Chapter 5

Line Extraction

Beside the structures with a voluminous shape as discussed in the previous section, structures with completely different shape properties like the dendritic trees of neurons are of interest in this work. These, however, require different segmentation approaches as we will see in the following.

In Figure 3.5 it is obvious, that the shape of the neuron is almost impossible to perceive in a slice. It is even hard to distinguish the little dots stemming from the section of the dendrites with the slicing plane from noise. Therefore, it is clear that an interactive method relying on a slice-based visualization cannot be efficient. The tools discussed in the previous section where designed to define volumes (or areas per slice) in order to assign them to a specific structure. However, dentritic structures have a very small volume and it seems to be much more natural to base an interactive definition on the center lines and branching points of the tree.

Not only interactive but also automatic segmentation approaches are strongly adopted to the structure of the objects to be segmented. While the algorithms for segmentation of voluminous objects try to find an optimal boundary curve (2D) or surface (3D), this is not suitable for structures which often have a thickness of only one voxel. Here algorithms which focus on the extraction of center lines are more appropriate as we will see.

In this section we will discuss automatic and interactive approaches to the extraction of center-line graphs from image data sets showing dendritic trees or vascular networks. While for the structures discussed in Section 4 the image segmentation and the geometry reconstruction step could be well separated from each other, we will see that for line-type structures these stages will be much more toothed.

5.1 Interactive Extraction

In this section we present a method for the interactive extraction of (the centerlines of) line-like structures. Inspired by the *Intelligent Scissor* technique originally proposed by Mortensen and Barrett [93] for contour tracing in two dimensional images, we develop a tool that allows quick and accurate interactive extraction of graphs from three dimensional image volumes.

As pointed out earlier the key to any interactive data manipulation algorithm is a good perceptual coupling between the user and the computer. As we have explained in section 3 and see in Figure

3.5 the maximum intensity projection (MIP) is especially well suited for the type of data we have to treat here. Our goal is to extract a graph-type representation consisting of the centerlines of these structures.

In the following we will first describe the methods we have developed to segment the lines in the two dimensional MIP and then discuss how the third dimension that has been lost in the projection step can be recovered.

The approach we have taken for the interactive tracing of line-type structures is inspired by snakes [53] and intelligent scissor [93, 122] methods for segmentation of voluminous objects. In both cases a rough approximation of the contour, specified interactively by the user is refined by an optimization procedure that takes into account the image information.

Snakes methods take a user specified coarse contour as the initial condition for an iterative energy minimization procedure that takes into account image based as well as locally and globally defined shape based energy terms. The disadvantage is that the user does not know what level of accuracy for the initial contour is required, in order to achieve accurate results. There is no interactive control over the method after the specification of the initial contour.

Intelligent scissor class methods overcome this limitation. They compute piecewise optimal paths between two interactively specified points called seed point and free point. Due to the properties of the cost function and path definition that will be discussed in the following sections, the computation can be performed in real time. While the free point is moved, the corresponding optimal path segment is displayed. This way the user can simply optimize the path segment by moving the mouse until the result is satisfactory.

5.1.1 Line Extraction in MIP

In our method the user successively defines points on the structure to extract in the MIP image. The algorithm connects them with the optimal path. The costs of a path are defined based on the local image characteristics. The path is defined in a discrete way by a series of neighboring pixels. The problem of path extraction can be formulated as a graph search problem, where the pixels are nodes of the graph and the links between neighboring pixels are the graphs edges. To each edge costs are assigned. We will now describe the computation of the optimal paths, followed by a discussion of the cost function.

Path Finding

Once the user has selected the first point of a structure, the optimal paths from all image pixels to that seed point are computed simultaneously using the Dijkstra algorithm [116, 134]. Then as the user moves the mouse about to set the next point, the cheapest path connecting the current mouse position with the seed point can be displayed in real time.

For the initial computation of the shortest path from any pixel to the seed point s the algorithm starts from s and iteratively propagates the current costs from one pixel to its neighbors.

Each node u has a value $C(u)$ associated, which specifies the costs for the cheapest path to s known so far. We call a node permanent once we know that the cheapest path from s to u has been

found, i.e. $C(u)$ will not change anymore. Initially we set $C(u)$ to infinity except for the start point which is assigned zero.

All nodes u that are not yet permanent and have finite costs (initially only s) are maintained in a list L, sorted according to $C(u)$. From the list we remove the node u with minimal $C(u)$, and check for all its neighbors v_i whether the sum of $C(u)$ and the costs for the link between $C(u)$ and $C(v_i)$ is smaller than the current value of $C(v_i)$. If so, these new costs are assigned to $C(v_i)$ and the node is repositioned in the list L. As we will show later both, the extraction of the current minimum from L, as well as a sorted inserting operation into L can be performed in constant time in this special case.

The pseudo-code for this algorithm looks like this:

Definitions:

s	Start point
L	List of non-permanent nodes
$c(u, v)$	Local costs for link $u \rightarrow v$
$C(u)$	Total costs from s to u

Algorithm:

```
1   C(s) = 0 and C(u) = ∞ for u ≠ s
2   L = {s}
3   while L not empty do
4       u = min(L)
5       remove u from L
6       for each v ∈ {neighbors of u} do
7           if C(u) + c(u, v) < C(v) then
8               if v ∈ L then remove v from L
9               C(v) = C(u) + c(u, v)
10              insert v into L
11          end
12      end
13  end
```

After termination $C(u)$ is initialized for all pixels. If $c(u, v) > 0$ for all voxels the actual path from s to u can then be extracted as follows: Starting from u we start towards the neighbor with the smallest costs. From that we continue like this until we reach s.

Notice that for positive local cost functions once a node has been processed and removed from L in line 5 it will never be re-inserted, i.e. it is permanent then. Therefore, as an optimization it is possible to keep track of that information and restrict the loop in line 6 to non-permanent pixels.

The algorithm is then optimal in the sense that it evaluates each edge only once. Normally the inserting of an element into a sorted list (line 10) would require a complexity of at least $O(log\, n)$. Fortunately in our case we can reduce that complexity to constant time $O(0)$ when the local cost function is integer valued, strictly positive and limited by an upper bound.

In this case we can use a *bucket list* for sorting [27]. Each bucket contains all nodes with the same global costs. The buckets can be directly accessed with an index corresponding to the global costs. In the first step obviously only one bucket is needed (for $C = 0$). In each following iteration for all pixels $C(u)$ can never be smaller than the minimal costs of the previous iteration and never exceed these minimal costs plus the maximal value of $c(u, v)$. In our implementation we define our cost function so that its values are in the range $1 \ldots 255$ so that at any time at most 256 buckets with successive C values are needed. To avoid moving the buckets in memory they are organized in a cyclic array and the index of the current first bucket together with its associated C value is stored. If the lowest bucket is empty we only need to increase the index to the next lowest bucket.

The Dijkstra algorithm is fast so that for typically sized images (e.g. 512^2) even the initial full graph search can be performed in fractions of a second. Some ways for further acceleration are proposed e.g. in [122, 139].

Cost Function

The cost functions used for traditional contour finding in 2D images typically take into account the magnitude of the image's gradient,

$$G = \sqrt{I_x^2 + I_y^2},$$

where I_x and I_y are the partial derivatives of the image intensity I in x and y direction. In [93] an additional binary term is used based on zero crossings of the laplacian operator. Though suitable for detecting the contour of an object such cost functions are not useful for our case, where we target to extract the centerline rather than the contour of a structure.

A first approach is to use a linear function of the intensity:

$$c(u, v) = (\alpha \cdot (I(u) + I(v)) + \beta) \cdot d(u, v) \tag{5.1}$$

$d(u, v)$ corresponds to the physical distance of the two pixels, i.e.

$$d(u, v) = \begin{cases} 1/\sqrt{2} & ; \quad u, v \text{ are horizontal/vertical neighbors} \\ 1 & ; \quad u, v \text{ are diagonal neighbors} \end{cases}$$

In the images we consider here, the structures are brighter than the background. Therefore, we must choose $\alpha < 0$. The parameter β has to be chosen so that $c(u, v) \geq 0$. With β we can furthermore control the "stiffness" of the path. The larger β is the more impact has the constant part of the costs yielding shorter paths, while for low constant costs, longer "detours" can be "afforded" in order to adapt to strong lines in the neighborhood.

We have found a convenient way to determine good values for α and β by deriving them from the user defined gray value window $g_{\text{min}} \ldots g_{\text{max}}$ used to display the MIP. For display, all values smaller than g_{min} are mapped to black, while values greater than g_{max} become white.

Especially for confocal microscopy images the maximal intensity will vary for different parts of the recording. Therefore, it is often necessary to lower g_{max} to increase contrast and brightness. Also it is often useful to crop away some lower intensity noise by choosing $g_{\text{min}} > 0$.

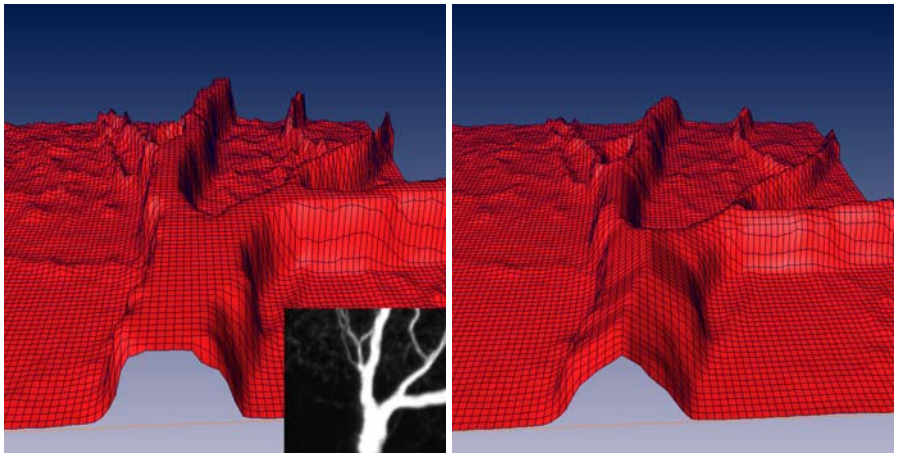

Figure 5.1: Left: MIP of confocal recording of a honey bee's PE1 neuron along with height-field rendering of the intelligent scissors cost function. Right: Improved cost function taking into account centeredness.

The same mapping can then be used for the cost function:

$$\begin{aligned} \beta &= 255 \\ \alpha &= \frac{-255}{2(g_{\max} - g_{\min})} \end{aligned}$$

Note that the resulting cost function has to be clipped to 0 to avoid negative values. Due to the use of the bucket list based sorting described in the previous section attention has to be paid to the maximal possible value. A clipping for large positive values can make sense also. This will also be discussed in the next section.

Centeredness

For tubular objects with diameter in the range of the optical resolution of the image acquisition device, the method described so far produces very good results. For larger structures however there is a problem which can be understood by looking at figure 5.1, that shows the MIP of a sub-region of a neuron recording together with a height-field visualization of the cost function.

While for thin structures the cost function has a clear ridge, the larger dendrites form a plateau with constant costs. Obviously, some clipping has occurred since there are numerous pixels with maximal value. Note we have not artificially introduced the clipping by the visualization or computation of the cost function but it is due to the settings of the microscope while recording. Notice that even when reducing the photo multipliers sensitivity (which would make the smaller and darker structures in other parts of the recording invisible) so that no clipping occurs, the qualitative behavior would remain the same: There is no reason why points in the center of a large dendrite should be brighter than those near the border, if the dendrite is filled homogeneously with fluorescent dye.

The formation of plateaus in the cost function leads to undesired results. In the case of constant local costs, the algorithm will always choose the shortest path connecting the anchor points while we

would like it to use the most centered one. To overcome this we have to incorporate a centeredness term into the cost function.

In order to do that we use a boundary seeded distance map, which encodes for every pixel u the distance $D(u)$ to the nearest boundary point. The obvious problem is that we do not have a segmentation of the object yet, which would be needed for a definition of the boundary.

Fortunately we need the distance map only for the larger structures which in general are much brighter and of higher contrast. We have found that after some basic initial preprocessing a hysteresis threshold segmentation lead to good results for most data sets.

Notice that it is not necessary and not possible to find a threshold that results in a fully accurate segmentation. If that were the case for a specific data set we would rather do a threshold segmentation followed by a skeletonization as described in the previous section, thereby obtaining a completely automated model extraction. Instead, it is sufficient for the segmentation to label the larger parts and it does not perturb the results if some parts of the background are wrongly selected.

With the distance function $D(u)$ the new cost function looks like this

$$c(u,v) = (\alpha \cdot (I(u) + I(v)) + \beta + \gamma \cdot (D(u) + D(v))) \cdot d(u,v) \tag{5.2}$$

Suitable parameters for γ have been found to lie in the range $9...15$ for typical datasets, where D is measured in voxel size.

Instead of computing a two dimensional distance map in the MIP one could also compute a three dimensional distance map on the original image, that could be precombined with the image with appropriate scaling. Qualitatively the result is the same and we have not found particular advantages of one approach over the other in the final results.

Using the cost function (5.2) we achieve curves that are in close correspondence with the actual medial axis, thus yielding the desired results.

5.1.2 Recovering Depth

The previous steps were performed in a two dimensional projection. In order to extract a full three dimensional model we need to recover the third dimension. To do that we again use a cheapest path algorithm in the third dimension. Let $I^{3D}(u,v,w)$ be the original three dimensional image volume. Let $(u_i, v_i), i = 1...n$ be the vertices of the polyline chain in the MIP image. Then a new 2D image is defined by

$$I^{2D}(i,w) = I^{3D}(u_i, v_i, w) \tag{5.3}$$

In this image we have to find a path connecting the left and right border

$$I^{2D}(1,\cdot) \rightarrow I^{2D}(n,\cdot)$$

In cases where the path to be extracted is not the first line drawn by the user but elongates a previously started path or defines a new side branch starting at a branching point, then the depth value is given for the first pixel already:

$$I^{2D}(1,w_0) \rightarrow I^{2D}(n,\cdot)$$

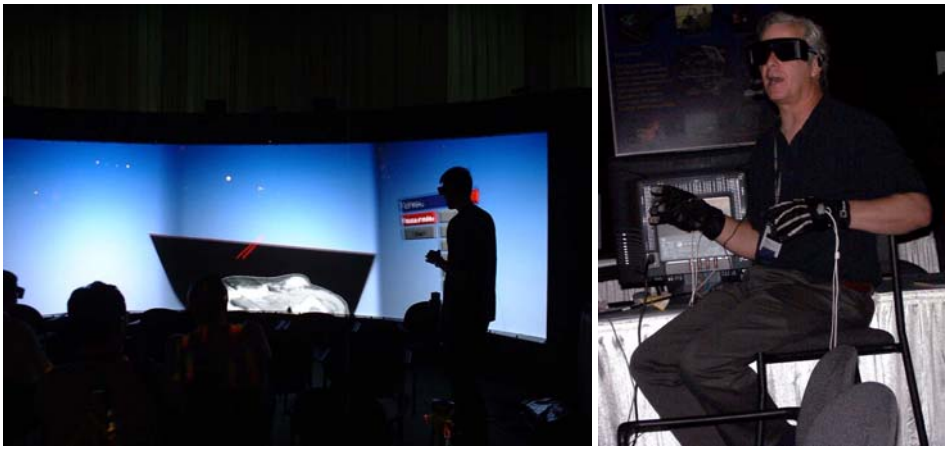

Figure 5.2: Interacting in front of a stereoscopic 120-degree curved screen with tracked pinch gloves. Handheld interaction can still be fatigueing for untrained users. It is most often used for navigation rather than editing. (Left image: The author's silhouette, right image: Paul Mlyniec from Digital ArtForms, both pictures taken in Emerging Technology Forum SIGGRAPH 2002, San Antonio.)

In this case the same algorithm as described in section 5.1.1 can be used, otherwise all pixels $(1, \cdot)$ are used as seed points simultaneously and the iteration is done until the first pixel $(1, n)$ becomes permanent.

5.1.3 Results and Discussion

We have developed a method for the interactive tracing of dendritic trees in neurons. With this method it is possible to accurately extract the centerline graph in even complex neurons. Also in cases where the image quality is poor, with a few more mouse-clicks good results can be achieved.

An obvious question is, whether it would not be possible to work directly in three dimensions and search the optimal paths there. Beside performance issues, two main factors complicate this: Visualization and interaction. We have already discussed the difficulty of visualizing an entire data volume at once. The problem can also be seen in Figure 4.2, where a "crisp" visualization of the structure in the image data set contradicts good visibility of the selected voxels inside this structure.

The interaction would require an input device with at least three degrees of freedom. Such devices exist, but in the majority of applications they are primarily used for navigation rather than editing. It has turned out, that using a hand-held 3-DOF or 6-DOF input device is quite fatiguing for the arm and therefore not well suited for editing over several hours, compare Figure 5.2. Also it was important for us to develop a tool that can be used by numerous biologists, which do not have access to stereoscopic display systems and tracking hardware. Virtual Reality is still a quite new technique and we expect to see significant improvements in the usability in the next years, which may change the situation.

In the next section we will show, how the results from this section can be used for automatic tracing algorithms in three dimensions.

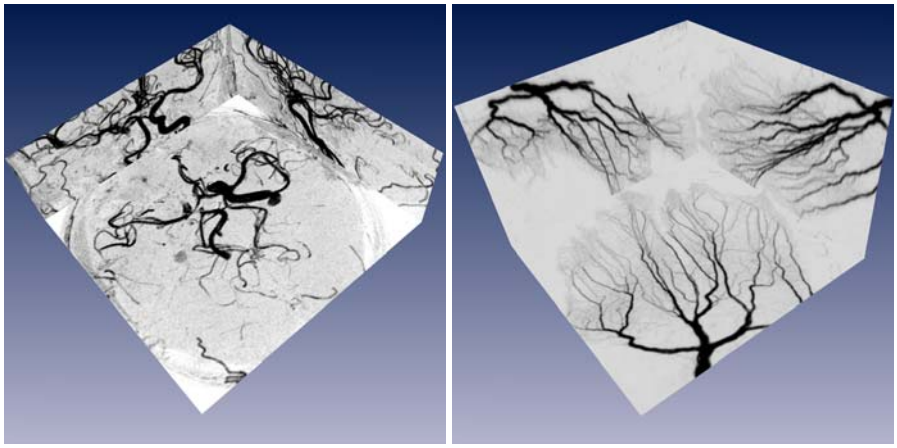

Figure 5.3: Images from angiography and recordings of single neuron fills look similar. Left image shows a human cerebral digital subtraction angiography. Right image shows PE1 neuron in the honey bee brain. In detail, however, there are differences in contrast, signal-to-noise ratio and the structures the researchers are interested in.

5.2 Automatic Extraction

In the following we will discuss methods for automatic extraction of line-like structures from image data. The vast majority of previous work related to this field, however, has not focused on the extraction of dendrites from microscopic images, but on the extraction of vessel trees. The data sources for three dimensional images of vascular network are mainly Computer Tomography Angiography (CTA) and Magnetic Resonance Angiography (MRA).

Angiography images are structural similar to the recordings of neurons as can be seen in Figure 5.3. But there are also important differences:

- The signal-to-noise ratio is often much better in the angiograms than in confocal recordings, especially when contrast agents are used.
- For medical diagnosis and treatment planning especially the larger vessels are of interest, e.g. when investigating stenosis or aneurysms, while the neuro biologists are interested in complete models of the dendritic tree containing even the finest branches.
- Although a non-isotropic voxel size is common in medical images, there is not such a pronounced anisotropy in recording resolution (unsharpness in z-direction) like in the confocal recordings. Therefore, carefulness is advised when generalizing the results from vessel tree extraction.
- Most automatic vessel detection methods use some functional, which for each point in the image domain gives a likelihood for this point to be part of a vessel or to lie on the centerline of a vessel.

5.2.1 Scale Space

We have distinguished voluminous and line-like structures. However, the decision whether a cylindrical object is line-like or not depends on the scale of observation, i.e. the resolution at which we are looking. This is a very general property of many computer vision problems. Note that the term 'scale' is different from (though related to) the sampling frequency of the input image, i.e. the voxel resolution.

If we use a an image operator based on partial derivatives, like the gradient magnitude or the Laplacian, and approximate it by differences of the nearest neighbors of a voxel, as it is standard in image processing, we would implicitly choose a particular scale which is determined by the voxel size. Since structures with different diameters are to be detected, this is not a good idea. We have to analyze the image on multiple scales. Good overviews on scale spaces and multi scale image analysis are given by Lindeberg in [72, 74, 73]. The scale space representation L of an n-dimensional function $f : \Re^n \rightarrow \Re$ is a family of functions with a family parameter t called the scale parameter t, which can be derived from f by convolution with a Gauss kernel.

$$\begin{aligned} f : \Re^n \times \Re^+ &\rightarrow \Re \\ L(\cdot, t) &= g(\cdot, t) \circ f(\cdot) \\ g(\cdot, t) &= \frac{1}{(2\pi t)^{\frac{N}{2}}} e^{\frac{-\sum x_i^2}{2\sigma^2}} \\ \sigma &= \sqrt{t} \end{aligned} \tag{5.4}$$

It is well-known that Equation (5.4) can also be derived as the solution of the linear diffusion equation:

$$\begin{aligned} \partial_t L &= \frac{1}{2}\Delta L = \frac{1}{2}\sum_{i=1}^{n} \partial_{x_i x_i} L \\ L(\cdot, 0) &= f(\cdot) \end{aligned}$$

An important property of a scale space representation is the decrease of "amplitude", i.e. the magnitude at local maxima and minima will in general decrease when a function is subject to scale space smoothing according to Equation (5.4). Due to the commutativity of convolution and derivation the same is true for derivatives of L. As an example consider a one-dimensional sinusoid:

$$\begin{aligned} f(x) &= \sin(\omega_0 x) \\ L(x, t) &= e^{-\omega_0^2 t/2} \sin(\omega_0 x). \end{aligned}$$

From this it can be derived immediately that the amplitude of the function and its m-th order derivative decrease exponentially with scale:

$$\begin{aligned} L_{\max}(t) &= e^{-\omega_0^2 t/2} \\ L_{x^m,\max}(t) &= \omega_0^m e^{-\omega_0^2 t/2} \end{aligned}$$

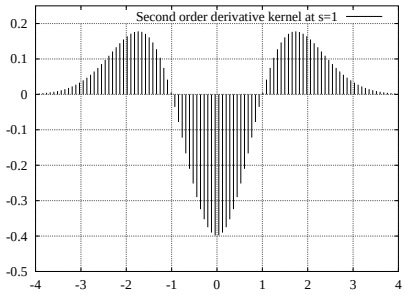

Figure 5.4: Second order derivation in scale space is identical to convolution with second order derivation of Gaussian (example for scale 1). If applied to a bright structure of radius 1 on dark background the response will be a high magnitude negative value, since near pixels and farther pixels are weighted with opposite sign.

structure	λ_1	λ_2	λ_3
bright blob	$\ll 0$	$\ll 0$	$\ll 0$
dark blob	$\gg 0$	$\gg 0$	$\gg 0$
bright line	≈ 0	$\ll 0$	$\ll 0$
dark line	≈ 0	$\gg 0$	$\gg 0$
bright plate	≈ 0	≈ 0	$\ll 0$
dark plate	≈ 0	≈ 0	$\gg 0$

Table 5.1: Eigenvalue patterns of Hessian matrix for blob-like, line-like and plate-like structures. Eigenvalues are $|\lambda_1| < |\lambda_2| < |\lambda_3|$.

Lindeberg shows that in order to be able to detect image features on different-scales and automatically detect the scale at which a feature is strongest, it is necessary that the image operators use 'normalized derivatives':

$$\partial_{x,\gamma-norm} = t^{\gamma}\partial_x$$

Note that for $\gamma = 1$ this corresponds to the dimensionless variable

$$\xi = \frac{x}{\sigma} \quad \Rightarrow \frac{\partial}{\partial\xi} = \sigma\frac{\partial}{\partial x}$$

Using ξ and ∂_{ξ^n} makes an operator invariant against image scaling, i.e. the change of physical units.

5.2.2 Line Detection Operators

As we have seen earlier, the magnitude of the image intensity gradient is an indicator for the frontier between two objects of different intensity. A pixel in the center of a vessel on the other hand corresponds rather to a maximum of intensity than to a jump, leading to vanishing intensity gradient.

Therefore, a second order derivative operator like the Laplacian seems to be more appropriate. And indeed, if applied at the appropriate scale, it could be used as an indicator for center line

points, since it responds to local maxima as can be understood from Figure 5.4. However, it does not take into account the constant intensity tangential to the vessel and would therefore even better respond to point-like spots.

For a pixel on (the center of) a line we would expect the intensity to drop significantly in all directions normal to the line and to not change in the direction tangential to the line. In order to encode this more complex directional structure of the intensity derivatives, the Hessian matrix can be employed:

$$H_ij(I(\vec{x}) = \partial_{x_i}\partial_{x_j} I(\vec{x})$$

The three eigenvector/eigenvalue pairs encode the principle directions of second order derivatives and the corresponding second order derivative value. Several researchers have used line detection criteria based on these eigenvalues and/or eigenvectors [79, 32, 109, 63, 5]. In [61] it is proposed to use the Hessian eigenvectors to detect the line direction and then apply a non-isotropic line detection filter. In [28] a multi-scale wavelet based edge-detection followed by a reconnection step is proposed. [5] reviews and benchmarks several of these methods.

There is no canonical way to create a scalar valued criterion from the three eigenvalues and eigenvectors, instead the measures proposed in literature are based on geometric interpretations, study of analytical models, or empirical studies.

The qualitative behavior can be seen in Table 5.1, compare [32]; they use a combination of three different expressions: Let $\lambda_1, \lambda_2, \lambda_3$ be the three eigenvalues sorted in order of increasing magnitude. Then let

$$R_A = \frac{|\lambda_2|}{|\lambda_3|}, \qquad R_B = \frac{|\lambda_1|}{\sqrt{|\lambda_2\lambda_3|}}, \qquad S = \sqrt{\sum_i \lambda_i^2}$$

R_B measures deviation from blobs: it is large for blobs and small for lines and plates. R_A distinguishes lines from plates and becomes large for line-like structures. Finally, S, which is the Frobenius matrix norm measures "structuredness", i.e. it is supposed to distinguish background noise from structures, where it becomes large. These three values are then combined to the "vesselness" measure at the current scale t:

$$V_t = \begin{cases} 0 & ; \quad \lambda_2 > 0 \vee \lambda_3 > 0 \\ \left(1 - exp(-R_A^2/(2\alpha^2))\right) exp(-R_B^2/(2\beta^2)) \left(1 - exp(-S^2/(2c^2))\right) & \end{cases}$$

α, β, and c are empirical parameters and in [32] the authors suggest $\alpha = \beta = 0.5$ and to set c to half the maximal occuring value of the S.

Krissian *et al.* study analytical models to derive properties of the eigenvalues: they investigate a cylindrical circular model, a toroidal circular model and an elliptical cylindrical model, all with a Gaussian intensity cross section. They derive properties for the λ_i which are similar to the above mentioned as well as an estimation of the magnitude of λ_3.

However, they suggest to not use the eigenvalues for vessel detection directly because the discretization artifacts occuring with one-pixel sized vessels and and the uncertainty introduced by non-circular cross sections make such criteria unreliable. Instead, they use the condition $\lambda_2 > 0 \vee \lambda_3 > 0$ as a pre-selection criterion and measure the intensity drop in normal direction on a circle around the tangential direction (as defined by the eigenvector $\vec{e_1}$):

$$V_t(\vec{x}) = \frac{1}{2\pi} \int_{\alpha=0}^{2\pi} -\nabla_t I(\vec{x} + \sqrt{3}\sqrt{t}\vec{v}_\alpha) \cdot \vec{v}_\alpha$$

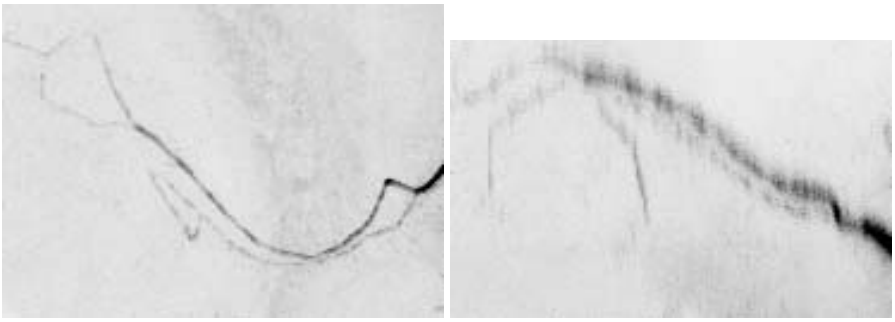

Figure 5.5: Maximum intensity projection of a small part of a confocal neuron recording. Left image is a projection in recording direction, right image is an orthogonal view. Notice the lower resolution in z-direction (right) and the gaps and blobs due to staining inhomogeneities (image inverted for better printing).

where $\vec{v}_\alpha$ lies on a circle in the plane spanned by the two eigenvectors corresponding to the two strongest eigenvalues: $\vec{v}_\alpha = \cos(\alpha)\vec{v}_1 + \sin(\alpha)\vec{v}_2$. The radius parameter $\sqrt{3}$ is derived from the analytical model.

Another model based method is proposed in [3]. They directly trace line-like structures by shifting a set of 4 $N \times N$ kernels along the line to be extracted. At each point the optimal scale (diameter) and orientation (centerline direction) is detected by probing in multiple directions and detecting the maximal response function. In effect this is quite similar to the response function in [63].

5.2.3 Neuron Data Sets

We have tested the above mentioned methods for the neuronal data sets. Especially the method [63] enhanced quite well the dendrites while suppressing noise. However, none of these methods was able to robustly detect the smallest structures in the data sets and create a connected graph that was consistent with the image data. Figure 5.5 illustrates why. (Note that this example has still quite good quality compared to the worst parts of typical confocal recordings). We see that in the smaller parts of the dendrites the dye distribution is relatively inhomogeneous. Combined with the lower resolution in z-direction, this leads to little blobs with major axis in z-direction separated by intensity gaps. It is clear that the eigenvectors of the Hessian matrix will not yield a reliable criterion.

5.2.4 Tree Re-Assembling

Probably any local criterion will fail in fully and correctly identifying all the parts of the dendritic tree that a human is able to detect. The reason is that also global aspects have to be considered, like human observers do. Although the multi scale approach extends the locality criterion to a certain degree it is still not able to take into account global structural information. A second problem is the inhomogeneous staining in the thin parts combined with the lower resolution in z-direction. This leads locally to structures with major direction in z-direction rather than in direction of the dendrite as seen in 5.5.

Therefore, we have concentrated on developing a method which does rely primarily on local information. We have found that using even simple filtering combined with a subsequent thresholding,

it is easily possible to extract a sub-set of the true object.

In a next step, a more relaxed criterium is used to identify candidate points, which are potentially part of the object. The decision of whether such candidates are considered true object points or background noise depends on the "costs" to connect them to the object, in the sense of Section 5.1.1, with an appropriate cost function. The method can be built on top of any other line detection operator. The steps of the algorithm are in detail:

1. Identify a set C of core voxels that are part of the object.
   Note: C will in general not be a single connected component.
2. Identify a set P of voxels that are most likely part of the object.
3. Identify and label all connected components C_i in $P \cup C$.
4. If still more than two components C_i exist, identify the two *closest* components C_i and C_j. If the costs for the *cheapest path* from C_i to C_j does not exceed a given threshold, add this path to C. Continue with step 3.

The definition of the *shortest path* in step 3 is completely analogous to the definition in Section 5.1.1. We define the distance of two components to be the costs of the cheapest path connecting them.

The result is a single connected object. This can be further processed using a skeletonization algorithm.

5.2.5 Skeletonization

Most of the described methods for automated line-structure extraction will not generate a one-voxel thin tree. Therefore, we use a thinning algorithm to reduce the structures to their centerlines. The centerline is closely related to the skeleton. Blum introduced a definition for the skeleton based on the grass-fire analogy [8]: If a fire was lit at all boundary voxels and the fire propagated with constant speed towards the center of the object, the skeleton consists of the points where fire fronts meet. Other definitions base on local discontinuities of the boundary seeded distance map. For a general object the skeleton consists not only of line but also of surfaces.

Numerous methods exists to compute or approximate skeletons. The methods include simulation of the grass-fire, boundary shrinking [146], analytic computations, extraction from distance maps [108, 103], and topological thinning [62, 65, 104]. We do not want to give a complete survey of skeletonization and thinning here, but merely concentrate on two methods that we have found particular usefull for our application. For further reading we refer to the given references.

Sato *et.al.* [108], propose a method which builds a tree by succesively connecting the farthest point in a point seeded distance field to the tree generated so far. The algorithm then marks the parts already visited. The method guarantees, that the result is a graph without loops. If the initial object, however, is not tree like, the results are not always satisfying.

Topological thinning algorithms guarantee that the result is homotopic to the input object. They successively remove points until no further points can be removed without changing the topology.

To guarantee centeredness of the resulting skeleton, Pudney has proposed to test and remove voxels in order of their distance value [104].

Both methods have produced similar results in our application and worked quite well. Notice that many of the input structures are not much thicker than one voxel anyway.

5.2.6 Radius Estimation

For morphometric analysis, as well as for display purpose, one is not only interested in the centerline tree, but also in thickness information. One way to compute the thickness is based on the distance map. The local thickness that should be associated with a point on the centerline is the value of the boundary seeded distance field at that point. If a distance ordered thinning algorithm is used, this does not require an additional computational step, but it is generated automatically. Note that the distance map computation requires a segmentation of the object. If line extraction operators like those discussed in Section 5.2 are used, such a segmentation is not necessarily generated. Instead, the operator directly enhances the centerline. K. Krissian *et. al.* propose in [63] to derive the diameter from the scale in which the response to the line detection operator is strongest. We have found however, that in most cases the structures that cannot be threshold segmented, are the thinnest lines, which always have their strongest response on the smallest scale.

5.2.7 Results and Discussion

We have developed a method to automatically extract graphs of neurons from image data. Based on existing line structure indicators, we have employed a cheapest-path algorithm to connect initially disconnected components.

As we have shown, various different methods exists, which deal with the extraction of line-like structures. One important observation we have made for the data sets relevant in our work is this: The thicker structures normally have a good contrast and can be extracted quite easily. The difficult parts are the very thin lines. Here the fluorescent dye concentration is often inhomogeneous and the small volume leads to a small signal-to-noise ratio.

Therefore, we have found that applying a combination of methods for the different parts of a data set can be required. Also we have found that the optimal combination of methods depends on the particular data set. The typical steps are

1. Correction of an intensity drop in z-direction.
2. Interactive removal of undesired objects and noisy parts in the image.
3. Hysteresis thresholding to extract bright (and thick) parts of the image.
4. Filtering with a line enhancement operator based on the eigenvalues and vectors of the Hessian on a small scale to extract small lines.
   If the resolution in z-direction is significantly lower than in axial direction however, we have found that skipping this step can lead to better results.
5. Selection of candidate points followed by the re-assembling step described in Section 5.2.4.

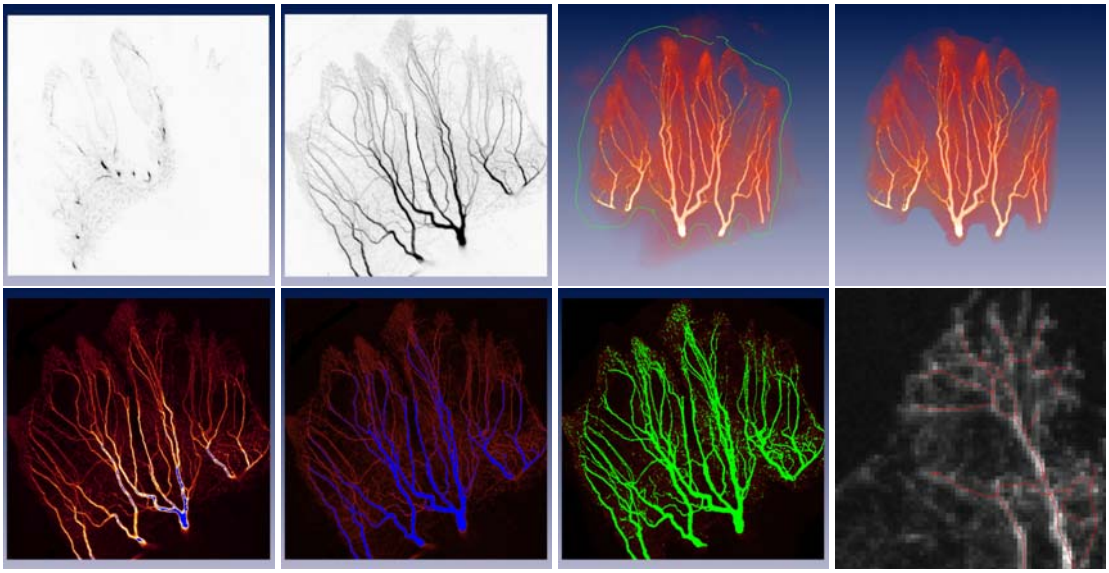

Figure 5.6: top(i-iv), bottom (v-viii), Work-flow for automatic extraction of a graph model of the dentritic tree: A confocal recording of a honey bee's PE1 neuron is shown with one slice (i) and a maximum intensity projection (ii). Automatic compensation of the intensity drop in z-direction has already been done. Interactive volume editing tools are used to remove some undesired objects in the data set (iii,iv). At this stage a line enhancement filter could be applied, which has not been done in this image due to unisotropical resolution (see text). By interactively modifying a colormap in the maximum intensity projection (v and vi), a confidence level is found. All blue points will be part of the object. A second level is chosen to define points that can potentially be part of the object, if suitable connections exist (green pixels in vii). After the re-assembling, skeletonization, line-extraction and line-smoothing step, a polyline-based description is obtained. The full result is shown in Figure 5.7.

6. Thinning and distance map computation in the thicker parts for radius information.
7. Conversion of the voxel based skeleton description into an explicit poly-line based description.

These steps are illustrated in a typical example in 5.6. The resulting tree is shown in Figure 5.7.

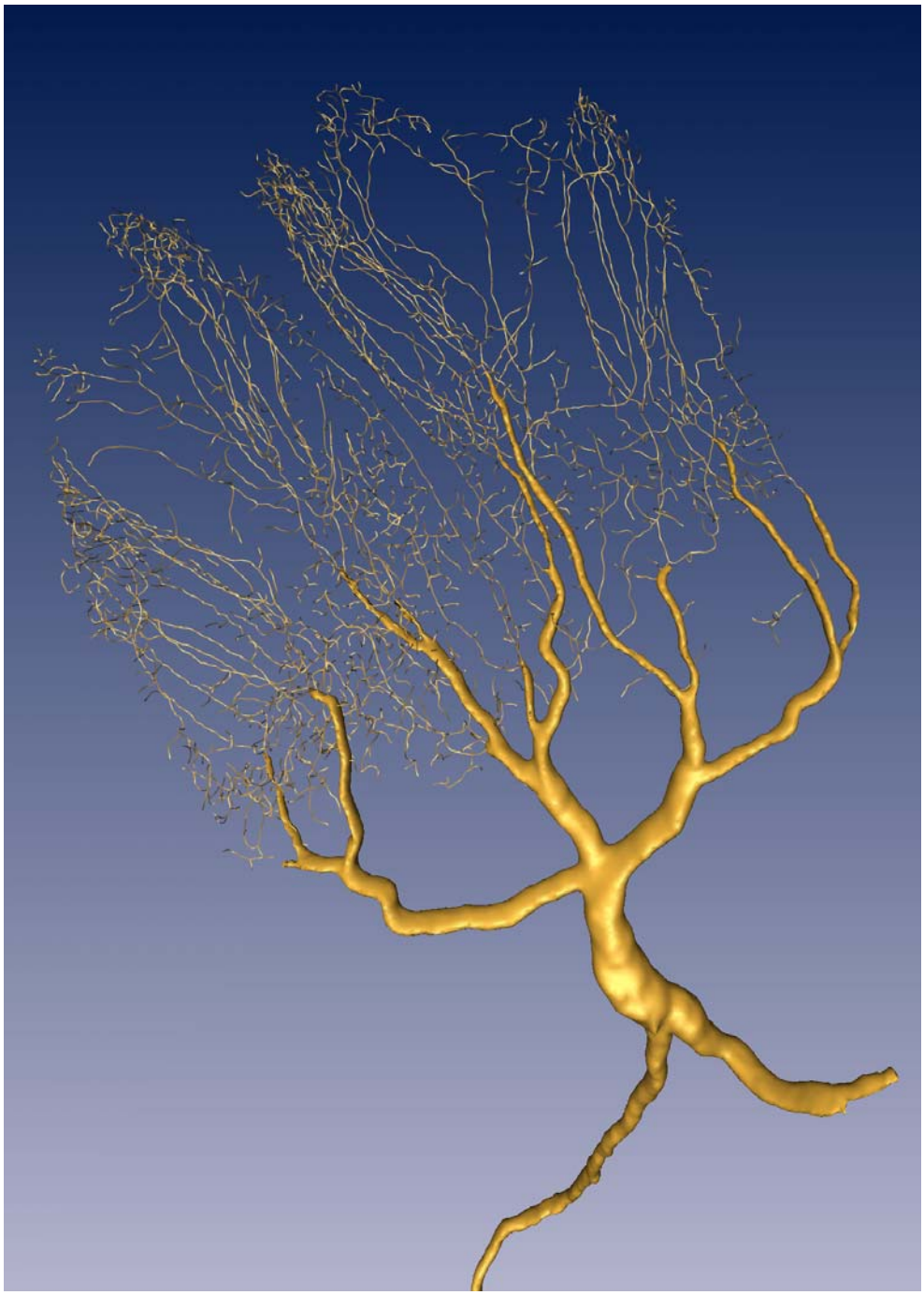

Figure 5.7: Dendritic tree of a honey bee's PE1 neuron. The poly-line tree has been extracted automatically from a high-resolution confocal recording. It is rendered using illuminated lines. The thicker parts are rendered as polygonal models, which have been extracted using hysteresis thresholding from an overview scan.