256 \times 113$) with (c) and (d) being slice views of the classified data.

5.2 Benchmarks

The data sets used in the previous sections have completely different resolutions, as mentioned in the caption below the figures. In order to obtain comparable results for these data sets the runtime of the algorithms was scaled with the number of voxels and mapped to a theoretical standard size of 128^3. Although the data sets contain different relative amounts of empty space, the numbers still give an impression of the required processing time. It is important to note that these calculations are only executed once for each data set. They are then stored persistently and are reused at a later time.

On average, the calculation of the gradients, i.e. the three first partial derivatives, takes 0.4 seconds for the central difference method and up to 2.5 seconds for a Gaussian derivative kernel of size 7. The duration of the procedure is shown in Figure 5.10. The small variations of the durations between the datasets can most likely be explained by the varying, relative number of scalar values being zero which speeds up the calculations.

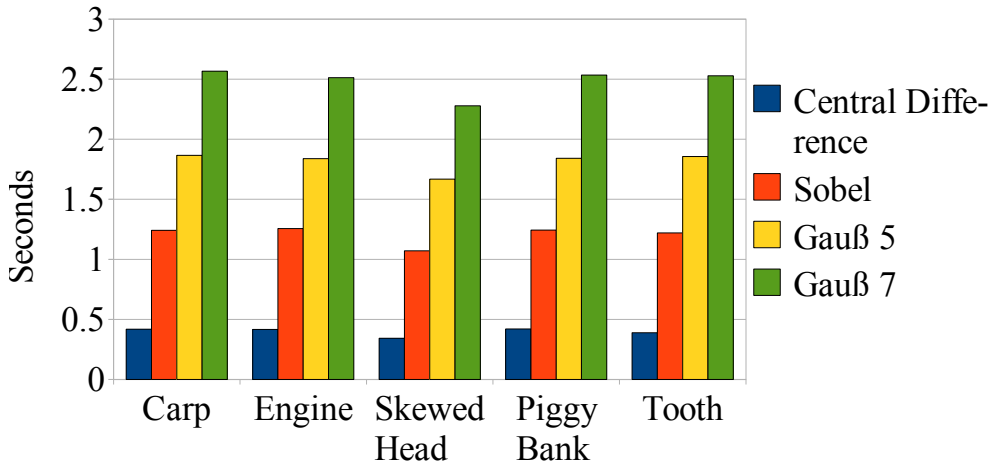


Figure 5.10: Calculation of the gradients using different convolution kernels for the first partial derivatives.

The duration for the integration of the gradient field, in contrast, is hardly influenced by the chosen derivative method but depends on the characteristics of the data set itself.

On average, the engine finishes fastest in only 18 seconds whereas the integration of the piggybank takes up to 40 seconds, as shown in Figure 5.11.

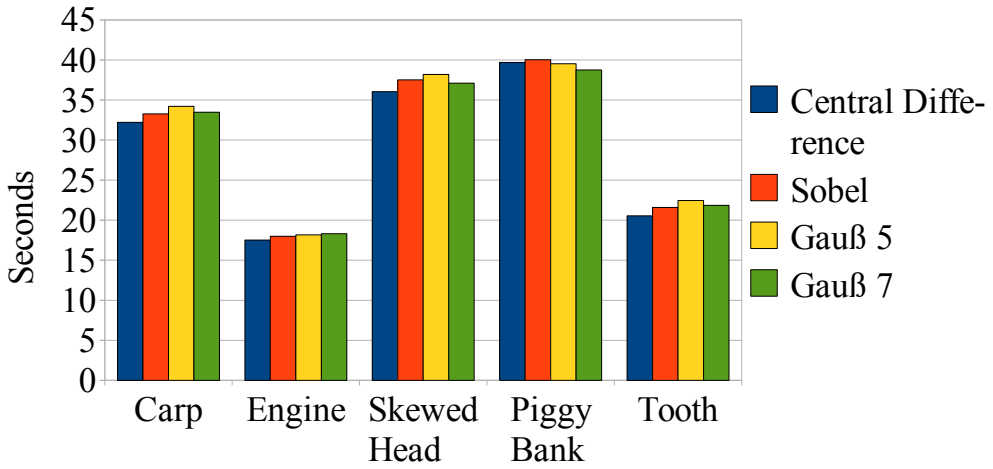


Figure 5.11: Calculation of the LH values based on the gradients which have been obtained using different convolution kernels for the first partial derivatives using Heun's method with a step size of one voxel.

Another important factor for the duration of the procedure is the chosen integration technique. The choice for a specific method is a tradeoff between accuracy and computational effort. Euler's method finishes on average in about 20 seconds whereas Runge-Kutta and Heun take 30 seconds, as can be seen in Figure 5.12.

Even more important for the quality of rendering especially for thin structures and also for the duration of the integration is the chosen integration step size. The most accurate tested value of 0.25 voxels takes about 56 seconds on average whereas a step size of 1 voxel takes about 30 seconds, as shown in Figure 5.13.

The clustering procedure heavily depends on the bandwidth parameter as it defines the size of the neighborhood which is checked for each element of the histogram at each step of the gradient ascent iteration. For a value of 10% the average duration is about 0.7 seconds whereas for 30% it takes 2.6 seconds on average. The reason why the computation time for the skewed head data set significantly differs from the other data sets becomes obvious when looking at the LH histogram shown in Figure 5.8b in comparison to the histograms of the other data sets. The gradient ascent procedure for the mean

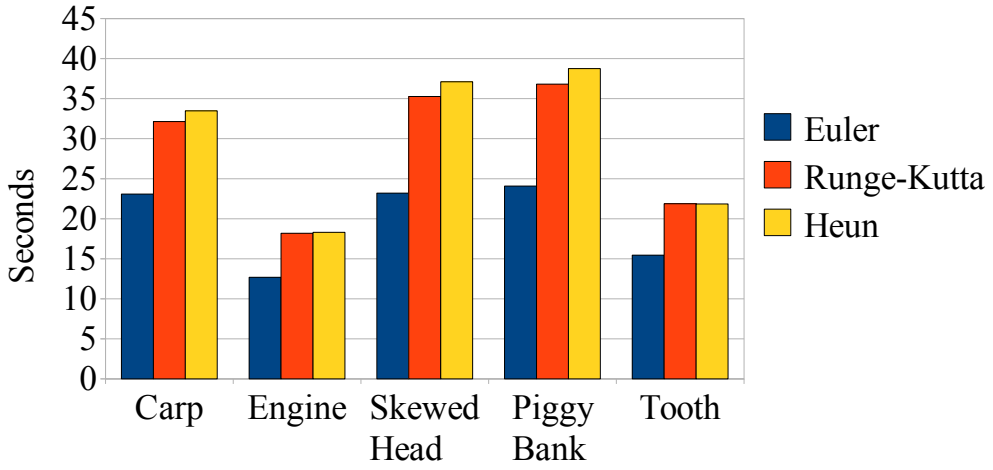


Figure 5.12: Calculation of the LH values using different integration techniques with a step size of one voxel.

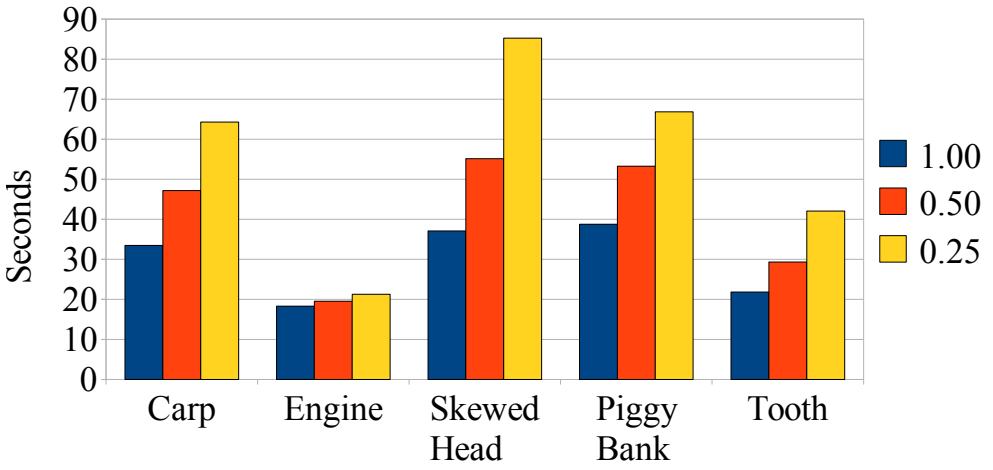


Figure 5.13: Calculation of the LH values using different integration step sizes for Heun's method.

shift clustering is only applied to non-empty histogram bins. One of the characteristic features of the skewed head data set is that nearly every bin of its LH histogram is occupied. Therefore the gradient ascent procedure has to be executed more often than for the other data sets which is the reason for the increased runtime for this data set. The corresponding diagram can be seen in Figure 5.14.

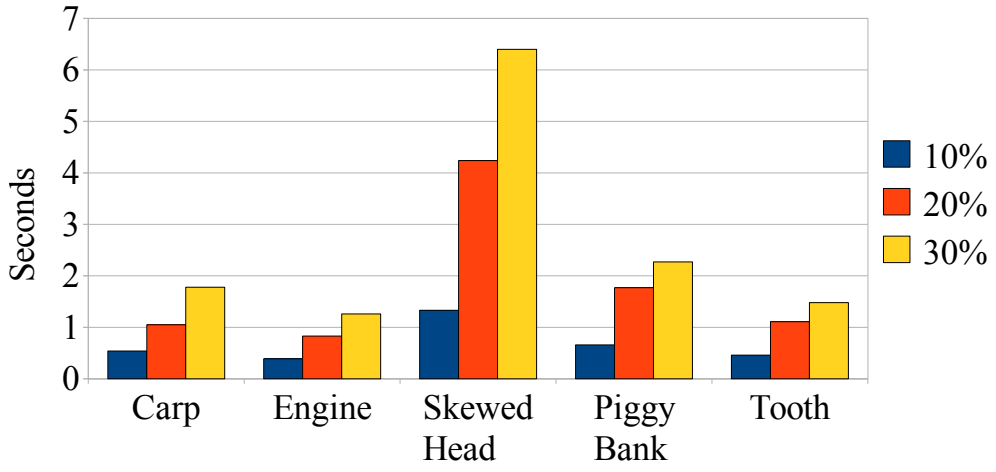


Figure 5.14: Duration of the mean shift clustering applied to the downsampled LH histogram (128×128) using different bandwidths.

6 Conclusion

This thesis has shown various approaches to the design of transfer functions. Several techniques have been developed which produce good results concerning the accuracy of the classification. However, the drawback of these methods is the complexity of the parameter adjustment which is required in order to obtain the desired result. Therefore LH histograms and mean shift clustering have been the ideal candidates for the automation of this process. When implementing this technique it is important to pay attention to the correct usage of the stopping criteria for the integration of the gradient field. It proved to be very useful to create a debugging tool in order to check the integration paths of some of the voxels. This enables the detection of possible mistakes and helps to improve the quality and the performance of the algorithm. The easier the clusters are visually distinguishable in the LH histogram, the better is the visualization using a transfer function which is based on the automatic clustering. The algorithm performs well on (industrial) CT data sets with precise material borders in contrast to (medical) MRI which often lead to blurred clusters due to the unclear tissue transitions and noise. Although the presented technique does not outperform other methods concerning the quality of the visualizations, it does simplify the design of a transfer function. This is because the user can create a three-dimensional visualization of an object by simply selecting it in the slice view. Additionally, there is only a single configuration parameter to control the output of the clustering algorithm. If the users are not satisfied with the result, they have the possibility of either varying this parameter or to modify the transfer function directly using certain interaction widgets. Objects with the same LH values which are separated in space currently cannot be distinguished by the algorithm and are therefore assigned to the same cluster in the LH histogram. A possible improvement could therefore be the introduction of region growing.

7 Bibliography

[ABKS99] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. OPTICS: Ordering points to identify the clustering structure. In *SIGMOD '99: Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, volume 28, pages 49–60, 1999.

[Bez81] James C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Kluwer Academic Publishers, Norwell, Massachusetts, USA, 1981.

[BKSS90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD '90: Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, volume 19, pages 322–331, 1990.

[Bli77] James F. Blinn. Models of light reflection for computer synthesized pictures. In *SIGGRAPH '77: Proceedings of the 4th Annual Conference on Computer Graphics and Interactive Techniques*, pages 192–198, 1977.

[Bon03] Andrea Bonarini. *Soft Computing Applications (Advances in Soft Computing)*. Physica-Verlag, Heidelberg, Germany, 2003.

[Bus00] Stewart C. Bushong. *Computed Tomography (Essentials of Medical Imaging)*. McGraw-Hill Medical, USA, first edition, 2000.

[CM02] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:603–619, 2002.

[CM08] Carlos D. Correa and Kwan-Liu Ma. Size-based transfer functions: A new volume exploration technique. *IEEE Transactions on Visualization and Computer Graphics*, 14:1380–1387, 2008.

[CR08] Jesus J. Caban and Penny Rheingans. Texture-based transfer functions for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 14:1364–1371, 2008.

[DAK09] Swagatam Das, Ajith Abraham, and Amit Konar. *Metaheuristic Clustering*, volume 178 of *Studies in Computational Intelligence*. Springer, Berlin Heidelberg, 2009.

[Def77] D. Defays. An efficient algorithm for a complete link method. *The Computer Journal*, 20:364–366, 1977.

[EHK+04] Klaus Engel, Markus Hadwiger, Joe Kniss, Aaron Lefohn, Christof Rezk-Salama, and Daniel Weiskopf. Real-time volume graphics. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Course Notes*. ACM Press, 2004.

[EHK+06] Klaus Engel, Markus Hadwiger, Joe M. Kniss, Christof Rezk-Salama, and Daniel Weiskopf. *Real-time Volume Graphics*. A K Peters, Wellesley, Massachusetts, USA, 2006.

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD'96)*, pages 226–231, 1996.

[Fal98] Emanuel Falkenauer. *Genetic Algorithms and Grouping Problems*. John Wiley & Sons, Chichester, United Kingdom, 1998.

[FH75] Keinosuke Fukunaga and Larry Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on Information Theory*, 21(1):32–40, 1975.

[For65] Edward W. Forgy. Cluster analysis of multivariate data: Efficiency versus interpretability of classifications. *Biometrics*, 21:768–780, 1965.

[GRS98] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: An efficient clustering algorithm for large databases. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 2–4. ACM Press, 1998.

[GRS00] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. ROCK: A robust clustering algorithm for categorical attributes. In *Proceedings of the 15th International Conference on Data Engineering*, pages 345–366, 2000.

[Har75] John A. Hartigan. *Clustering Algorithms*. John Wiley & Sons, New York, USA, 1975.

[HJ04] Charles D. Hansen and Chris R. Johnson. *The Visualization Handbook*. Academic Press, Oxford, United Kingdom, 2004.

[HK98] Alexander Hinneburg and Daniel A. Keim. An efficient approach to clustering in large multimedia databases with noise. In *Knowledge Discovery and Data Mining*, pages 58–65. AAAI Press, 1998.

[HKG00] Jiří Hladůvka, Andreas König, and Eduard M. Gröller. Curvature-based transfer functions for direct volume rendering. In *Proceedings of Spring Conference on Computer Graphics 2000*, pages 58–65, 2000.

[Hla01] Jiří Hladůvka. *Derivatives and Eigensystems for Volume-Data Analysis and Visualization*. PhD thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, 2001.

[Jai00] Lakhmi C. Jain. *Fuzzy Sets & Their Application to Clustering & Training*. CRC Press International Series on Computational Intelligence, Boca Raton, Florida, USA, 2000.

[JHG99] Bernd Jähne, Horst Haußecker, and Peter Geißler. *Handbook of Computer Vision and Applications*, volume 2. Academic Press, San Diego, California, USA, 1999.

[JMF99] Anil K. Jain, M. Narasimha Murty, and Patrick J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[KHK99] George Karypis, Eui-Hong Han, and Vipin Kumar. CHAMELEON: Hierarchical clustering using dynamic modeling. *IEEE Computer*, 32(8):68–75, 1999.

[Kin99] Gordon Kindlmann. Semi-automatic generation of transfer functions for direct volume rendering. Master's thesis, Cornell University, Ithaca, NY, 1999.

[KKH01] Joe Kniss, Gordon Kindlmann, and Charles Hansen. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *Proceedings of the Conference on Visualization '01*, pages 255–262. IEEE Computer Society, 2001.

[KKH02] Joe Kniss, Gordon Kindlmann, and Charles Hansen. Multidimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 8:270–285, 2002.

[KNT06] Jacob Kogan, Charles Nicholas, and Marc Teboulle. *Grouping Multidimensional Data. Recent Advances in Clustering*. Springer, Berlin Heidelberg, first edition, 2006.

[Koh95] Teuvo Kohonen. *Self-Organizing Maps*, volume 30 of *Springer Series in Information Sciences*. Springer, Berlin Heidelberg, 1995.

[KR90] Leonard Kaufman and Peter J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, New York, USA, 1990.

[KW03] J. Krüger and R. Westermann. Acceleration techniques for GPU-based volume rendering. In *Proceedings of the Conference on Visualization '03)*, 2003.

[KWTM03] Gordon Kindlmann, Ross Whitaker, Tolga Tasdizen, and Torsten Möller. Curvature-based transfer functions for direct volume rendering: Methods and applications. In *Proceedings of the Conference on Visualization '03*, pages 513–520, 2003.

[LA99] Bjornar Larsen and Chinatsu Aone. Fast and effective text mining using linear-time document clustering. In *KDD '99: Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 16–22. ACM Press, 1999.

[Lev88] Marc Levoy. Display of surfaces from volume data. *Computer Graphics and Applications, IEEE*, 8:29–37, 1988.

[LKM01] Erik Lindholm, Mark J. Kligard, and Henry Moreton. A user-programmable vertex engine. In *SIGGRAPH '01: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 149–158. ACM, 2001.

[LL94] Philippe Lacroute and Marc Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *SIGGRAPH '94: Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, pages 451–458. ACM, 1994.

[LM04] Eric Lum and Kwan-Liu Ma. Lighting transfer functions using gradient aligned sampling. In *Proceedings of the Conference on Visualization '04*, pages 289–296, 2004.

[MA96] John A. Markisz and Michael G. Aquilia. *Technical Magnetic Resonance Imaging*. Appleton & Lange, Stamford, Connecticut, USA, 1996.

[Mac67] James B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Symposium on Math, Statistics, and Probability*, volume 1, pages 281–297, 1967.

[MC98] Klaus Mueller and Roger Crawfis. Eliminating popping artifacts in sheet buffer-based splatting. *IEEE Visualization*, pages 239–246, 1998.

[MJ95] Jianchang Mao and Anil K. Jain. Artificial neural networks for feature extraction and multivariate data projection. *IEEE Transactions on Neural Networks*, 6(2):296–317, 1995.

[MK96] Geoffrey J. McLachan and Thriyambakam Krishnan. *The EM Algorithm and Extensions*. John Wiley & Sons, USA, 1996.

[Mue05] Klaus Mueller. Splatting and its applications on the body-centered cartesian (BCC) lattice. Point Lattices in Computer Graphics and Visualization - Tutorial at the Conference on Visualization '03, 2005.

[NH94] Raymond T. Ng and Jiawei Han. Efficient and effective clustering methods for spatial data mining. In *Proceedings of the Twentieth International Conference on Very Large Databases*, pages 144–155, 1994.

[PBT93] Nikhil R. Pal, James C. Bezdek, and Erik C. K. Tsao. Generalized clustering networks and Kohonen's self-organizing scheme. *IEEE Transactions on Neural Networks*, 4(4):549–557, 1993.

[PHBG09] Daniel Patel, Martin Haidacher, Jean-Paul Balabanian, and Eduard M. Gröller. Moment curves. In *Proceedings of the IEEE Pacific Visualization Symposium 2009*, pages 201–208, 2009.

[Pra01] William K. Pratt. *Digital Image Processing*. John Wiley & Sons, New York, USA, 2001.

[RSKK06] Christof Rezk-Salama, Maik Keller, and Peter Kohlmann. High-level user interfaces for transfer function design with semantics. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1021–1028, 2006.

[SBSG06] Petr Sereda, Anna Vilanova Bartroli, Iwo W. O. Serlie, and Frans A. Gerritsen. Visualization of boundaries in volumetric data sets using LH histograms. *IEEE Transactions on Visualization and Computer Graphics*, 12:208–218, 2006.

[SFGM93] Michael Stonebraker, Jim Frew, Kenn Gardels, and Jeff Meredith. The SEQUOIA 2000 storage benchmark. In *SIGMOD '93: Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, volume 22, pages 2–11, 1993.

[Sib73] Robin Sibson. SLINK: An optimally efficient algorithm for the single link cluster method. *The Computer Journal*, 16:30–34, 1973.

[SKK00] Michael Steinbach, George Karypis, and Vipin Kumar. A comparison of document clustering techniques. In *Proceedings of Workshop on Text Mining at the ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD 2000)*, pages 109–110, 2000.

[SML97] William Schroeder, Ken Martin, and Bill Lorensen. *The Visualization Toolkit: An Object-Oriented Approach to 3-D Graphics*. Prentice Hall, New Jersey, USA, second edition, 1997.

[STF+03] Iwo Serlie, Roel Truyen, Jasper Florie, Frits H. Post, Lucas J. van Vliet, and Frans Vos. Computed cleansing for virtual colonoscopy using a three-material transition model. In *MICCAI 2003: Proceedings of the 6th International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 175–183, 2003.

[SVG06a] Petr Sereda, Anna Vilanova, and Frans A. Gerritsen. Automating transfer function design for volume rendering using hierarchical clustering of material boundaries. In *Proceedings of the Eurographics/IEEE VGTC Symposium on Visualization 2006*, pages 243–250, 2006.

[SVG06b] Petr Sereda, Anna Vilanova, and Frans A. Gerritsen. Mirrored LH histograms for the visualization of material boundaries. *Vision Modeling and Visualization (VMV)*, pages 237–244, 2006.

[TK03] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition*. Academic Press, San Diego, USA, 2003.

[TM04] Fan-Yin Tzeng and Kwan-Liu Ma. A cluster-space visual interface for arbitrary dimensional classification of volume data. In *Proceedings of the Eurographics/IEEE VGTC Symposium on Visualization 2004*, pages 17–24, 2004.

[UEN09] Department of Computer Science Universität Erlangen-Nürnberg. The volume library. `http://www9.informatik.uni-erlangen.de/External/vollib/`, 2009. last accessed: 10.11.2009.

[Voo86] E. Voorhees. Implementing agglomerative hierarchic clustering algorithms for use in document retrieval. *Information Processing & Management*, 22:465–476, 1986.

[Wei07] Daniel Weiskopf. *GPU-Based Interactive Visualization Techniques*. Springer, Berlin Heidelberg, 2007.

[Wes89] Lee Westover. Interactive volume rendering. In *Proceedings of the 1989 Chapel Hill Workshop on Volume Visualization*, pages 9–16. ACM, 1989.

[Wes90] Lee Westover. Footprint evaluation for volume rendering. In *SIGGRAPH '90: Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, pages 367–376. ACM, 1990.

[Wik09] Wikipedia. K-means clustering. `http://en.wikipedia.org/wiki/K-means`, 2009. last accessed: 10.11.2009.

[WMLK89] Jerold Wallis, Tom Miller, Charles Lerner, and Eric Kleerup. Three-dimensional display in nuclear medicine. *IEEE Transactions on Medical Imaging*, 8(4):297–230, 1989.

[Zah71] Charles T. Zahn. Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Transactions on Computers*, C-20(1):68–86, 1971.

[ZRL96] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: An efficient data clustering method for very large databases. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 103–114, 1996.

[ZRL97] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery*, 1(2):141–182, 1997.