Chapter 4

Image Segmentation

4.1 Introduction

Many questions a scientist, engineer, or medical doctor may ask concerning a specific data set can be answered by applying appropriate visualization techniques and looking at the resulting images. This is true for both, qualitative as well as quantitative questions. In cases, however, where a computer aided measurement is needed, like the accurate computation of an organ's volume, an explicit representation of that organ is needed. The same is true if a numerical simulation based on the image data is to be performed which incorporates parameters that cannot be derived directly from the image intensity. Examples are bioelectrical fields relevant in EEG and ECG [49], mechanical properties in tissue deformation simulations for computer aided surgery [141], or electric properties for microwave hyperthermia treatment planning [140]. In all these cases an assignment of each voxel to classes of a suitable set of structures, like anatomic organs is needed. Beside the abovementioned application areas we will see how this can also be used for the generation of polygonal models, which are suited for fast display and data transmission.

In computer vision literature the process of computing such a classification is often divided into two steps: A *segmentation* which decomposes the image domain into subsets and a *labeling* that assigns structure names to these regions. This split-up, however, is only suitable for a certain set of methods. In this section we will discuss both aspects, since they are closely linked and not even separable sometimes.

During the last three decades hundreds of algorithms for different aspects of feature detection and image segmentation have been developed. A comprehensive review of segmentation methods is given in [92]. The authors show that the vast majority of the different algorithms that they discuss can be derived as special cases of a very general continuous optimization problem, the Mumford-Shah segmentation model [95, 18]. Let us consider one slice of a computer tomography shown in Figure 4.1. Even without anatomical knowledge the observer can identify four major region types which appear in different shades of gray, if a suitable mapping of the Hounsfield units to gray values is applied: Air (black), muscle tissue (dark gray), fat tissue (light gray), bone (white). According to a definition given in [92] a segmentation is a piecewise "smoothed" or piecewise constant image, as also shown in 4.1. This definition is driven by the idea that images that are to be segmented typically have smooth intensity variations within the objects which are

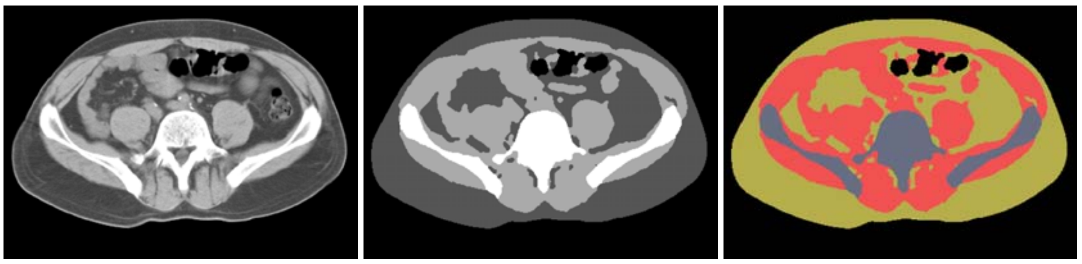

Figure 4.1: The left image shows a CT slice of an abdomen. The different image intensity values (gray levels) correspond to different tissue types. In the middle image the intensity value has been replaced with the average intensity for the tissue the voxel belongs to according to some given segmentation (not a threshold segmentation). In general, however, it is important to understand that two different regions in the segmentation do not necessarily correspond to image regions with different gray values. Also pixels belonging to the same region do not necessarily have similar gray values. In general the labels are unrelated to image intensities and depicting labels with colors as in the right image can help to avoid misunderstandings.

to be extracted and abrupt intensity changes at their borders. A large class of automatic and semi-automatic segmentation methods fall into the class of or can be interpreted as *variational methods*, also referred to as *active contour* models. One of the most prominent representants of this class are the *Snakes* type methods proposed by [53]. Variational segmentation methods employ a functional which measures the quality of a segmentation and try to optimize this value by finding a local or a global optimum. As an example consider the simplest form of the piecewise constant Mumford-Shah model. Let Ω be the domain of an image and $g : \Omega \rightarrow \Re$ be an image. Let $u : \Omega \rightarrow \Re$ be a regularized piecewise constant function (the segmentation) with a set of discontinuities $K \subset \Omega$ (the contours). Then the ideal segmentation according to this simple model would minimize the L2-functional

$$E(u, K) = \int_{\Omega \setminus K} (u - g)^2 dx + \lambda l(K). \tag{4.1}$$

$l(K)$ denotes the combined length of the contours K. The first term in the integral controls the similarity of the segmentation with the image and the second term enforces parsimony of boundaries. The parameter λ controls the total length of the contours, i.e. the degree of detail that should be represented in the segmentation. This is often referred to as *scale*, which is a parameter in many computer vision algorithms. We will discuss details of multi-scale image analysis in Section 5.2.1.

It is not surprising that a functional like Equation (4.1) is too simple to accurately detect the globally best segmentation of a complex image. Though already for this simple model it is difficult to find this global optimum. Therefore, various modifications have been used which take into account smoothness of the contours, coincidence of the set K with edges in the image as indicated e.g. by the intensity gradient magnitude.

There are many computer vision tasks, where specialized computer vision methods are successfully used for automatic segmentation. A general solution for the segmentation of medical or biological images, however, has not been found yet. And using an approach which is purely based on the image data itself it probably will not. The reason for this is that the best segmentation depends to a high degree on external expert knowledge and on the intended use of the segmentation. This influences for example the degree of detail that is considered optimal - often in a sense, however, that is more sophisticated than could be modeled with a single global scale parameter. Consider as an example a segmentation used for cancer treatment planning, where a slight shadow in an organ that is diagnosed as a tumor by a medical doctor would have to be considered a separate region, while the same amount of intensity variation in different parts of the body can be non-significant. Also general knowledge about the shape of a specific object can lead to different interpretations of which regions of similar gray value are to be segmented as one region.

Although significant effort has been made in recent years to build a sound mathematical fundament for various image segmentation algorithms (see e.g. [80]), the general problem of finding functionals that distinguish a correct from and inaccurate segmentation has not been solved in general.

These observations have lead to the development of model based or atlas based segmentation approaches, which is a very active field of research currently [21, 125, 131, 30, 54, 66]. These methods try to incorporate additional knowledge about the objects to be extracted, which is not present in the original image data into a segmentation algorithm. Often by imposing additional constraints on the set of possible segmentations (u, k), by providing initial values for local extremum searches, or by defining additional ene.g. terms in the segmentation functionals which are based on the similarity with an anatomical atlas. The model can be encoded into the algorithm explicitly, or it can be acquired by training the method with a set of segmentations that are known to be correct. It can be expected that model based segmentation will in the future provide solutions for many routine tasks which require a lot of human interaction today. The drawback, however, is that for every new task a new model has to be generated.

Today in practice the majority of image segmentation in medical and biological research is done completely manually or with semi-automatic methods that require a certain degree of manual intervention. For the discussed reasons this is not surprising and an abrupt change of this situation within the next few years is not very likely.

Therefore, the challenge for researchers working on practical solutions today is to develop suitable interaction paradigms. And the combination of automatic methods with interactive controls, which requires a mapping of the free parameters to intuitive values that can be controlled by the user - ideally geometrically by using the multi-dimensional input devices like a mouse.

The most important property of a *good* interactive segmentation system is simple but crucial: It is guaranteed that the desired segmentation can always be achieved. The better the integration of semi-automatic tools is, the faster this segmentation can be achieved.

We have developed and implemented such a system, which has been tested in numerous different application areas and has proven to be a rather universal and efficient tool. It is part of the Amira visualization system today. A complete description of all its details is not in the scope of this work and we refer to [147, 40]. Here we will limit ourself to the basic concepts and some of the scientific contributions contained therein.

4.2 Interactive Segmentation

Two basic prerequisites for an efficient interactive segmentation environment are a good method of interaction and a tight perceptual coupling with the user. We have discussed that in Chapter 3.

Clearly the most wide-spread and most efficient multi-dimensional input devices are computer mice and graphic tablets. These devices have essentially two degrees of freedoms (DOF). This matches well with the two dimensional output devices normally used and the two dimensional human retina, as discussed in Section 3.2.

Therefore, we have decided to use two dimensional interaction in slices to be the basis of our segmentation system. In order to benefit from the three dimensional nature of the data and not be obliged to segment each slice independently, working in orthogonal slices is possible and a true three dimensional visual control is available. To exploit three dimensional coherency of the data sets we have implemented interpolation methods.

The basic in-slice interaction tools allow to select voxels which then can be assigned to or subtracted from a specific label. The most important are

- **The Brush** selects on mouse click pixels which are within a user-defined radius around the mouse cursor.
- **The Lasso** lets the user draw a closed contour, and selects all pixels within the contour. We have added an Intelligent Scissor mode [93, 122], which adopts the contour to automatically detected image contours. We will go into more detail on this in Section 5.1.
- **The Magic Wand** is a region growing algorithm, which starting from a seed point will select all connected voxels with a gray value in a given tolerance interval. Further options like exclusion of previously labeled regions can be chosen.
- **The Blow tool** is based on a level set method [113]. Here we place a small circle on mouse click and then extend its contour as the user moves the mouse. Each contour point is allowed to move in the local contour normal direction. An underlying function, the *speed function*, which is derived from the image data, controls the local velocity. This way the contour locally decelerates when it reaches image contours.

Some of these tools are depicted in Figure 4.2. Both, on the set of selected pixels, as well as on sets of already labeled pixels various image processing filters can be applied, like erosion, dilatation, smoothing etc. [107, 41].

These tools allow for the segmentation of virtually any structure that can be seen in the images. We have experienced that many of our users in biomedical application areas were able to save 90% or more of time needed for segmentation when using this framework. Before, they used less specialized tools like *Photoshop* for segmentation.

In many cases an additional order of magnitude of efficiency can be gained, by combining these tools with methods for inter- and extrapolation. We will detail this in the next sections.

4.3 Interpolation and Extrapolation

Interpolation and extrapolation is an important concept for an interactive image segmentation environment. It can help to reduce the amount of manual work in regions with little changes and it can be used to compute initial guesses for a local interactive or automatic optimization. The interpolation of shapes is not trivial.

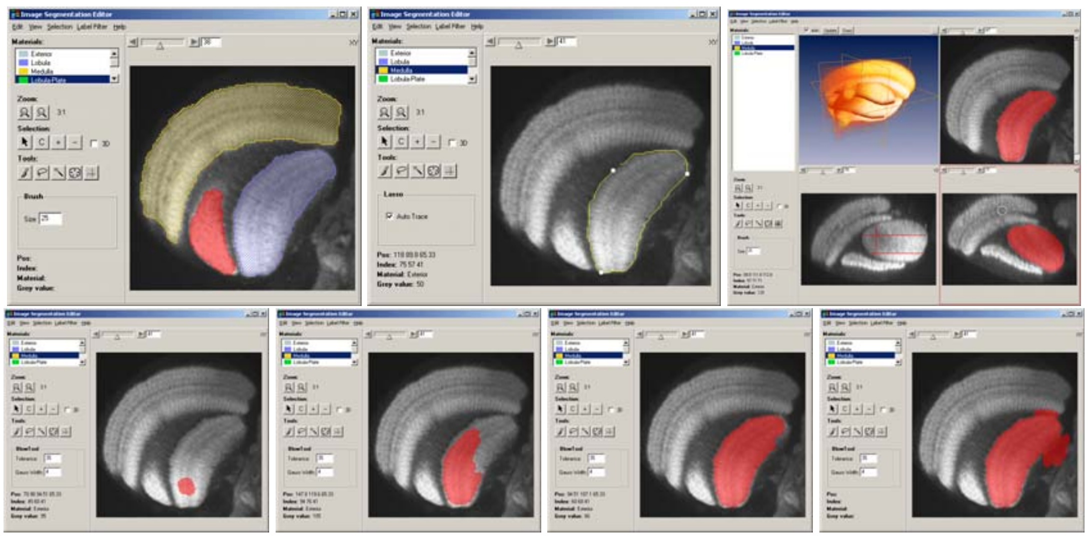

Figure 4.2: The image segmentation editor. Pixels are interactively selected (red part) and then assigned to specific structures. Already labeled pixels can be depicted in various styles, like hatched in image (i). The lasso tool with *Intelligent Scissor* option enabled automatically adjusts a contour to edges in the image. In image (ii) only three mouse clicks were necessary (white dots) to define the contour. Simultaneous display of orthogonal views (iii) allows better understanding of the three dimensional relationship. Here two slices in one orientation and one slice in an orthogonal orientation were labeled. In the third orientation these appear as lines. A preview in the 3D viewer shows the selection in combination with a transparent volume rendering. The lower row shows the blow tool. *Lower row:* Starting at a seed point (iv) the user can blow a contour by moving the mouse (v). Ideally it will reach a state were the contour is adjusted to the structure to be selected (vi). If the contrast is too low or if the mouse is moved too far, the contour will break out (vii). In the latter case the mouse can simply be moved back again.

A wide class of methods work on a parameterization of the boundary of the objects to be interpolated: A curve for two-dimensional, a surface for three-dimensional objects.

For the two-dimensional case this could look like this: Let the interval $P = [0 \ldots 1]$ be the parameter domain and let $K_1, K_2 \subset \Re^2$ be two closed intersection-free curves to be interpolated. Let $k_1 : P \to K_1$ and $k_2 : P \to K_2$ be parameterizations of the intersection-free curves, i.e. k_1 and k_2 are bijections. A parameterization of two objects onto the same parameter domain implicitly defines a correspondences $C : K_1 \to K_2$ between the two objects, by

$$C(\vec{x}) = k_2\left(k_1^{-1}(\vec{x})\right)$$

k_1^{-1} denotes the inverse of k_1, which exists since k_1 is bijective. Using the parameterization it is easy to construct a family of curves which interpolate between K_1 and K_2, e.g. linearly:

$$\begin{aligned} \hat{K}_\lambda &: P \to \Re_2 \\ \hat{K}_\lambda(t) &= (1-\lambda)k_1(t) + \lambda k_2(t) \qquad (4.2) \end{aligned}$$

The curve family $\hat{K}_\lambda$ is an interpolation, i.e. $\hat{K}_{\lambda=0} \equiv K_1$ and $\hat{K}_{\lambda=1} \equiv K_2$ and $\hat{K}_\lambda$ is continuous (C^0). Using the correspondence C we can even keep track of an individual point on the curve K_1, which will move on a straight line to its corresponding point on K_2. If parameters $\lambda < 0$ or

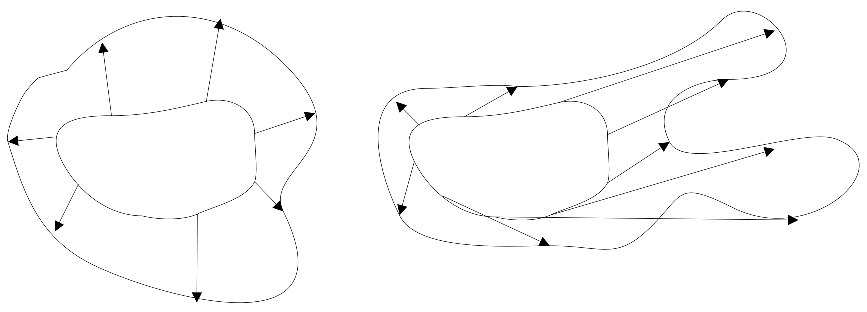

Figure 4.3: Correspondence definition and inter-/extrapolation based on arc-length segmentation can fail, even if a suitable origin $t = 0$ is found. The figure illustrates this qualitatively. While in the left image the correspondence between the two contours is acceptable, one would probably like the right part of the smaller contour to be "stretched" stronger than the left part.

$\lambda > 1$ are used, the same formula can be used for extrapolation. The question whether $\hat{K}_\lambda$ is a *good* interpolation depends to a high degree on the parameterization. Figure 4.3 illustrates that a simple arc-length parameterization is not always suitable.

For two dimensional surface in three dimensional space the situation is much more difficult. Already the choice of a suitable parameter domain depends on the topology of the objects, examples could be the unit square, the unit-sphere, or a torus. In [152] we have presented a method which decomposes the objects into patches that can be mapped to the unit circle or a unit-cylinder. User input is used to assure that the correspondence is correct in a geometric and semantic sense. See the references in [152], and the review article [67] for related work. The method has been used successfully for generation of morphing sequences for animation. For an interpolation tool, however, the required user interaction reduces the time gain, that is expected from the interpolation, too much to be efficient.

Both, the two-dimensional as well as the three-dimensional version of parameter based interpolation suffer not only from the problem of finding the correct parameterization but the main problem is the requirement of equal topology of K_1 and K_2, which is a prerequisite for finding a common parameter domain. If for example K_1 is a single closed loop and K_2 contains two separate loops, it is not obvious how a meaningful correspondence can be established in general.

Therefore, we have taken a different approach, which solves these problems in an elegant way. Instead of working on the object boundary, an embedding is constructed which contains the object boundary as a level set. The most frequently used embedding in this context is the distance map which defines for each point the distance to the nearest boundary point. Typically with a negative sign for points inside the object and positive sign outside. Some of the numerous works related to this type of interpolation are [70, 69, 19, 126, 33].

In the following we will review the important concept of (signed) distance maps, show how they can be used for shape interpolation and then point out the particular problems that occur if they are used in an image segmentation system and how we have solved these problems.

4.3.1 Distance Maps

Distance maps are a powerful tool and they will be used at several places throughout this work. A good review of the various concepts and algorithms is given in [22].

Let Ω be the domain of an image, which contains a binary object O. Then the distance map $\Omega \rightarrow \Re$ assigns to each pixel the distance to the nearest boundary point of δO:

$$D(\vec{p}) = \min\{\text{dist}(\vec{p}, \vec{q}), \vec{q} \in \delta O\}, \quad \vec{p} \in \Omega$$

In order to distinguish the points inside the object from those outside, often a negative sign is assigned to the interior part of the distance map. A signed distance map has been shown in Figure 3.4.

In this section we will focus on the case that the object O is given in a discrete way, more specifically as a set of voxels. In order to measure a distance, a metric has to be supplied. Of course the Euclidean metric is the most natural one, but in practice often other metrics are used, which approximate the Euclidean metric, but are faster to compute. Frequently used metrics that we will discuss are

$$\begin{aligned} \text{dist}_{\text{euclidean}}(\vec{p}, \vec{q}) &= \sqrt{\sum_i |p_i - q_i|} && (4.3) \\ \text{dist}_{\text{manhattan}}(\vec{p}, \vec{q}) &= \max\{|p_i - q_i|\} && (4.4) \\ \text{dist}_{\text{chamfer}}(\vec{p}, \vec{q}) &= 1 \cdot \max\{|p_i - q_i|\} + \sqrt{2} \cdot \min\{|p_i - q_i|\} \qquad \text{(2D case)} && (4.5) \end{aligned}$$

The first metric is the standard Euclidean distance. The second one is the max-norm (l_∞). On a discrete grid it is identical to the number of strictly vertical or horizontal steps that are needed to walk from $\vec{p}$ to $\vec{q}$. The third metric additionally allows diagonal steps with the respective weighting. Other chamfer metrics exists, which use different weighting coefficients or allow more directions, as we will see.

An ad-hoc algorithm for the computation of a distance map can be given easily: Compare each voxel in the domain Ω with each boundary voxel of O (a discussion about the correct boundary treatment in the discrete case will follow). If we assume that the object surface grows with the second power of the linear dimension in three dimensions, this algorithm has a complexity $O(n \cdot n^{2/3}) = O(n^{5/3})$ for the three dimensional case and $O(n^{3/2})$ for the two dimensional case, where n is the total number of voxels in the domain.

If a chamfer-type metric is used, the distance value at a given point can be deduced from the distance value of one of its neighbors. Therefore, it is possible to implement more efficient algorithms: First all the points on the border are initialized with distance value zero. Then each voxel propagates its distance to its neighbors adding the appropriate weight. A voxel can receive propagated distance values from several neighbors, but only the propagation resulting in the lowest distance value is kept. If the propagation starts with the zero-valued boundary voxels and then processes the remaining voxels in order of increasing distance value, each voxel has to propagate only once. Therefore, an $O(n)$ algorithm can be realized (we will discuss this in more detail for a similar problem in Section 5.1.1). However, the bookkeeping of the current propagation front can be quite memory intensive and such an algorithm is not cache-local.

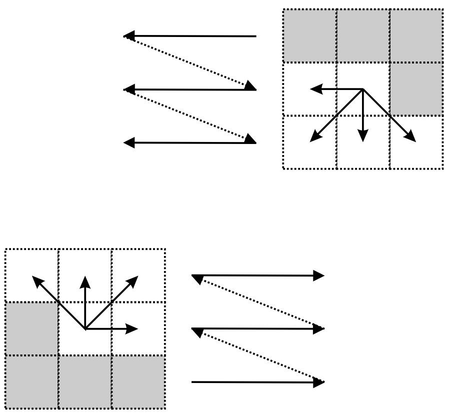

Figure 4.4: Propagation of distance values with chamfer metric illustrated in 2D. Two passes are performed in scanline order. In the first pass the scan starts at $(0, 0)$, advancing first in x, then in y direction. In the second phase the scan is done in the exact opposite order. In each scan the distance value of each voxel is propagated to 4 of its neighbors, after it has been increased with offsets according to the mask depicted in the image. In the second scan, a value is only propagated if that would lower the value assigned so far.

It has been shown that it is possible to propagate the distance values from the image border to the whole domain with a much simpler processing scheme. Two passes in scanline order with opposite direction are sufficient [11]. This is depicted in Figure 4.4. If a distance map with a chamfer metric is used to approximate the Euclidean metric, a systematic error is made. One can try to minimize this error by choosing a different weight in Equation (4.5). The maximal possible relative error can be reduced to $\approx 4\%$ by using a weight of 1.351 instead of $\sqrt{2}$. For the three dimensional case optimal results are obtained with the weights $(1, 1.314, 1.628)$. The maximal error is then $\approx 6\%$. Instead of using a $3{\times}3$ neighborhood in the propagation a larger ask can be used, thus allowing not only vertical/horizontal and diagonal steps, but e.g. using an $5{\times}5$ mask, thus further reducing the error. See [12] for details. Note that in a practical implementation, often 8 or 16-bit integer variables are used to store the distance map, to save memory and computation time. In these cases the offsets in the chamfer mask have to be replaced by integer values. Frequently used values for the $3{\times}3{\times}3$ case are $(3, 4, 5)$, approximating $3 \cdot (1, \sqrt{2}, \sqrt{3})$.

As already stated above the distance value with respect to a Euclidean norm can *not* be derived from the distance values of its neighbors. Therefore, Danielsson [24] has proposed to not propagate the distance value, but the difference vector pointing to the nearest boundary point. Then of course the Euclidean distance to this point can be calculated exactly. It is not a priori clear whether this is always the nearest point though. This would only be the case if the following statement was true: *For any given point the nearest boundary point is the same as for at least one of its neighbors.* This is true for the chamfer metric and for the Euclidean metric in the continuous case but not for the discrete case. However, the maximal relative error that is made with a $3{\times}3$ propagation scheme is only $\approx 0.3\%$, compare [24, 71, 22].

4.3.2 Distance Based Interpolation

From a distance map, the original object boundary can be extracted (approximatively in the discrete case) by computing the zero level set, e.g. using the marching cubes algorithm. A label representation can be computed by simply selecting all voxels with negative distance value. The idea of distance based interpolation is to compute the distance fields of the two objects to be interpolated, point-wise interpolate these fields and finally re-extract the object. The interpolated object for interpolation parameter $\lambda = 0 \dots 1$ is:

$$O_\lambda = \{\vec{p} \in \Omega | (1-\lambda) D_1(\vec{p}) + \lambda D_1(\vec{p}) < 0\}$$

where D_1 and D_2 are the distance maps of the two objects.

The method is elegant. It automatically handles topological changes and no parameterization has to be computed. In some applications it is a disadvantage, that no explicit correspondence is established by the method, for example in graphics and animation where often additional per-vertex properties like texture coordinates or material properties have to be preserved and interpolated. In our application this is not a problem.

To improve the quality of the results, several authors have combined the distance based interpolation with a previous rigid or other coarse warping transformation [19, 69].

4.3.3 Interpolation for Segmentation

A useful application for interpolation in an image segmentation environment is the interpolation of labels between (almost) consecutive slices, i.e. between two two-dimensional objects. This is desired since in regions of little change often sufficient accuracy can be achieved by labeling a structure only in every n-th slice and interpolating for the in-between slices.

Also it is often helpful to insert additional slices for subsequent algorithms in data sets which have a large slice distance compared to the resolution within a slice.

A third use of such a method is the extrapolation of a labeling to a slice that has not been segmented so far. The extrapolated result could be adjusted then, using manual or automatic methods, if necessary.

We have implemented a distance based interpolation method and used it for our segmentation framework. We have observed two problems: If used to interpolate between multiple slices with large separation, using C^0-elements does not suffice to represent organic shapes. The second problem is related to boundary effects. In the following will show how we have extended the method to higher order continuity and show comparing results. Then we will illustrate this boundary problem and propose a solution.

4.3.4 Higher Order Interpolation

As can be seen in the left two images of Figure 4.5 the interpolation as described so far produces reasonable results. However, the linear type of Equation (4.2) results in surfaces which are continuous but not C^1 in interpolation direction. Therefore, we propose to use local cubic Hermite interpolation instead [26]. Here a piecewise cubic polynom is constructed from the function values and the first derivatives. Lets assume that we have to construct an interpolation involving n equidistant slices. Let the corresponding distance maps be $D_i(\vec{p})$. These are the function values

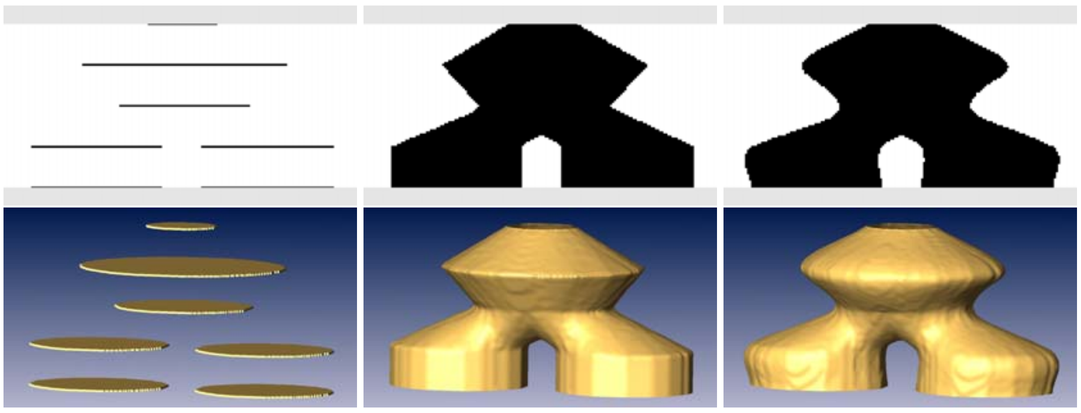
Figure 4.5: Interpolation demonstrated with an artifical test object. Only every 10th slice is labeled. A slice orthogonal to the labeling orientation is shown in the top row. The lower row shows the same data set in three dimensions. Linear interpolation yields correct but unorganic results (center). The results can be significantly improved by using cubic Hermite interpolation (right).

to be interpolated. We now ask the derivative to be

$$D'_i(\vec{p}) = \begin{cases} (D_{i+1}(\vec{p}) - D_{i-1}(\vec{p}))/2 & ; i = 2 \dots N-1 \\ D_2(\vec{p}) - D_1(\vec{p}) & ; i = 1 \\ D_N(\vec{p}) - D_{N-1}(\vec{p}) & ; i = N \end{cases} . \tag{4.6}$$

For the interpolation we use the Hermite polynoms which have the following defining properties [26]:

$$\begin{array}{llll} H_0^3(t_0) = 1, & \frac{d}{dt}H_0^3(t_0) = 0, & H_0^3(t_1) = 0, & \frac{d}{dt}H_0^3(t_1) = 0 \\ H_1^3(t_0) = 0, & \frac{d}{dt}H_1^3(t_0) = 1, & H_1^3(t_1) = 0, & \frac{d}{dt}H_1^3(t_1) = 0 \\ H_2^3(t_0) = 0, & \frac{d}{dt}H_2^3(t_0) = 0, & H_2^3(t_1) = 1, & \frac{d}{dt}H_2^3(t_1) = 0 \\ H_3^3(t_0) = 0, & \frac{d}{dt}H_3^3(t_0) = 0, & H_3^3(t_1) = 0, & \frac{d}{dt}H_3^3(t_1) = 1 \end{array} \tag{4.7}$$

The interpolating polynom in the interval $t_0 \dots t_1$ is then given by

$$P(t) = f(t_0)H_0^3(t) + f'(t_0)H_1^3(t) + f(t_1)H_2^3(t) + f'(t_1)H_3^3(t) \tag{4.8}$$

as can be seen by inserting t_0 and t_1. For equidistant slices we can use $t_0 = 0$ and $t_1 = 1$. The Hermite polynoms then are

$$\begin{array}{rcl} H_0^3(t) & = & 2t^3 - 3t^2 + 1 \\ H_1^3(t) & = & t^3 - 2t^2 + t \\ H_2^3(t) & = & -2t^3 + 3t^2 \\ H_3^3(t) & = & t^3 - t^2. \end{array} \tag{4.9}$$

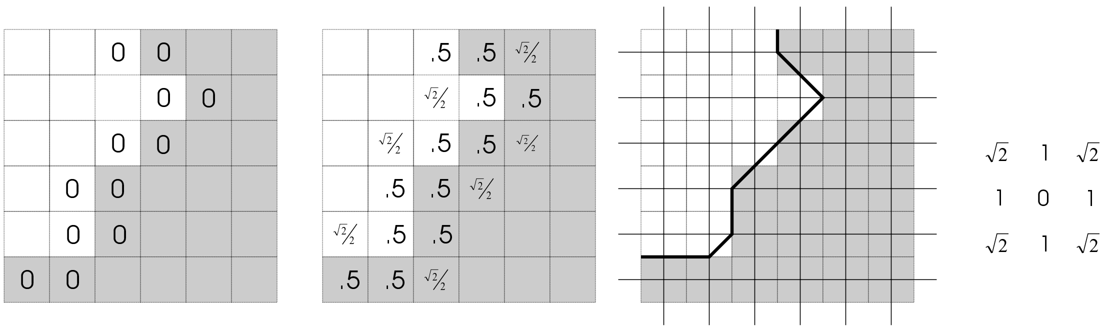

Figure 4.6: Boundary initialization for distance map propagation inside and outside the object. Grey pixels are object voxels. The initialization in the left image is relatively inaccurate. In the second image half the offsets from the $3{\times}3$ chamfer mask are used. The third image displays a boundary contour that corresponds to such an initialization. Note that the contour intersects the edges of the dual grid always in the middle.

4.3.5 Border Initialization

We have discussed in the previous sections how the distance values are propagated from one pixel to the next and what errors can occur. Such algorithms, however, require an initialization of the boundary voxels to start with. In many implementations simply all object voxels, that have at least one background voxel as neighbor, are considered to be part of the boundary and initialized with zero. This is not compatible with the usual interpretation of an object in a segmentation framework. Here we would expect the object boundary to pass somewhere between such boundary object voxels and the neighboring background voxels. Therefore, the distance value at the centers of the boundary object values is different from zero. Implementations taking this into account typically initialize the boundary with half the values from the chamfer mask, depending on whether the neighboring voxel is in the 4 or 8-neighborhood (6,18,26 in 3D). This is illustrated in Figure 4.6. Since in general we do not have any information about where the boundary passes exactly, this type of initialization is not a priori wrong. The resulting contour is compatible with the labeling according to the following definition:

A contour k is compatible with a labeling, if all voxels centers labeled inside lie on one side of the contour and all voxel centers labeled outside lie on the other side.

However, the contour in Figure 4.6 corresponds to a staircase-like boundary (with four different directions of the steps).

Looking at Figure 4.7 we see that the iso-line for iso-value zero (the one which is drawn stronger) is rather jerky. This is due to this effect. For increasing distances distance maps have a smoothing property. Therefore, the boundary aliasing is not a problem for the majority of distance map based algorithms. In our case, however, it is, because an extrapolation is often performed from two consecutive and therefore very similar slices, the contour of the resulting object will be close to the contours of the original objects. Therefore, it will be computed from the distance values in that jerky region, resulting in undesired artifacts.

As stated above we do not know where the boundary of the real object passes exactly. However, we do have significant reason to believe that the boundary of an organic object does not look like a staircase. Instead, we would prefer to choose from all compatible contours the one which is as

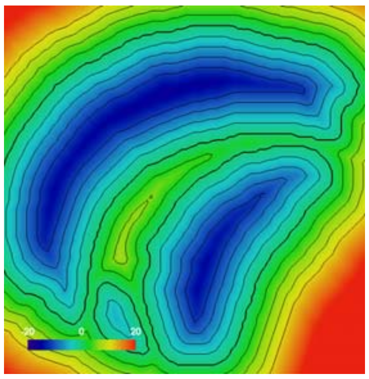

Figure 4.7: Signed distance map, shown with pseudo coloring and iso-lines. The strong line corresponds to zero distance. The distance map gets smooth with increasing distance, but at the original contour, the staircase artifacts can occur due to binary boundary initialization. This can cause aliasing for extrapolation.

smooth as possible. Of course there may be details in the object which are not resolved this way, but this is a general limitation of the finite image resolution.

In order to find "the smoothest" curve we define a functional based on the curvature. Let $k : [0 \ldots 1] \rightarrow \Re^2$ be a parameterization of the object boundary. Let k be proportional to the arc-length parameterization, i.e. $\partial k(t)/\partial t =$ const.. For the following discussion we assume that the contour consists of only one closed loop. If this is not the case the loops are processed independently. The functional that we strive to minimize is:

$$E(k) = \int_0^1 |k''(t)|^2 ds$$

k'' the second derivative of k.

This is a standard problem of variational calculus. The solution of the problem

$$E(k) = \int_0^1 f(s, k, k', k'', ..., k^{(n)}) ds \overset{!}{=} \text{Extremum}$$

must satisfy the Euler-Lagrange differential equation:

$$f_k - \frac{d}{ds} f_{k'} + \frac{d^2}{ds^2} f_{k''} - + \ldots + (-1)^n \frac{d^n}{ds^n} f_{k^{(n)}} = 0 \,.$$

In our case the Euler-Lagrange equation results in

$$\begin{aligned} \frac{\partial f}{\partial k} - \frac{d}{ds}\left(\frac{\partial f}{\partial k'}\right) + \frac{d^2}{ds^s}\left(\frac{\partial f}{\partial k''}\right) &= 0 \\ \Leftrightarrow \frac{d^2}{ds^2} k''(s) = 0 \Leftrightarrow k^{(4)}(s) &= 0 \,. \end{aligned}$$

In order to find an optimal contour which is compatible with the labeling we introduce an additional parameter t which denotes an artificial iteration time in an iterative process of finding the sought smoothing effect:

$$\frac{\partial k(s,t)}{\partial t} + k^{(4)}(s,t) = 0$$

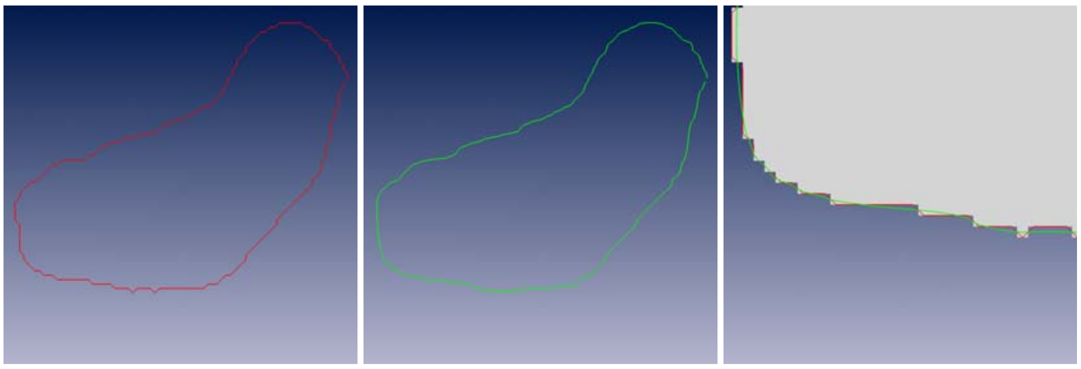
Figure 4.8: A binary border initialization as used in standard distance map algorithms is not suitable for extrapolation, because it implicitly defines a jerky contour (left). After smoothing the contour is still compatible with the labeling (right) but significantly smoother (center).

If the iteration converges then $\partial k(s,t)/\partial t$ becomes zero and the solution is identical to the static case.

In order to solve this differential equation, we have to discretize it. The parameter s is discretized into N discrete steps s_i. We define $k_i = k(s_i)$. If the s_i are equidistant with $h = s_{i+1} - s_i$ the fourth order derivative is approximated using central differences:

$$k_i^{(4)} \approx \frac{k_{i+2} - 4k_{i+1} + 6k_i - 4k_{i-1} + k_{i-2}}{h^4}, \qquad i = 1, \ldots, N$$

Also the time is discretized with constant time steps τ, the corresponding values of k are denoted k^n. The derivative $\partial k(s,t)/\partial t$ can then be approximated by:

$$\frac{\partial k(s,t)}{\partial t} \approx \frac{k^{n+1}(t) - k^n(t)}{\tau}$$

Putting all this together, we end up with an explicit Euler scheme [25]:

$$k_i^{n+1} = k_i^n - \tau \frac{k_{i+2}^n - 4k_{i+1}^n + 6k_i^n - 4k_{i-1}^n + k_{i-2}^n}{h^4}, \qquad i = 1, \ldots, N$$

In order to satisfy the constraint of compatibility with the labeling, we proceed as follows. We start with an initial contour as shown in Figure 4.8. As can be seen the vertices of the contour lie always in the middle of the edges of the dual grid. For each vertex we store this edge and restrict them to remain on this edge after each iteration. This guarantees the compatibility according to the above definition. In fact this is even a little stricter than would have been necessary, but this has not posed any problem.

Typical contours contain at most a couple of hundred vertices. Therefore, each iteration step can be computed very quickly. Typically after 100 iterations the solution has been converged sufficiently well. Results are shown in Figure 4.8. Once the smooth contour is found, the distance values of the pixels intersected by the contour can be set to the exact distance of this pixel to the contour. Also computing a distance map at a higher resolution than the original image is possible this way, since the contour is given explicitly now.

4.3.6 Non-Planar Interpolation

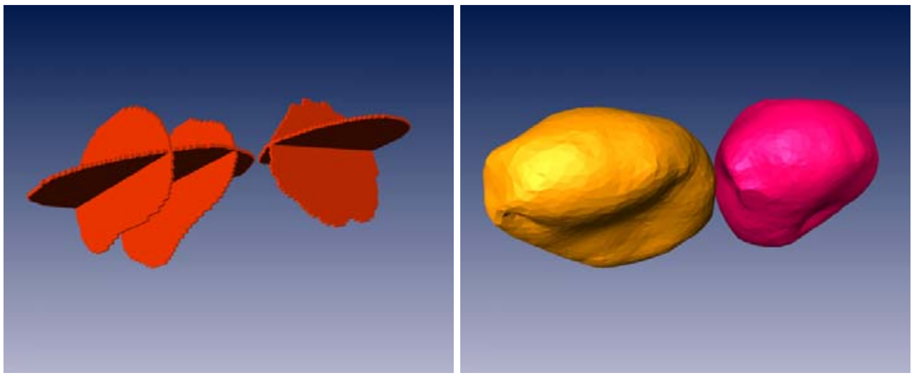
Figure 4.9: For smooth structures or if only an approximative segmentation is needed, very few, potentially non-orthogonal, slices are sufficient (left image) to define the structure sufficiently accurate, after the wrapping based on implicit surfaces is applied. Example for two glomeruli in the honey bee's antennal lobe.

Another useful technique is the segmentation and interpolation in non-parallel slices. Instead, one or more slices are labeled in two or all three orthogonal plane orientation, yielding a "skeleton" of the desired structure (compare Figure 4.9). In order to reconstruct the full object from this skeleton, a technique presented in [126] for shape morphing applications is used. Again, it is based on the idea of constructing an embedding. In [126] a scattered data interpolation approach is suggested. Points are distributed equally on the contours and assigned a function value 0. By subtracting at each point the local normal of the contour, new points lying a little bit inside the contour are computed and a function value of -1 is specified. Then the function values are interpolated by constructing an interpolant based on a linear combination of thin-plate splines. This technique will be described in more detail in Section 7.1.2.

It is expected that the interpolant is negative within the objected to be segmented and positive outside. Its boundary surface can be computed then by extracting the 0-iso-surface, a labeling by thresholding with threshold 0. Figure 4.9 shows results.

4.3.7 Results

We have presented a new method for border initialization and employed cubic splines to improve the quality of distance map based shape interpolation and extrapolation. We have implemented and applied the described interpolation methods in a number of different applications. Interpolation has proven to be a very efficient tool to speed-up the interactive segmentation process in certain areas or for certain structures, or when pixel-exact accuracy is not needed. It can also serve as initialization for automatic segmentation algorithms like Snakes. Interpolation is also used to increase resolution in z-direction for non-isotropic data sets if subsequent algorithms require isotropical voxels. It has turned out that the tri-linear interpolation significantly increases quality here, as can be seen in Figure 4.10. An example where non-planar interpolation has proven to be efficient is the segmentation of glomeruli in the bee antennal lobe, as shown in 8.4.

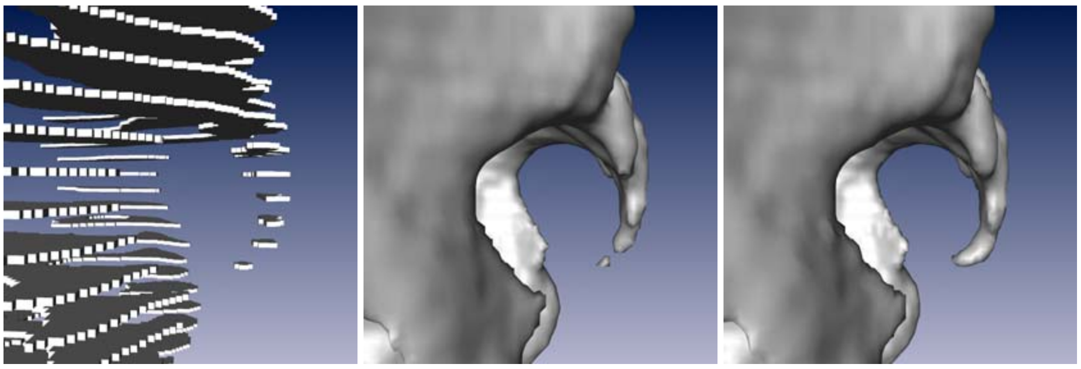

Figure 4.10: Interpolation in a medical recording of the sacrum. The data set has a large slice distance. Left image shows the labeling in the original slices. Central image shows a surface extracted after inserting four additional slices between each two input slices using linear distance map interpolation. Right image corresponds to cubic Hermite interpolation.

4.4 Sub-Voxel Accuracy

Advanced painting programs, like *Adobe Photoshop* or *Gimp* provide brush types that use transparency at the boundaries to avoid aliasing effects. We have developed a set of tools for interactive segmentation, that does not only modify the labeling, but simultaneously edits a set of weights, that can be interpreted as the likelihood that the labeling is correct. By interpolating these weights exact intersection points for interfaces between two materials can be computed. We have described this method in [149] and shown that with the appropriate choice of weights, smooth boundary surfaces can be extracted. We have also described how standard tools like *Lasso, Brush*, or *Magic Wand* can be defined to produce these weights. One problem, however, is, that not all segmentation tools can be generalized this way. Additionally if such tools are applied subsequently in the same region, e.g. when changing or correcting a previous labeling several times, the resulting weight fields are not always as smooth as expected. A probably more flexible way to achieve the same goal of smoothness is to stick with a binary labeling in the interactive stage and then use the *Constrained Smoothing* method that we describe in Section 6.3.2 to generate smooth consistent surfaces.

Up to here we have discussed the problem of generating smooth contours and surfaces from a per-voxel labeling. Another aspect would be to extract the shape of the physical objects depicted in the images with a spatial accuracy that is better than a voxel size.

Computer vision systems, for example used in optical tracking systems, are able to compute the center of a white sphere on black background recorded with a video camera with an accuracy that is a small fraction of the pixel size. And also the human perceptual system is able to detect misalignments in the sub-pixel range in medical images. How this can be used in a practical segmentation system, however, is subject to future research. Also the conceptual ease and consistency of a label representation compared to a description based on contours suggests that even if real sub-voxel-accurate interactive tools were found, the best underlying representation might still be a uniform or adaptively refined label field with higher resolution than the image data. Again, the final answer to this question has not been given yet.

4.5 Results

We have designed and implemented the above described methods in one common environment, and found that this approach to interactive segmentation is very efficient. All the segmentations and labelings shown in this work are generated using this tool. The concept can easily be extended by other slice-based or volume-based tools, like a projection based volume editor, that we will demonstrate in Figure 5.6 in Section 5.2.