Chapter 6

Geometry Reconstruction

In this chapter we present algorithms to create explicit polygonal representations of anatomical structures from label images created with the methods discussed in Section 4 and from the graph structures as discussed in Section 5.

Polygonal models are an essential prerequisite for many visualization, analysis, and simulation techniques. In particular they can be used as input for tetrahedral grid generators which produce volumetric grids needed for finite element or other numerical methods. Especially the two latter applications require the polygonal models to be of high quality and consistency. I.e. there should be no gaps, self intersections or wrongly oriented polygons. In the following we will concentrate on the generation of triangular meshes since they are the most commonly used and easiest to treat type of polygonal meshes.

6.1 From Labels to Geometry

The given task is the generation of a (triangular) surface mesh, of a model implicitly encoded in a three dimensional array of scalar values or labels as described in Section 4, called label field. The model should consist of surfaces separating the regions with different labels.

An suitable algorithm has to take into account three main requirements:

Consistency: The surface should be consistent with the labeling, i.e. no two different labels should be on the same side of the separating surface. In addition, the surface should be topologically correct, i.e. it should not contain self intersections or gaps.

Shape: Within the constraints imposed by the labels the mesh should have a nice shape, e.g. often smoothness is required.

Quality: Especially if the mesh is used for subsequent computations, like numerical simulations, the triangular mesh should meet quality criteria like good aspect ratio or an even angle distribution.

Each of these criteria can complicate meeting the others. For instance simple smoothing approaches to assure nice shapes will violate consistency with the labels.

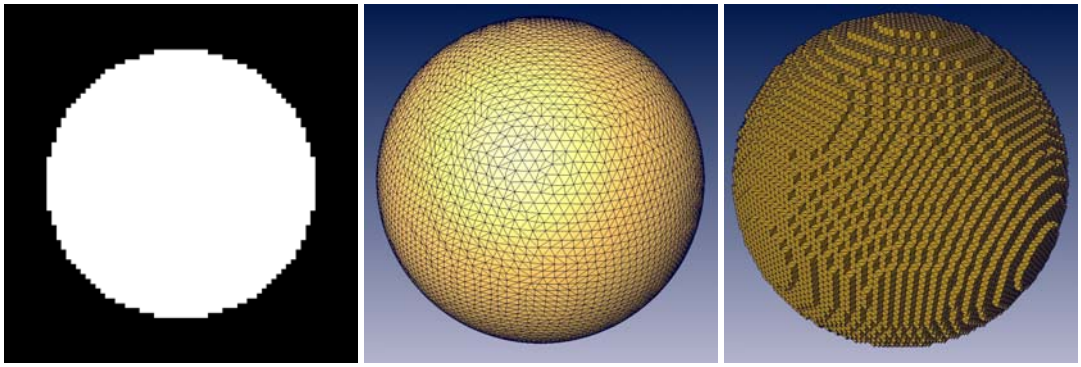


Figure 6.1: Two possible triangulations of a labeling. Although the two surfaces differ less than one voxel edge length at each point, the naive voxel surface solution on the right is not suitable for most applications.

The task of extracting surface meshes from regularly sampled three dimensional scalar fields has been addressed by many researchers. One of the main applications have been iso-surface extraction [78, 91, 88, 96, 106] and related problems, ranging from polygonalization of implicit surfaces used in modeling and animation [36] to computation of complex meshes like removal envelopes [110].

These methods share the assumption of a *binary* partitioning of the domain, e.g. into regions with a value smaller than a given threshold and those with a larger value. The label fields we are dealing with, however, typically contain more than two labels so that these methods are not applicable.

The ad-hoc approach to compute a triangulation for each label individually does not produce satisfying results since it does not generate a consistent triangulation in regions where three or more different labels are adjoining. However, for pure visualization purposes such meshes can be sufficient. This may be one of the reasons that this problem has been addressed much less frequently than the binary case.

A straight forward solution to the problem would be to insert quadrilaterals perpendicular to the main axes between voxels with different labels ("Lego surface", compare Figure 6.1). Though this produces a consistent mesh, this staircase surface is not suited for visualization or analysis (like area measurement).

Approaches that have been taken for generating surface meshes from non-binary labeled medical image data often rely on connecting contours in neighboring slices [86]. In such methods ambiguities can arise and treating changing topology of the contours is not easy. Again, most of these methods treat each compartment surface as a separate object.

Bloomenthal and Ferguson [7] describe a different method for generating surfaces from non-binary space partitionings, they refer to as non-manifold surfaces. Their paper mainly addresses modeling via implicit surfaces and computational solid geometry applications. In contrast to our table-based approach they propose a continuation method. A cubic cell is propagated across the surface and decomposed into tetrahedra. For each tetrahedron a plausible triangulation is constructed algorithmically. Although decomposition into tetrahedra simplifies the polygonalization problem, it produces an excessive number of triangles, many of them not well-shaped. This is also the case with marching tetrahedra algorithm by Nielson and Franke [97].

For these reasons we have developed a method that avoids these problems and generates well

shaped and consistent surface meshes. The surfaces created by our algorithm may contain contours and points where three or more regions join. More than two triangles are attached to an edge which is part of such a multi-region contour.

In the following description we will first concentrate on the consistent triangulation (based on our method [142]) and then discuss how weights that complement the label information can be taken into account to generate smooth surfaces. Finally, we will discuss how proper weights can be generated.

Similar to the iso-surface algorithms mentioned above our algorithm works on a per-cell basis. A cell consists of 8 adjacent grid points. In the binary case like in the marching cubes only $2^8 = 256$ possible configurations can occur for one cell (two possible colors for each of its corners), many of which are topologically equivalent. For the general case with up to eight different colors this number grows drastically. Therefore, the use of look-up tables is not straight forward and the manual creation of them not feasible.

Therefore, we will present a fully automatic method for creating the triangulation. In addition, we will show how look-up tables for the *most common* configurations can be generated automatically and used to achieve a very good performance.

6.2 Triangulation Algorithm

In the following sections we will outline the ideas of the surface meshing algorithm. We will first describe how to determine a unique region type for every point in space. Then a method is presented to compute a consistent approximate triangulation. Finally, we show how the meshing algorithm can be accelerated via look-up tables and compare the method with standard marching cubes algorithms.

6.2.1 Space Partitioning

Our goal is to compute non-manifold surfaces which accurately separate regions of different type in space. For example in a medical application some parts of a volume may be classified as bone, while others are classified as muscle or fat. We denote the total number of region types by n.

In the following we assume the region types to be defined on a discrete uniform grid. In case of surface modeling via generalized implicit surfaces such a labeling can be obtained by evaluating an analytical expression at each grid node. In medical applications usually a segmentation algorithm like thresholding or region growing is applied to a stack of tomographic images, e.g. CT or MR images.

Let us consider a cubic grid cell defined by eight vertices. In the general case it is not quite obvious how to subdivide the grid cell into regions of different types, since the type information is located at the vertices. To constitute a simple and unique space partitioning strategy we define a set of n probabilities p_i at each vertex. If a vertex is assigned to material k then all p_i are set to zero except p_k which is set to 1.

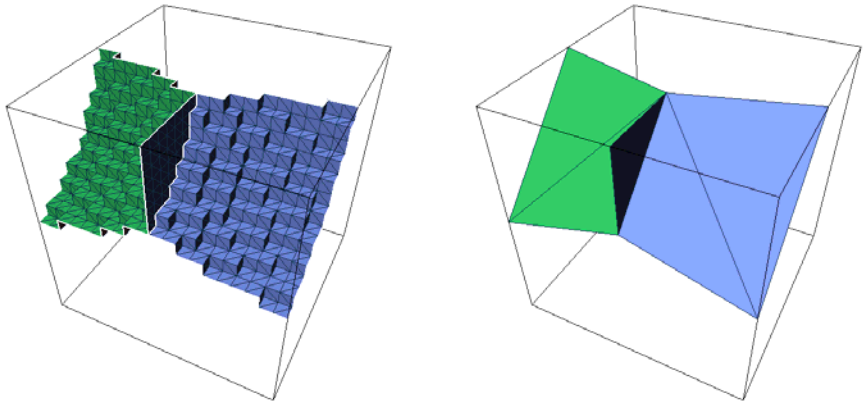


Figure 6.2: The left image shows the surface in a cell before the simplification step. The surface in the right image has been simplified by keeping only branching points and points on edges.

The probabilities for all types are interpolated independently within a cell. To classify an inner point $\boldsymbol{x}$ we simply choose the region of maximal probability, i.e.,

$$\text{region}(\boldsymbol{x}) = \{i \mid p_i(\boldsymbol{x}) \text{ maximal}\}, \tag{6.1}$$

where $p_i(\boldsymbol{x})$ denotes an interpolated probability at $\boldsymbol{x}$. For the triangulation algorithm described in the following section we use trilinear interpolation to compute $p_i(\boldsymbol{x})$. Other interpolation methods can be used as well, but usually they will result in more complex surface topologies.

It should be mentioned that a classification according to maximal probability does not require p_i to be 0 or 1 at the vertices of a cell. However, the case of distinct region labels naturally corresponds to such integer vertex probabilities. In addition, integer probabilities make it easy to distinguish between topologically different configurations. Therefore, cell triangulations can be stored in look-up tables using a simple indexing scheme. The generation and use of non-integer probabilities will be discussed in subsequent sections.

6.2.2 Cell Triangulation

The interpolation model defines a unique set of separating surfaces inside a cell. On these surfaces the two largest probabilities are equal. Since trilinear interpolation is a non-linear transformation the inner surfaces are not planar. Our goal is to construct a triangular approximation of these non-planar surface patches. Such a triangulation has to be consistent between adjacent grid cells to avoid holes in the resulting mesh. It should also preserve the topology inside a cell.

In the case of two different region types 256 configurations are possible which can be divided into 14 different topological classes. For three region types there are 44 additional topological classes, see Figure 6.4. To be able to handle any case with up to eight different region types we would like to have a method which is capable of generating the triangulation for any given configuration automatically. Such a method can be obtained by a straight-forward subdivision approach.

Subdivision. To obtain a representation of the inner structure of a grid cell we start by subdividing the cell into a number of smaller sub-cells. For each sub-cell the probabilities p_i at a

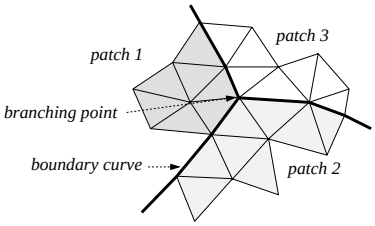
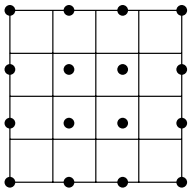


Figure 6.3: Parts of three different patches delimited by boundary curves (dark lines). At a branching point multiple boundary curves join.

representative point $\boldsymbol{x}$ are interpolated. We evaluate the probabilities not at the centers of a sub-cell, but at slightly translated locations as depicted in the following diagram:



Points are distributed equi-distantly. Notice, that they are located on the cell's faces for sub-cells at the boundaries. This way we can guarantee consistency between adjacent grid cells.

The sub-cells are classified according to Equation (6.1). Whenever two adjacent sub-cells are of a different class their common face is added to an intermediate triangulation. An example of a resulting sub-cell triangulation is shown in Figure 6.2. It turns out that in case of integer vertex probabilities 6^3 sub-cells are sufficient to give a topologically correct representation of the implicitly defined separating surfaces.

From the high resolution sub-cell triangulation we compute the final cell triangulation using a mesh simplification step.

Finding Patches. Simplification of the sub-cell triangulation first requires analysis of the surface topology. All connected triangles separating the same vertex classes are grouped into patches. Then the boundary curves surrounding each patch are extracted. For each vertex of the triangulation the number of boundary curves it belongs to is determined. For inner points of a patch this number is zero. Vertices which belong to more than two boundary curves will be referred to as branching points. The notion of surface patches, boundary curves, and branching points is illustrated in Figure 6.3.

Simplifying Boundary Curves. The key observation for the following simplification step is that only few vertices of the current triangulation are of real importance. Among these vertices are the branching points because they reflect the inner topology. Also vertices lying on the cell's edges are important, since these are referenced by multiple cells. All other points will merely introduce complexity on sub-cell level, which is not necessary to create a topologically correct triangulation. Therefore, one can thin out the boundary curves by keeping important points only. After this procedure the boundary curve of a patch often consists of just three or four vertices. However, there are also patches with five or more vertices.

Re-tiling Patches. After simplifying the boundary curves the patches are re-tiled with triangles again. If a patch is planar this can be done in a straight-forward way, for example using an anchor point strategy.

More attention has to be paid in cases where non-planar patches occur. Then the simplification of the bounding curve in conjunction with an unfavorable triangulation can lead to patches which penetrate each other. We avoided all penetration problems by inserting an additional center vertex into patches with 5 or more boundary points and choosing a fan-type triangulation scheme. However, in many cases it is also possible to find a suitable triangulation without introducing an additional vertex.

6.2.3 Look-Up Tables

Subdividing grid cells, computing an intermediate sub-cell triangulation, and simplifying the resulting surfaces is a relatively time-consuming procedure and is definitely not suited to handle ten-thousands of grid cells occuring in common data sets. Fortunately the overall algorithm can be accelerated very much by making use of look-up tables. In contrast to standard marching cubes algorithms in our case not only points on edges have to be created, but also inner points and points on faces. The look-up table has to contain the point types in addition to the triangle information for each cell configuration.

As already mentioned in the case of two vertex classes only 256 different configurations are possible. For three vertex types this number increases to 6561 (3^8), although there are only 58 different topological situations (including the 14 standard cases). These 58 configurations are shown in Figure 6.4.

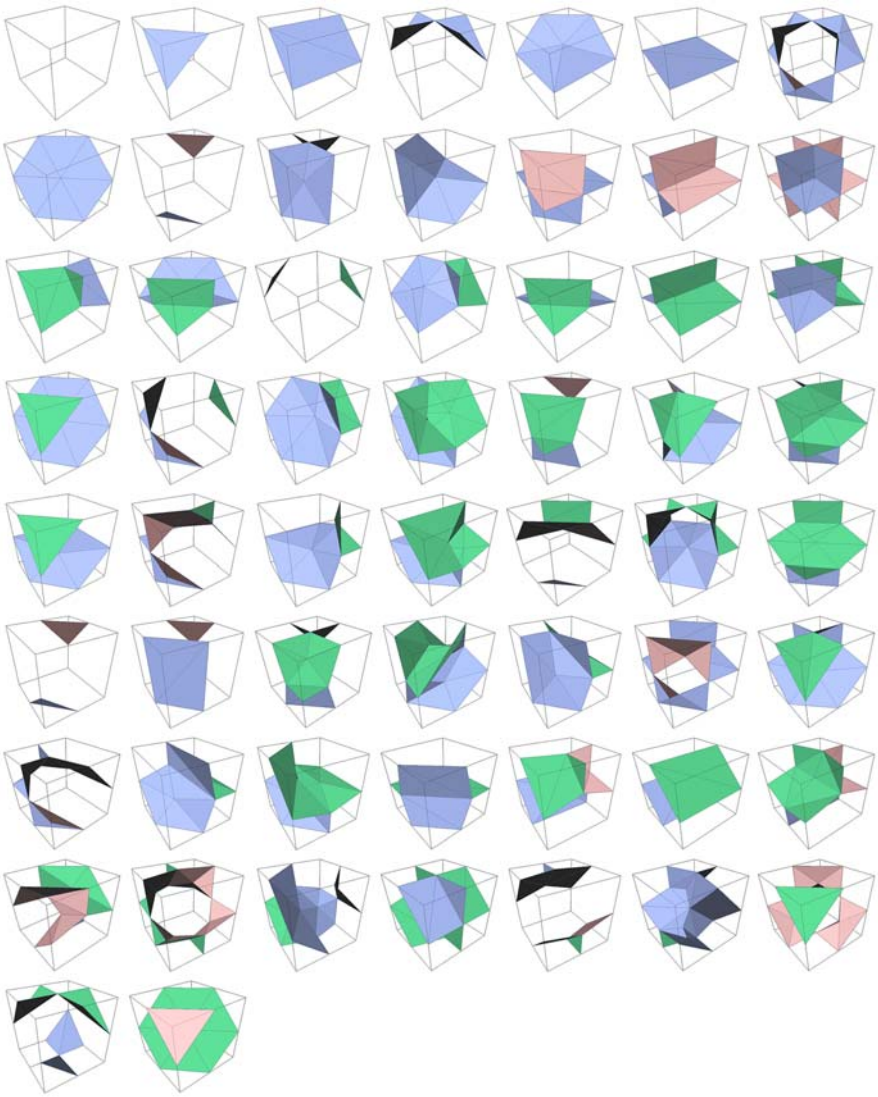


Figure 6.4: The image shows the look-up table for up to three different vertex classes. The first two rows contain the cases with only two classes. For some cases the triangulation is different from the original marching cubes [78].

If the full triangulation for all 6561 configurations is tabulated, about 70 000 triangles have to be stored. The exact number depends on how many inner points are inserted during the patch re-tiling step. For every triangle three one-byte point indices have to be stored. For every vertex an additional byte is necessary to encode the point type (e.g. vertex, face vertex, or inner vertex). Consequently, the table can easily be kept smaller than 1 MB. While this is still a handable size for a look-up table, a full table gets far too large in case of four vertex classes or more. For four classes 65536 (4^8) configurations are possible, but only 124 of them are topologically distinct (including the 58 cases from Figure 6.4). There are two ways out of this dilemma. The first way is to store only the topological different cases for the 'more than 3' configurations and to perform a mapping to a base configuration before the table look-up. Then the vertices read from the table have to be rotated and mirrored accordingly. Of course this method could also be applied to the 3-type case. However, here the use of a full table is feasible and easier to implement.

The second alternative to handle configurations involving four and more vertex classes arises from the observation that in most applications these cases are very rare. This makes it feasible to compute the triangulations for these cases on-the-fly using the comparatively expensive subdivision technique described above.

6.2.4 Comparison to Marching Cubes

Our method resembles the standard marching cubes algorithm [78] and its variations in respect of the cell-by-cell traversal and the use of look-up tables to create triangular surface patches. However, there are also some important differences.

First of all, checkerboard cases are handled differently than in most isosurface algorithms. Such cases occur if adjacent vertices on a face are of different type, while the vertices located at opposite corners have the same type. In this case the original marching cubes algorithm chooses a triangulation which does not preserve the symmetry of the configuration index. Special care has to be taken to ensure consistency between neighboring cells. Otherwise the resulting surface would exhibit holes [89, 77, 96, 106]. The classification model described in Section 6.2.1 avoids such problems, prescribing a symmetric triangulation, i.e., a triangulation which has a branching point at the center of a face. Such cell triangulations are depicted in Figure 6.4. The approach avoids any ambiguities at the expense of an increased topological complexity of the resulting surface.

Another difference to the original marching cubes algorithm is that we have integer probabilities at the vertices instead of fractional values. As a consequence, points on edges are always located at the edge midpoint. In case of a binary vertex classification a similar approach is known as discretized marching cubes [88]. In contrast to the general case e.g. bisection has the effect that only a limited set of triangle orientations occur. This makes it possible to combine multiple triangles with equal orientation. Triangle decimation was the reason why the discretized marching cubes algorithm has been developed. A similar decimation strategy may be applied to our results as well.

6.2.5 Results

We have developed an algorithm for generating consistent surface meshes from non-binary space partitionings, like an anatomical labeling of a tomographic data set. Figure 6.5 shows a surface

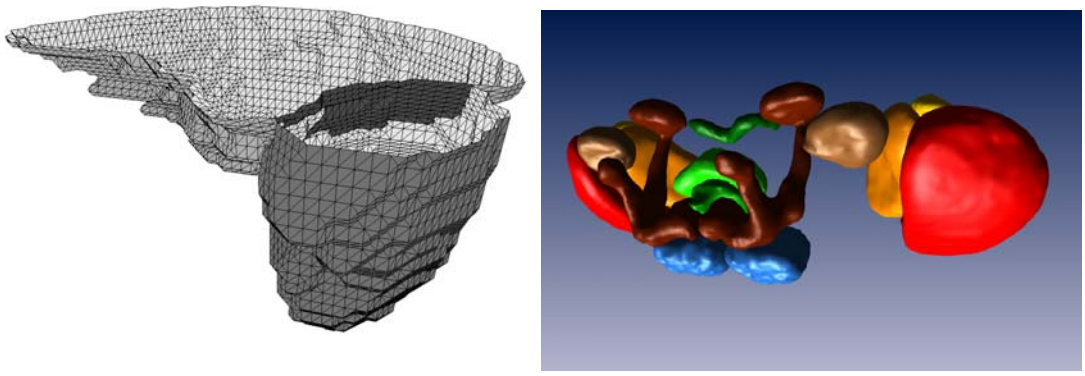


Figure 6.5: The left image shows a surface from a segmentation of kidney and liver. The darker patch represent the common interface between both organs. In the right image the vertices have been adjusted according to sub-voxel weights, yielding much smoother surfaces.

generated using our algorithm applied to a segmentation of kidney and liver. Note how topological information is revealed correctly by the algorithm.

The surface looks relatively un-smooth. Much smoother surfaces can be generated if additional sub-voxel weights are used to shift the vertices correctly. This will be described in the next section. Vertex shifting has been used in the right image in Figure 6.5.

6.3 Computing Weights

Surface smoothness is a very important criterion for a good surface extraction algorithm. For display, the variation of the surface normal is often much more prominent than the actual position of the surface.

As an example compare an iso-surface obtained with marching cubes from a smoothed microscopic recording and the corresponding staircase ("Lego") like surface obtained by drawing the voxel faces, in Figure 6.1. Although the two surfaces geometrically differ less than one voxel edge length (e.g. measured with the Hausdorff distance), the normal directions differ significantly resulting in a completely different visual appearance. Smoothness can also have a significant impact to quantitative investigations like surface area computation.

When surfaces are extracted from image data using conventional iso-surface extraction, the resulting surface often has satisfying smoothness properties. This mainly is attributed to the limited resolution of the image acquisition devices and the related partial volume effect which results in a smoothing of the recorded image. If the recorded image is understood as the sampling of a continuous band-limited function, then the original function can be fully reconstructed from the samples. Instead of the (tri-)linear interpolation used by the marching cubes algorithm a convolution with sinc function has to obtain interpolated values (compare [81, 102]). This way obviously the exact iso-surface of the continuous function could be reconstructed.

The situation is different if the input for surface mesh generation is a binary or, as in our case multi-valued labeling of the volume. Such labels could be the result of automatic segmentation

algorithms which do not operate with sub-voxel accuracy or they can be the result of a manual per-voxel labeling operation. In this case the sub-voxel positioning of the extracted surface meshes cannot be derived from the label data alone. Additional criteria have to be introduced to choose from all surface meshes that consistently separate regions with different labels the one most suitable for the application.

A common approach is to work on a smoothed version of the data (e.g. [130]). A smooth surface can be generated by smoothing the binary labels $\{0, 1\}$, e.g. by convolution with a Gaussian filter, to obtain non-integer labels and extracting the 0.5-isosurface. Note that potentially voxels with original label 0 can be assigned a value greater 0.5 and vice versa, thus resulting in a change of the original labeling. For a non-binary classification, the situation is a little more difficult. Although the labels for different material/tissue types are typically represented as integer numbers, calculating with them like with numbers is wrong. As an example consider a label field containing (among others) the regions *muscle* (represented by the number 1), *liver* (represented by 2) and *kidney*, (3). In this case a smoothing of a boundary region *muscle-kidney* (1-3) could result in voxels being assigned 2 (*liver*) which is obviously senseless. One correct approach is to smooth each region individually using a $\{0, 1\}$-classification and properly combine the results.

Smoothing can also be applied after the extraction of the surface mesh, see e.g. [132, 124, 58] and the references therein.

All such smoothing methods have a significant drawback. They act as a spatial low-pass filter; small features in the label filter can get lost. More generally: The algorithm potentially modifies the label information of a voxel in a way that is not intuitively anticipated by the user. For many applications that involve manual segmentation and especially for medical applications this is not accepted. Additionally loosing consistency between the label information and the inside-outside relationship from the surface mesh can lead to problems with subsequent algorithms.

A different approach is being proposed by Gibson [39, 38] and further developed by Whitaker [137]. Both will be discussed in the next section.

6.3.1 Minimal Surfaces

Gibson [39] uses from all possible surface meshes the one with minimal surface area. She achieves this with a deformable surface approach called constraint surface nets. Starting from an initial (consistent) surface mesh, the mesh is shrunk iteratively while the labeling acts as a constraint to this regularization process.

Whitaker [137] follows the idea of using the minimal surface area as a regularization but develops a method that does not need an explicit surface representation for the smoothing step but instead uses a volume-based approach derived from Level Set theory [113]. We will briefly review his findings.

Level set methods represent surfaces not explicitly but describe them as iso-surfaces (level sets) of a volumetric scalar valued function $\phi : \Re^3 \rightarrow \Re$. Surface deformation may be performed by solving a partial differential equation (PDE) on a discrete sampling of ϕ. We call ϕ an embedding of the surface.

For the problem of extracting a surface from a labeling, we have to find a discrete sampling of a ϕ that is compatible with the labeling.

Let $B_(i,j,k)$ be a binary labeling defined on the discrete domain D (a three dimensional uniform grid). For convenience of notation of the following let B be -1 for voxels inside the object and $+1$ for non-object voxels. An embedding ϕ is compatible with the labeling if it has the same sign as B at the grid points D:

$$\phi(\bar{x}_{i,j,k})B(\bar{x}_{i,j,k}) \geq 0 \quad \bigvee \bar{x}_{i,j,k} \in D \tag{6.2}$$

It is obvious that the zero level set of ϕ would then separate inside and outside voxels from each other, or at most pass through a voxel if $\phi(\bar{x}_{i,j,k}) = 0$.

From all possible functions ϕ a specific one is to be chosen. In [137] minimal surface area is suggested as criterion often leading to smooth surfaces. The combined area of all level sets of a function ϕ is equal to integral of the level set density, which is given by the gradient magnitude of ϕ. From this equation and the constraints (6.2) properties of solution of the minimization problem can be derived which lead to a gradient descent scheme to iteratively determine ϕ from a suitable initial guess (i.e. B) (see [137] and the references therein for details). Let $\hat{\phi}^n_{i,j,k}$ be the discretization of ϕ in the n-th step of the iteration. Then the next step can be computed as:

$$\hat{\phi}^{n+1}_{i,j,k} = \begin{cases} \max(\hat{\phi}^n_{i,j,k} + \Delta t H^n_{i,j,k}, 0) & B_{i,j,k} = 1 \\ \min(\hat{\phi}^n_{i,j,k} + \Delta t H^n_{i,j,k}, 0) & B_{i,j,k} = -1 \end{cases} \tag{6.3}$$

where

$$H^n_{i,j,k} = \frac{\begin{array}{c}\left((\delta_y\hat{\phi}^n_{i,j,k})^2 + (\delta_z\hat{\phi}^n_{i,j,k})^2\right)\delta_{xx}\hat{\phi}^n_{i,j,k} \\ + \left((\delta_x\hat{\phi}^n_{i,j,k})^2 + (\delta_z\hat{\phi}^n_{i,j,k})^2\right)\delta_{yy}\hat{\phi}^n_{i,j,k} + \left((\delta_x\hat{\phi}^n_{i,j,k})^2 + (\delta_y\hat{\phi}^n_{i,j,k})^2\right)\delta_{zz}\hat{\phi}^n_{i,j,k} \\ -2(\delta_x\hat{\phi}^n_{i,j,k}\delta_y\hat{\phi}^n_{i,j,k}\delta_{xy}\hat{\phi}^n_{i,j,k} + \delta_x\hat{\phi}^n_{i,j,k}\delta_z\hat{\phi}^n_{i,j,k}\delta_{xz}\hat{\phi}^n_{i,j,k} + \delta_y\hat{\phi}^n_{i,j,k}\delta_z\hat{\phi}^n_{i,j,k}\delta_{yz}\hat{\phi}^n_{i,j,k})\end{array}}{(\delta_x\hat{\phi}^n_{i,j,k})^2 + (\delta_y\hat{\phi}^n_{i,j,k})^2 + (\delta_z\hat{\phi}^n_{i,j,k})^2} \tag{6.4}$$

The $\delta_x\hat{\phi}^n_{i,j,k}$, $\delta_{xx}\hat{\phi}^n_{i,j,k}$, $\delta_{xy}\hat{\phi}^n_{i,j,k}$, denote approximated first and second order partial derivatives of $\hat{\phi}^n$ which can be computed using centralized differences.

The equations (6.3) and (6.4) define an iteration scheme for finding a suitable embedding $\hat{\phi}$ for the wanted surfaced. The scheme can easily be implemented and converges fairly rapidly. In [137] 20 iterations are reported to produce a sufficiently accurate solution. Note that it is sufficient to compute $\hat{\phi}_{i,j,k}$ in a narrow band in the vicinity of the zero set (see [1, 113]), so that the total computational costs are moderate.

Using this method we have created the surface mesh in Figure 6.6 from a binarized image of a microscopic recording. Comparing it to a marching cubes surface of the binary image we see a significant improvement. However, there are some parts where it seems that a better surface could be found which still is consistent with the label information. The minimal surface spans tightly over the constraints thereby developing sharp edges and corner.

Another shortcoming of the method is the behavior in regions of thickness one. If instead of setting the values to 0 a small positive or negative number $\pm\epsilon$ is used for pixels initially positive or negative, than no features can disappear. However, structures with a thickness of only one voxel will shrink to (almost) zero. Whitaker has pointed out himself, that the criterion of minimal surface

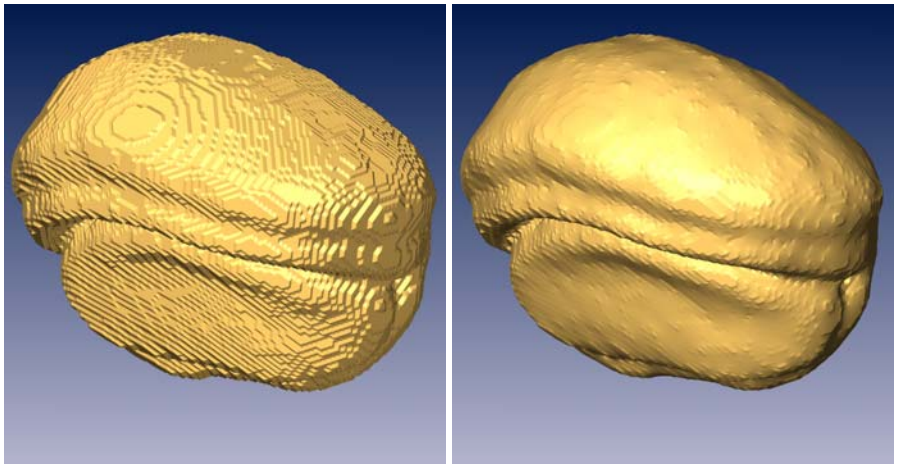


Figure 6.6: The left image shows a surface reconstruction from a binarized image of the optic lobe data set without additional weights or smoothing constraints. Note that only 12 different surface normal directions occur. The right image shows how the minimal surface area approach improves the quality of the resulting surface. However, significant staircase artifacts still remain, like in the upper left part of the data set.

area does not necessarily correspond to maximal surface smoothness and is probably not the best criterion that could be found. Though [137] is not the final answer to the smoothing problem it gives rise to interesting ideas that will be discussed in the next section.

6.3.2 Constrained Smoothing

In the above discretized iteration rule, we see that in each step (not surprisingly) the value at each voxel is replaced by a combination of the voxels from its 26-neighborhood. The constraints are incorporated by the min and max functions in Equation (6.3) which reset the $\hat{\phi}_{i,j,k}$ to the valid interval ($\Re^+$ or $\Re^-$) after each step if necessary. An analogous scheme of assuring the constraints can of course be used as well in other iteration schemes or filter passes, like smoothing with Gaussian filter kernels, leading to a procedure we call *constrained smoothing*.

In its simplest version the method reads:

$$\hat{\phi}_{i,j,k} = \begin{cases} \max((G \circ B)(\bar{x}_{i,j,k}), 0) & B_{i,j,k} = 1 \\ \min((G \circ B)(\bar{x}_{i,j,k}), 0) & B_{i,j,k} = -1 \end{cases} \tag{6.5}$$

$G\circ$ is a smoothing operator, typically a convolution with a Gaussian filter kernel, which is applied to the binary image B.

$$G(\vec{x}) = \frac{1}{(2\pi\sigma)^{\frac{3}{2}}} \quad e^{-\frac{|\vec{x}|^2}{2\sigma}} \tag{6.6}$$

In a practical implementation the convolution with G is approximated by a finite discrete filter kernel $g_{i,j,k}$. We have used a $5\times5\times5$ kernel.

$$(G \circ B)(\bar{x}_{i,j,k}) \approx \sum_{\Delta_i=-2}^{2} \sum_{\Delta_j=-2}^{2} \sum_{\Delta_k=-2}^{2} g_{i,j,k} B(\bar{x}_{i+\Delta_i, j+\Delta_j, k+\Delta_k}) \tag{6.7}$$

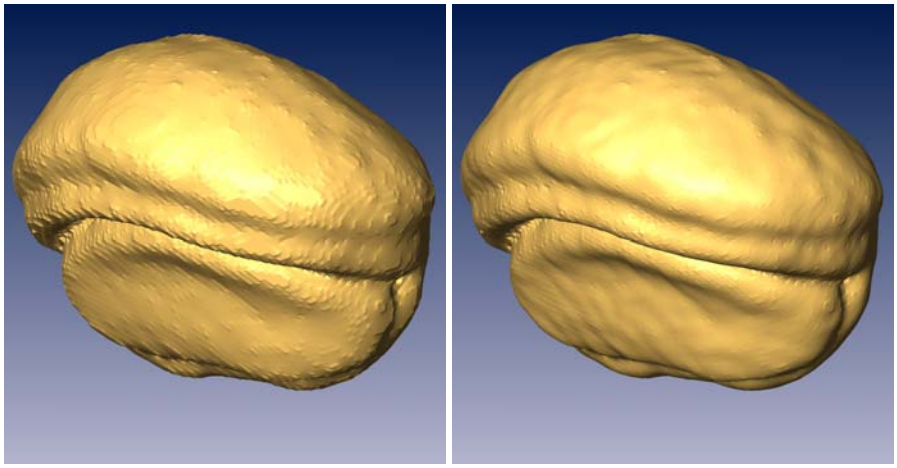


Figure 6.7: Comparison of the minimal surface approach (left) with the surface obtained by *constrained smoothing* (right). While both images are still fully consistent with the labeling information, the surface in the right image is significantly smoother and exhibits much less aliasing effects.

Appropriate boundary treatment has to be assured.

The minimal surface constraint that lead to Equation (6.3) is complete in the sense that no additional parameters are needed. For a Gaussian filter kernel a filter width has to be specified. However, for the minimal surface method the properties and appearance of the resulting surface highly depend on the resolution of the input field, i.e. the "resolution of the constraints". Similarly we have coupled the filter parameters to the voxel-size. If the linear voxel size is v in one dimension (the smallest in case of non-isotropic voxels) we have chosen $\sigma = 2v$ for all examples shown in this work.

We have applied this new method to the same data set as shown before. As can be seen in 6.7 the results are convincing. However, there is still room for improvement. In particular we have observed and addressed two issues:

1. As can be seen in the example in Figure 6.7 there are a number of little "scars" and "pimples" on the surface, which subjectively may seam to be a little too prominent. Most of them stem from one or two voxel sized features, that might be considered noise.

2. Structures of one voxel thickness will shrink to (almost) zero thickness, just enough to still contain the voxel center, and thus effectively disappear, compare Figure 6.8. This is not a theoretical problem: Especially in medical images with anisotropic voxels, some features can often be recognized and labeled in a single slice only leading to one voxel thick structures. Another example are the dendritic structures of neuronal cells.

Both methods, the minimal surface as well as the *constrained smoothing* behave similar with respect to these issues.

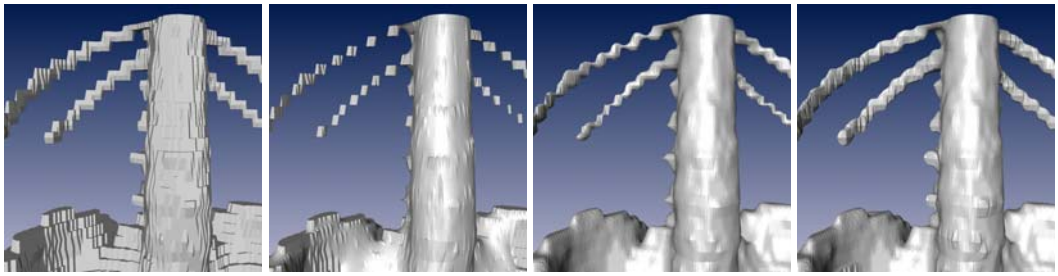


Figure 6.8: The images show the bones extracted from a clinical CT recording of a human's torso including two ribs. The large slice distance is typical for such recordings. While the two ribs are still well connected in the "binary" surface (left), they partly shrink to zero thickness after smoothing. The problem is similar for minimal surface (second) and constraint smoothing (third). Due to identification and special treatment of thin structures in the right image the ribs are nicely preserved.

6.3.3 Surface Noise

Figure 6.9 shows a closeup of a one-pixel feature configuration and a two dimensional sketch of the situation. We see that the tip of the little pyramid cannot be moved further down without violating the labeling constraints. This is typically one of the voxels that has to be forced to maintain its sign during the iteration. On the other hand the influence of the single elevated pixel is not sufficient to let the vertices on the surrounding edges deviate significantly from their undisturbed position in the middle. Notice that it clearly depends on the application if this behavior is desired or not. If the goal is a particularly smooth surface, it is probably not. We therefore propose a way to eliminate most of these effects:

From the above observations it follows that voxels which potentially cause this type of artifacts are those voxels for which the value had to be set to zero in the iteration. In the following discussion we will consider an initially negative (inside) voxel $\vec{v}$ in a neighborhood of positive (outside) voxels, such that the convolution at this point is significantly positive.

$$B(\bar{x}_{i,j,k}) = -1 \qquad (G \circ B)(\bar{x}_{i,j,k}) > 0$$

A typical value is $(G \circ B)(\bar{x}_{i,j,k}) = 0.2$. It is intuitively understandable that forcing the value of such a voxel to 0 leads to visual unsteadiness. Since the value of the voxel $\bar{x}_{i,j,k}$ must remain negative according to the labeling, the only way to produce a smoother surface at this location is to decrease the values of the neighboring voxels. This can be achieved by setting the initial value at

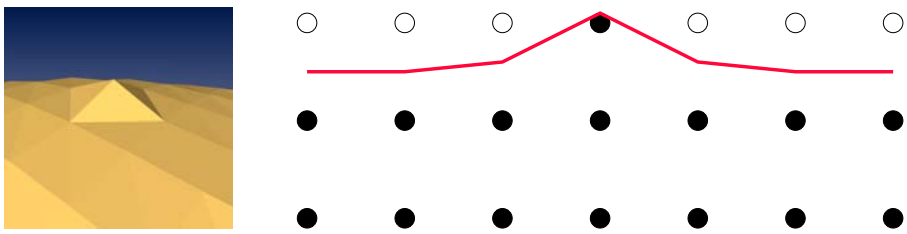


Figure 6.9: A one-pixel-feature on the 3D surface and two dimensional sketch of the situation. The top of the pyramid cannot move further towards the base surface without violating the constraints imposed by the labeling.

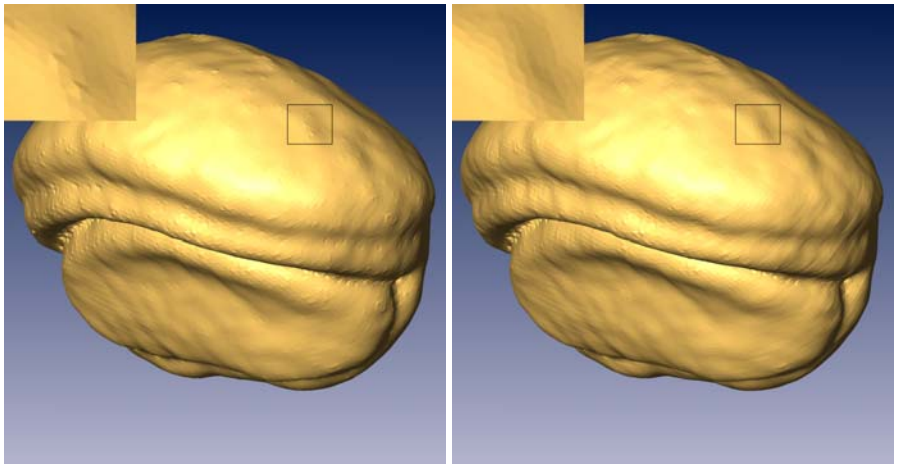


Figure 6.10: Single (few)-voxel features on the surface can disturb smooth appearance (see magnification in left image). If this is undesired an additional smoothness model turns the little spikes into larger but less obtrusive swellings (right).

$\bar{x}_{i,j,k}$ before the convolution not to -1 but to add an additional (negative) offset value $o_{i,j,k}$, thus increasing its negative impact (not pejorative here) on its neighborhood.

Let N be the set of such "noise" voxels. We consider a voxel to be in N if it was forced to 0 by the min/max function and if there are only few other such voxels in its vicinity. As a criterion for *few* we have used: less than 6 in the 125 neighborhood.

To completely avoid the unsteadiness at these points due to reseting to 0, the offset $o_{\bar{x}_{i,j,k}}$ has to be large (small) enough that the convolved value at $\bar{x}_{i,j,k}$ becomes zero.

We do not want to add an offset to voxels not in N. We demand:

$$o(\bar{x}_{i,j,k}) = 0 \quad \bigvee \bar{x}_{i,j,k} \notin N \tag{6.8}$$

$$(G \circ (B + o))(\bar{x}_{i,j,k}) = 0 \quad \bigvee \bar{x}_{i,j,k} \in N \tag{6.9}$$

$$\Leftrightarrow (G \circ o)((\bar{x}_{i,j,k}) = -G \circ B \quad \bigvee \bar{x}_{i,j,k} \in N \tag{6.10}$$

The right hand side of Equation (6.10) has been computed in the first step. $(G \circ o)((\bar{x}_{i,j,k})$ is according to Equation (6.7) a linear combination of the 125-neighbors of $(\bar{x}_{i,j,k})$, most of which are zero however. The coefficients are defined by $g_{i,j,k}$. In effect we have to solve a linear system with $|\text{card}(N)$ variables (card(N) denotes the number of elements in N). In usual data sets of resolution 512^3 this number has typically been less than 5000, often less than 1000. According to our definition of a "noisy" voxel, every voxel in N has at most 5 neighbors in N, thus every row of the matrix defining the linear equation system has at most five entries. Due to the symmetry of $g_{i,j,k}$ the linear equation system is symmetric. Such a sparse symmetric linear equation system can be solved most efficiently with an iteration method, like e.g. the conjugate gradient method.

In practice many voxels in N do not have neighbors at all. Thus the offset for these voxels can be computed directly by dividing the right hand side of Equation (6.10) by $g_{0,0,0}$.

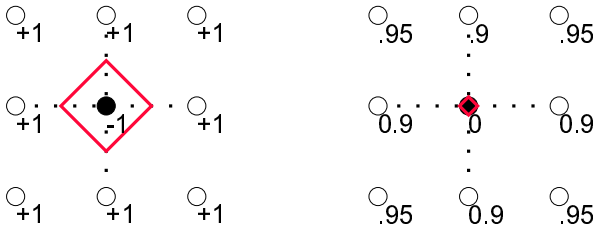


Figure 6.11: The Figure shows the a thin object (black object voxel) and the corresponding initialization of the distance function. The intersection points on the connecting edges at the zero-crossing of the linear interpolation. After the convolution (right) the middle pixel has to be artificially set back to zero (to avoid positive values for object voxels). The neighboring background voxels are only little affected. The resulting contour shrinks to the voxel center.

Typical values for the offset are in the order of -15.

The complete algorithm with noise correction is then:

1. Initialize $B_{i,j,k}$ from the labeling.
2. Compute $r_{i,j,k} = (G \circ B)(\bar{x}_{i,j,k})$ and mark voxels where $r_{i,j,k} B_{i,j,k} < 0$ as potential noise.
3. Identify noise voxels and collect in N.
4. Solve linear equation system $(G \circ o)(\bar{x}_{i,j,k}) = -r_{i,j,k} \qquad \bar{x}_{i,j,k} \in N$
5. Replace $B_{i,j,k}$ with $B_{i,j,k} + o_{i,j,k}$ and compute $\hat{\phi}_{i,j,k}$ according to Equation (6.5).
6. Extract zero-set from $\hat{\phi}$.

Results are shown in Figure 6.10. The little spikes almost completely disappear, resulting in a significantly increased overall smoothness.

6.3.4 Thin Structures

The shrinking of thin structures can be understood from the sketch in Figure 6.11. As pointed out above this behavior is often undesired. In order to improve this, first such thin structures have to be identified.

In order to identify thin parts of the object a morphological opening operator (an erosion followed by a dilatation) can be used, see e.g. the text book [120] for an overview. Voxels that are part of the original image but are removed by the operator are considered to be part of thin structures. A standard erosion operator based e.g. on a $18-$ or $26-$ neighborhood would not only remove structures of thickness 1 but also those which are two voxels thick. Thus too many voxels would be classified thin.

Therefore, we propose to work on the dual grid as follows: Construct a binary image $B^{\perp}$ with data values not defined at the voxel centers but on the voxel corners (if the voxels were little boxes). A voxel in $B^{\perp}$ is set if at least one one of the adjacent voxels in B is set. Then the opening procedure is applied. When converting the image back to the original grid a voxel is set only if all its eight

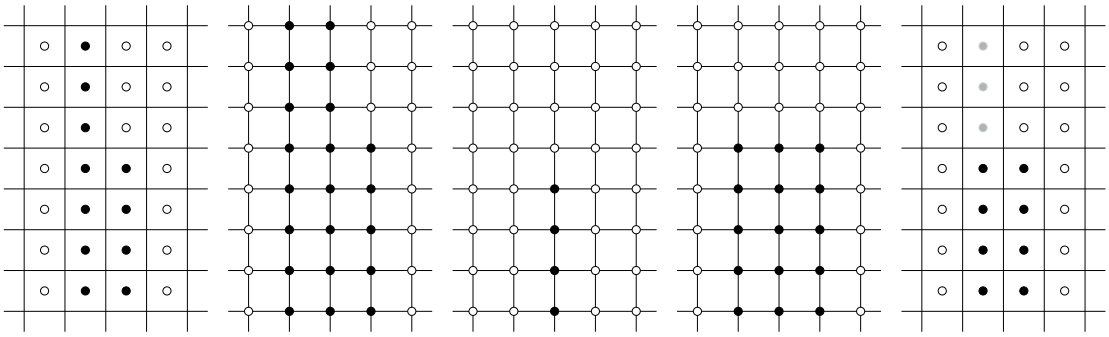


Figure 6.12: The figure depicts the algorithm for identification of thin voxels. From left to right the images depict (i) the binary input image, (ii) converted image to dual grid, (iii) result of 26(8)-neighbor-erosion on the dual grid, (iv) result of dilatation, and (v) object voxels which are still present after converting back (black) and those classified as thin (gray).

adjacent dual grid voxels are set. The full procedure is depicted (in two dimensions) in Figure 6.12. We have used morphological operators based on the 26-neighborhood.

Using this algorithm we are able to identify voxels at which the object is only one voxel thick. To avoid complete shrinking at these points (cf. 6.11) additional constraints are imposed. Namely instead of restricting the values for $\tilde{B}$ to $\Re^-$ we choose a smaller maximum value, i.e. $\tilde{B}$ can be fixed to -1 at such voxels.

6.3.5 Results

We have proposed a method to compute sub-voxel positions for the vertices of a polygonal surface extracted from a labeling. The approach does not violate the constraints imposed by the labeling in the sense that the centers of two voxels labeled differently will always be separated by a surface. The method is fast and easy to implement and the results are very well suited for anatomical objects. Results have been shown above and more results will be given throughout the remainder of this work.

6.4 From Graphs to Geometry

6.4.1 Introduction

The graph-representations that we have extracted from the neuronal cells or from vascular networks can be visualized directly (Section 3) and statistical analysis methods typically even require such a representation. Therefore, the need of converting these graphs into surface meshes is less urgent than in case of the label fields. Nevertheless, especially for high-end visualization, illustration purposes, and maximal compatibility for exchange of models it is desirable to be able to generate polygonal surface descriptions of these structures as well.

The given models consist of a set of points (nodes) which are connected by line segments. We assume a thickness information (diameter) to be defined at the nodes. The number of edges incident to one particular node can be one (end point), two (inner point of a branch), or greater than

two (branching point). We assume the diameter to vary linearly along ond edge. As a specialty in the data sets studied in the course of this work, some of the models contain varicosities ("blebs"). Although their biological function is not yet fully understood there are evidences that they have an important functional meaning and are therefore important to be visualized. The varicosities can geometrically be modeled by spheres. In order to obtain a visually smooth joining of the cylinders, additional spheres are placed at the nodes. In terms of computational solid geometry (CSG) our model can thus be described as the union of a set of spheres and conical frustrums (cylinders with varying radius).

There are two fundamentally different approaches to converting an analytical CSG representation of a model into a polygonal boundary representation: Direct or constructive algorithms explicitly triangulate parts of the primitive boundaries that form the model and try to stitch the patches together where primitives touch. The second class of algorithms use implicit surfaces. They compute an intermediate scalar field $F : \Re^3 \rightarrow \Re$, typically defined on uniform grid, such that the wanted surface is an iso-surface of F. A typical example for F is the signed distance field that encodes for each point in space the distance to the nearest surface point. For points inside the object the negative distance is used. Obviously, the iso-surface with iso-value zero will approximate the searched boundary of the CSG model. See e.g. [13, 33, 59] and the references therein.

For our problem we have chosen the second class of algorithms for two reasons: Compared to the constructive approach consistency problems due to limited arithmetic accuracy can be avoided more easily and the method allows easier extension to other primitives.

Although the iso-surface of the exact distance function is the exact object boundary, in most practical implementations there are two sources of approximation errors: The distance function is only approximated by a (tri-)linear interpolation of its uniform distributed samples. Second the iso-surface of this function is only approximated by a mesh of limited resolution. Both problems have bee addressed in a number of ways. In [33] and adaptive method is proposed that samples the distance function at a higher resolution where it cannot be accurately represented by linear interpolation of the coarse grid. In [114] an adaptive resulting mesh resolution is achieved by starting at a fine resolution and adaptively merging neighboring cells if a certain error criterion is not exceeded. In [59] additional points are added to the original marching cubes triangulation in cells where needed according to a feature detection heuristic. Additionally a tri-variate distance function is used to more accurately describe the shape of the objects.

Similar to the basic ideas in [33] and [59] we have developed an octree based method to efficiently triangulate the neuron-cell models at high resolution, taken into account the sparse nature of the models.

6.4.2 Algorithm

Our algorithm starts by building an octree data structure:

1. Compute the bounding box of the data set and initialize the octree data structure.
2. Insert the primitives into the octree. Thereby a primitive is inserted into all octree nodes that it intersects. A node is subdivided if it contains at least one primitive until a maximal depth is reached.

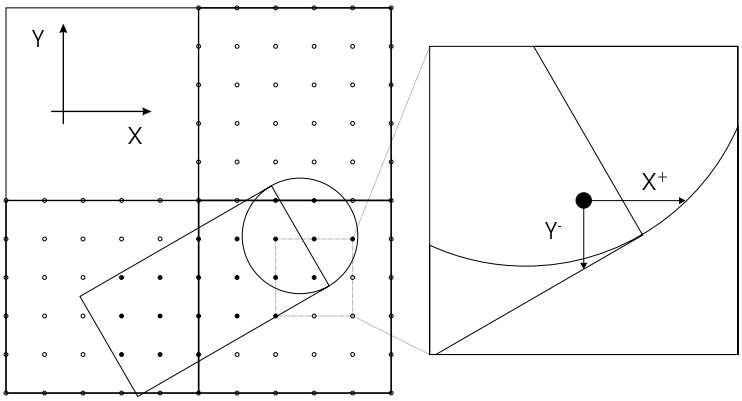


Figure 6.13: Illustration of the geometry extraction procedure for a sphere and a cylinder. Each of the three non-empty octree nodes is subdivided by a grid $g_{i,j,k}$. For each inner grid point (filled) the six directed distance values to the next primitive boundary are computed. The component wise maximum over all primitives is taken. E.g. for the gray-filled point, the x^+ distance will be taken from the sphere, while the y^- is that of the cylinder.

Note that each non-empty leaf node of the octree is on the same, maximal depth. In the next step a triangulation is computed for each non-empty leaf node.

In each non-empty leaf node of the octree we generate a uniform spaced grid $g_{i,j,k}$. For each point $g_{i,j,k}$ we compute the inside/outside flag by testing it against each primitive in that octree node (cf. 6.13).

Obviously, the boundary surface to be extracted will pass between two neighboring grid points if they have different inside/outside flags. In order to compute the location of the intersection on the edge we do not want to rely on the zero-crossing of a linearly interpolated distance function. Instead, we compute for each inner grid point which is adjacent to an outer point, the six directed distance values for each of the six major axis directions $x^{\pm}$, $y^{\pm}$, and $z^{\pm}$, similar to the trivariate distance function in [59]. The directed distance for a point $\vec{p}$ in direction $\vec{d}$ encodes distance between p and the nearest intersection of the directed ray $(\vec{p}, \vec{d})$. This six-dimensional distance vector is computed against each primitive and the component-wise maximum is taken. This is illustrated in Figure 6.13. The maximal distance directly determines the position of the intersection on the edge.

At this stage for each cell $(g_{i,j,k}, g_{i+1,j+1,k+1})$ of the fine grid the inside/outside configuration of the eight corners is known as well as the exact location of possible edge intersections. Therefore, a simple table based method like the marching cubes can be used to triangulate the cells.

The result of the algorithm applied to a simple test data set is shown in 6.14.

6.4.3 Algorithmic Details and Implementation

As mentioned above the algorithm is flexibly extendible to other geometric primitives. For each primitive only three functions have to be implemented:

1. Point inside primitive test.

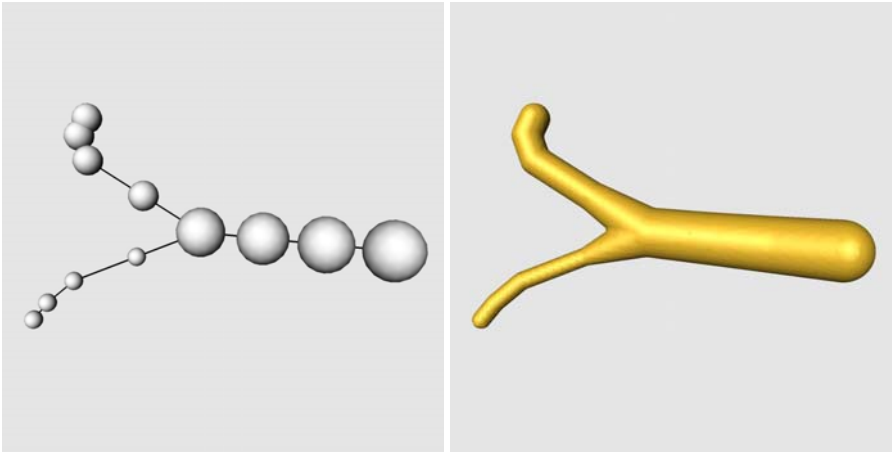


Figure 6.14: The figure shows the method for a (coarse) artificial test data set. Left: Graph structure along with the thickness information visualized by spheres. Right: Generated surface.

2. Primitive/box intersection test.
3. Computation of the six directed distance values for an inner point $\vec{p}$.

Instead of an exact implementation of the box intersection test efficiency may be increased by using a simplified version that can generate false-positive answers, as long as no real intersection is over-seen.

The relevant formulae for a sphere with center $\vec{c}$ and radius r can be derived using Pythagoras theorem:

$$\begin{aligned}
\vec{p} \in sphere(\vec{c}, r) &\Leftrightarrow |\vec{p} - \vec{c}|^2 < r^2 \\
dist_{x^\pm}(\vec{p}) &= \pm(p_x - c_x) \pm \sqrt{r^2 - (p_y - c_y)^2 - (p_z - c_z)^2} \\
dist_{y^\pm}(\vec{p}) &= \pm(p_y - c_y) \pm \sqrt{r^2 - (p_x - c_x)^2 - (p_z - c_z)^2} \\
dist_{z^\pm}(\vec{p}) &= \pm(p_z - c_z) \pm \sqrt{r^2 - (p_x - c_x)^2 - (p_y - c_y)^2}
\end{aligned}$$

For the conical case the expressions get a little more lengthy. Let a cone frustrum be defined by the centers $\vec{c_1}$ and $\vec{c_2}$ of the two circles and their radii r_1 and r_2.

With the following definitions:

$$\alpha = \frac{r_2 - r_1}{|\vec{c_2} - \vec{c_1}|} \qquad H = |\vec{c_2} - \vec{c_1}| \qquad \vec{a} = \frac{\vec{c_2} - \vec{c_1}}{|\vec{c_2} - \vec{c_1}|}$$

we get

$$\begin{aligned}
\vec{p} \in frustrum(\vec{c_1}, \vec{c_2}, r_1, r_2) \Leftrightarrow \quad & 0 < (\vec{p} - \vec{c_1}) \cdot \vec{a} < H \\
\wedge \quad & \sqrt{|\vec{p} - \vec{c_1}|^2 - ((\vec{p} - \vec{c_1}) \cdot \vec{a})^2} < \alpha(\vec{p} - \vec{c_1}) \cdot \vec{a} + r_1
\end{aligned}$$

In order to compute the directed distances we introduce some further definitions:

$$\begin{aligned} v &= (p_y - c_{1,y})a_y + (p_z - c_{1,z})a_z \qquad n = (\alpha^2 + 1)a_z^2 - 1 \\ p &= \alpha^2 a_x v + \alpha r_1 a_x + v a_x \\ d_{1,2} &= -p \pm \sqrt{p^2 - (\alpha^2 v + 2\alpha r_1 v + r_1^2 - (p_y - c_{1,y})^2 - (p_z - c_{1,z})^2 + v^2)} \\ d_3 &= \frac{H - v}{p_x - c_{1,x}} \qquad d_4 = \frac{-v}{p_x - c_{1,x}} \end{aligned}$$

Then the directed distance values at point $\vec{p}$ can be computed as

$$dist_{x^+}(\vec{p}) = min\{d_i | d_i > p_x - c_{1,x}\} - (p_x - c_{1,x})$$
$$dist_{x^-}(\vec{p}) = (p_x - c_{1,x}) - max\{d_i | d_i < p_x - c_{1,x}\}$$

The values for $dist_{y^\pm}(\vec{p})$ and $dist_{z^\pm}(\vec{p})$ can be derived analogously. Expressions for other primitive shapes could easily be derived.

6.4.4 Results

The described method efficiently generates consistent surfaces from analytical geometrical descriptions of neuronal trees. An example is shown in Figure 8.6, in Section 8. Details on the model are given in the figure caption.

As opposed to a surface composed of independent meshes for each primitive, our surfaces are one-layered, consistent and closed. Therefore, they can be used for high-quality shading (including transparent rendering), analysis like volume or surface rendering and could even be used for tetrahedral grid generation if desired.

Due to the fine details combined with a large overall extend of the objects, the surface meshes contain a large number of triangles. Therefore, they are especially useful for still images, while for interactive display a line-based rendering method will in most cases be preferable. Especially in the still images though the surface shading helps to understand shape and depth, which in an interactive setting can be retrieved by rotating the object. For a realistic appearance a good radius estimation is important, which can be a problem in very noisy images.

An interesting option would be to combine the high-quality surface rendering with a line-based rendering adaptively, based on a view dependent criterion.

The shapes that are relevant for this work are relatively smooth. If the method is used for models with sharp edges or corners, artifacts can appear at these features. Such problems can efficiently be reduced by inserting additional points in cells containing such features in the triangulation step as proposed by [59]. In a post-processing step edge-flips can be performed to connect points on edge features. A feature detector can be based on the surface normal at the intersection points of the surface with the grid $g_{i,j,k}$. Note that the surface normal at these points can easily be computed analytically.

Instead of refining the leaf nodes we could have chosen a larger maximal depth value for the octree. However, we have found that this generates an unnecessary overhead due to the large number of octree nodes which is not compensated by the slightly larger amount of empty volume that can be skipped.