

Interactive 3D Medical Image Segmentation with Energy-Minimizing Implicit Functions

Frank Heckel^{a,*}, Olaf Konrad^b, Horst Karl Hahn^a, Heinz-Otto Peitgen^a

^aFraunhofer MEVIS, Universitaetsallee 29, 28359 Bremen, Germany

^bMeVis Medical Solutions AG, Universitaetsallee 29, 28359 Bremen, Germany

Abstract

We present an interactive segmentation method for three-dimensional medical images that reconstructs the surface of an object using energy-minimizing, smooth, implicit functions. This reconstruction problem is called variational interpolation. For an intuitive segmentation of medical images, variational interpolation can be based on a set of user-drawn, planar contours that can be arbitrarily oriented in 3D space. This also allows an easy integration of the algorithm into the common manual segmentation workflow, where objects are segmented by drawing contours around them on each slice of a 3D image.

Because variational interpolation is computationally expensive, we show how to speed up the algorithm to achieve almost real-time calculation times while preserving the overall segmentation quality. Moreover, we show how to improve the robustness of the algorithm by transforming it from an interpolation to an approximation problem and we discuss a local interpolation scheme.

A first evaluation of our algorithm by two experienced radiology technicians on 15 liver metastases and 1 liver has shown that the segmentation times can be reduced by a factor of about 2 compared to a slice-wise manual segmentation and only about one fourth of the contours are necessary compared to the number of contours necessary for a manual segmentation.

Keywords: Interactive Segmentation, Contouring, Variational Interpolation, Radial Basis Functions, Medical Imaging, 3D

1. Introduction

In medical imaging, automatic segmentation is a challenging task and it is still an unsolved problem for many medical applications due to the wide variety of image modalities, scanning parameters and biological variability. Manual segmentation is time-consuming and frequently not applicable in clinical routine. Therefore, semi-automatic segmentation methods, i.e., methods which require user interaction, can be used in cases where automatic algorithms fail.

A wide range of semi-automatic segmentation methods exists that can roughly be classified into *voxel-based* methods, where the user draws *seed points* to define fore- and background voxels [1, 2], and *surface-based* methods, where the shape of an object is reconstructed based on *contours* or *object models* [3, 4, 5, 6]. A surface reconstruction approach that can also be used for 3D image segmentation is based on *implicit functions*.

In computer graphics, a surface reconstruction using energy-minimizing implicit functions is referred to as thin-plate-spline interpolation (in 2D) or variational interpolation (in 3D) [7]. Variational interpolation generates a smooth C^1 - or C^2 -continuous surface based on a set of unordered points (a so-called *point cloud*). Using this approach, the surface is represented by an implicit function. Because contours can be seen as a set of points as well, variational interpolation can be used for shape reconstruction and thus segmentation of objects in multi-dimensional medical images from a sparse set of user-drawn

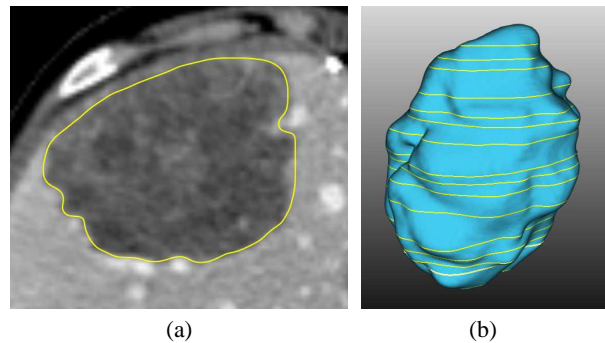


Figure 1: Example for segmentation of a liver metastasis in CT using 17 parallel contours: (a) shows a user-drawn contour in one slice, (b) shows the interpolation result in 3D. Also notice the cap on top of the segmentation in (b), where the segmentation is smoothly closed, even though no contours were drawn there.

contours. In contrast to other algorithms like the shape-based interpolation [8], the reconstruction algorithm works on arbitrarily oriented contours. The contours can be drawn freehand, by using algorithms like *live-wire* [4] or any other 2D contouring tool. Because of its interpolation character, the reconstruction algorithm guarantees that all contours given by the user are part of the surface. Moreover, it also extrapolates beyond the contours in a plausible way (see Fig. 1).

In this paper we discuss variational interpolation for interactive segmentation of medical images with respect to applications in tumor and liver segmentation from CT scans. Our goal is to support the *contour-based workflow*, where objects are segmented by drawing contours along its border and which

*Corresponding author

Email address: frank.heckel@mevis.fraunhofer.de (Frank Heckel)

	Semi-Automatic	Interactive
Segmentation process	“Fire and forget”	Iterative
Role of user	Initialization and parametrization	Steering and correcting
Manipulation / Input	Direct or indirect	Direct
Performance requirements	Moderate	Fast (real-time response)

Table 1: Differences between semi-automatic and interactive segmentation.

is a common way to manually segment objects in medical images. Using variational interpolation, a 3D segmentation can be calculated based on the 2D information given by the user. So far, variational interpolation has not been used for such interactive 3D medical image segmentation tasks. Reasons for this might be the computational complexity of the algorithm and the robustness of the algorithm to contradictory user inputs and complex contours. We will discuss both issues in detail.

In Sec. 5 we show how to speed up the algorithm to achieve almost real-time calculation times while preserving the overall segmentation quality. The performance improvements are based on a shape-preserving reduction of the number of contour points and a fast voxelization strategy for the resulting implicit function. A significant speedup is achieved by the parallelization of the algorithms, utilizing modern 64-bit multi-core CPUs. We also discuss a local interpolation scheme based on radial basis functions with compact support. In Sec. 6 we describe how to make the interpolation algorithm more robust to self-intersecting contours. Finally, we discuss how to improve the robustness of the algorithm by transforming it from an interpolation to an approximation problem. Results as well as a first evaluation on 15 liver metastases and 1 liver are given in Sec. 7 and Sec. 8.

2. Semi-automatic vs. interactive segmentation

Interaction in segmentation of medical images has been investigated by Olabbarriaga and Smeulders [9]. For segmentation tasks, both *semi-automatic* and *interactive* refers to algorithms that require some kind of user input. From our point of view, interactive segmentation differs from semi-automatic segmentation in the role of the user and thus in the algorithmic requirements especially concerning the computation times and the interface used for human-computer-interaction.

In contrast to semi-automatic segmentation, an interactive segmentation puts the user into an essential role during the segmentation task. While in semi-automatic segmentations the user input is typically used for initialization of some automatic algorithm, in interactive segmentation, the segmentation is an iterative process where temporary results are generated from each user input to give the user a direct feedback on the result generated by the algorithm based on the current parameters derived from the input data. This allows the user to evaluate the

result and to directly react on it. As a consequence, an interactive segmentation algorithm needs to be fast enough to allow this kind of (at least almost) real-time feedback. A well-known example for this class of segmentation algorithms in 2D is the live-wire algorithm. Table 1 compares the differences between semi-automatic and interactive segmentation.

In addition to the already discussed requirements, the process of interactive segmentation also poses requirements on the visualization of intermediate results and on the human-computer-interaction, which are both essential for an intuitive and efficient use of an interactive segmentation algorithm. Both fields have been well studied over the past years but are out of the scope of the current paper. The interested reader is referred to Preim and Dachsel [10].

3. Related work

In computer graphics, variational interpolation is a well-known method for smooth surface reconstruction from unorganized point clouds based on *radial basis functions* (RBFs). One of the first papers on variational interpolation by Turk and O’Brien focuses on shape transformation, i.e., transformations between different 3D objects over time, using variational implicit functions [11]. Later the authors discuss modeling objects with such implicit surfaces [12]. RBFs have also been used in other contexts, like object reconstruction [13, 14, 15] or automatic mesh repair [13]. Many authors address the robustness [13, 14, 15, 16, 17, 18] and performance [13, 19, 20] of object reconstruction with RBF-based implicit functions.

In medical image processing, the thin-plate-spline interpolation is an established method for elastic registration of images from different modalities or points of time [21, 22]. Some authors use RBFs for (semi-)automatic model-based segmentation where variational interpolation is combined with active contours or level sets. Morse *et al.* use active contours with a constraint-based implicit representation for segmenting objects in 2D medical images [23]. A combination of RBFs with active contours for segmentation of 2D ultrasound, CT and MRI data is also proposed by Slabaugh *et al.* [24]. Freedman *et al.* use level sets and RBFs for segmentation of the prostate in 3D CT data [25]. An RBF based level set approach is also used by Gelas *et al.* for segmentation of different tissues in 2D ultrasound images and for segmentation of the calcaneus bone in 3D CT data [26]. Wimmer *et al.* use an RBF-based surface reconstructed from a few user-drawn contours on 2D multi-planar reconstructions to initialize a level set algorithm [27]. Another application of RBFs is fitting surfaces to anatomical structures like the skull as shown by Carr *et al.* [28].

Although some authors suggest variational interpolation for medical image segmentation [11], Masutani is the first and only author who explicitly focuses on RBFs for data-driven segmentation of volumetric medical images [29]. Although the paper mentions semi-automatic liver segmentation using the RBF-approach, it does not use it in a semi-automatic manner, i.e., in terms of incorporating user interaction. Mory and Ardon suggest RBFs that are designed according to features in the image [30]. Their approach was applied to different 3D medical image

segmentation tasks using an interactive framework that allows the user to interactively add control points, which define foreground and background voxels. Mory and Ardon report a response time of 1 second for images with 256^3 voxels. However, the algorithm only works interactively for a few constraints and thus it would not be applicable for our segmentation problem, where we have to deal with thousands of constraints.

In conclusion, looking at the literature, variational interpolation has not yet been used for *contour-based interactive segmentation* of 3D medical images.

Another approach for interpolation between slice-wise binary segmentations is the *shape-based interpolation* developed by Raya and Udupa [8]. By generating binary segmentations from contours, this approach can also be used for some kind of object reconstruction, but in contrast to variational interpolation, the surface of the generated segmentation is not smooth and it cannot be used for arbitrarily oriented contours. This approach has been used in combination with live-wire by Schenk *et al.* for segmentation of the liver [31].

Finally, object reconstruction from a set of arbitrarily oriented contours can also be done directly in the surface-mesh (i.e., triangle-mesh) domain, as presented by de Bruin *et al.* and Liu *et al.* [32, 33]. But as our goal is a segmentation of the object in image space, we would still need to voxelize the resulting mesh. The main advantage of implicit functions over a reconstruction in the surface-mesh domain is that special cases like bifurcations (i.e., having a different number of contours in two adjacent slices for example) and multiple objects do not have to be handled explicitly. Moreover, it is not that easy to guarantee a C^2 -continuous reconstruction in the surface-mesh domain.

4. Variational interpolation

Implicit functions are a way to represent objects. Using an implicit function, the surface of an object is defined by all points in space that evaluate to 0 when inserted into the implicit function. If the implicit function is created based on generalized *thin-plate splines*, it is called a *variational implicit function* [12]. Variational implicit functions are at least C^1 -continuous, i.e., they are smooth. Using variational implicit functions, the interpolation problem can be solved in any dimension [11]. We call this *variational interpolation*, while in 2D it is called *thin-plate interpolation*. This means, given a set of *constraint points*, which are points on the surface of the object, a smooth implicit surface can be created that passes through each constraint point.

The variational interpolation defines an implicit function $f(\mathbf{x})$ that fulfills all constraints while it minimizes an *energy function* E that measures the smoothness of $f(\mathbf{x})$. For a C^1 -continuous interpolation in 3D, E is defined as

$$E = \int_{\Omega} f_{xx}^2(\mathbf{x}) + f_{yy}^2(\mathbf{x}) + f_{zz}^2(\mathbf{x}) + 2(f_{xy}^2(\mathbf{x}) + f_{xz}^2(\mathbf{x}) + f_{yz}^2(\mathbf{x})) d\mathbf{x}, \quad (1)$$

with Ω being the region of interest in which the interpolation shall be computed. $f(\mathbf{x})$ is called the *thin-plate solution*. Using an appropriate RBF $\phi(\mathbf{x})$, the interpolation function $f(\mathbf{x})$ can be

written as

$$f(\mathbf{x}) = P(\mathbf{x}) + \sum_{j=1}^k w_j \phi(\mathbf{x} - \mathbf{c}_j), \quad (2)$$

where $\mathbf{c}_j = (c_j^x, c_j^y, c_j^z)$ are the points in 3D space where the function is constrained to have a specific value, w_j are the weights of each RBF and $P(\mathbf{x})$ is a degree-one polynomial that accounts for the linear and constant portions of $f(\mathbf{x})$. According to Carr *et al.* [13], a commonly used RBF in 3D that minimizes Eq. 1, is the *biharmonic spline*

$$\phi(\mathbf{x}) = \|\mathbf{x}\|. \quad (3)$$

We use the *triharmonic spline* in this paper, which is another commonly used 3D RBF, because it results in a C^2 -continuous and thus smoother interpolation [21]. It is defined by

$$\phi(\mathbf{x}) = \|\mathbf{x}\|^3. \quad (4)$$

$f(\mathbf{x})$ must fulfill the constraints \mathbf{c}_i whose values are given by h_i , i.e.,

$$h_i = f(\mathbf{c}_i) = P(\mathbf{c}_i) + \sum_{j=1}^k w_j \phi(\mathbf{c}_i - \mathbf{c}_j). \quad (5)$$

If a constraint \mathbf{c}_i is located on the surface of the object, h_i equals 0, which is called a *surface* or *boundary constraint*. This results in the following linear system, with $\phi_{ij} = \phi(\mathbf{c}_i - \mathbf{c}_j)$.

$$\begin{bmatrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1k} & 1 & c_1^x & c_1^y & c_1^z \\ \phi_{21} & \phi_{22} & \cdots & \phi_{2k} & 1 & c_2^x & c_2^y & c_2^z \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \phi_{k1} & \phi_{k2} & \cdots & \phi_{kk} & 1 & c_k^x & c_k^y & c_k^z \\ 1 & 1 & \cdots & 1 & 0 & 0 & 0 & 0 \\ c_1^x & c_1^y & \cdots & c_k^x & 0 & 0 & 0 & 0 \\ c_1^y & c_1^z & \cdots & c_k^y & 0 & 0 & 0 & 0 \\ c_1^z & c_1^x & \cdots & c_k^z & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_k \\ p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_k \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \quad (6)$$

Solving Eq. 6 gives us the weights w_j for $f(\mathbf{x})$. According to Turk and O'Brien, the matrix in Eq. 6 is symmetric and positive semi-definite, so it is guaranteed that the linear system always has a unique solution [11].

4.1. Contour-based segmentation

In medical imaging, 3D anatomical data is often acquired as a set of parallel *slices* using CT or MRI. An object in such 3D images can be manually segmented by drawing contours along its border in all slices. By reconstructing the surface with variational interpolation, this only has to be done on a few slices (see Fig. 1). In addition, variational interpolation allows the contours to be drawn in arbitrary views (e.g., axial, coronal, sagittal or any MPR), as shown in Fig. 2.

The points of the user-drawn contours are surface constraints \mathbf{c}_i^S . But for unambiguously defining the implicit function, we need additional constraints that define which points should be located inside or outside the object. Turk and O'Brien distinguish between *interior*, *exterior* and *normal constraints* [7]. We use the latter ones as they allow defining the normal in each surface constraint.

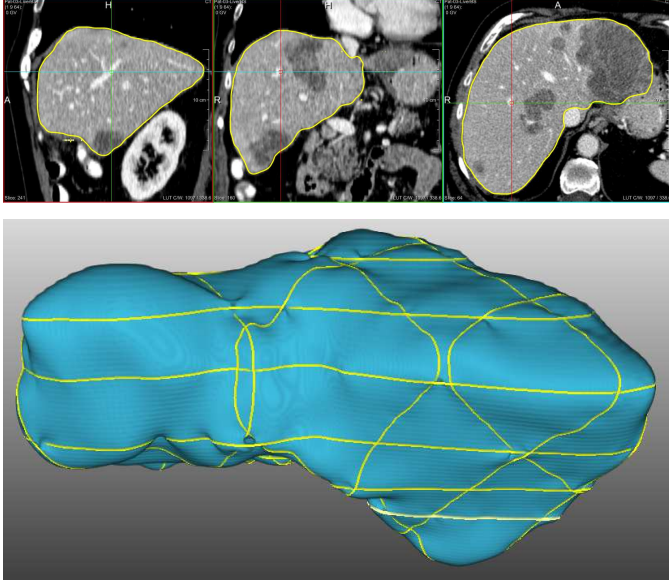


Figure 2: Example for segmentation of a liver in CT data using 14 contours in sagittal, coronal and axial view.

The location of a normal constraint \mathbf{c}_i^N is computed by adding the normal \mathbf{n}_i to the corresponding surface constraint \mathbf{c}_i^S , which is done for all surface constraints (i.e., for all points of all contours) (see Fig. 3). If a normal constraint is located inside the object, h_i^N is set to a positive value (typically 1). If it is located outside the object, h_i^N is set to a negative value (typically -1). The normal \mathbf{n}_i in a point \mathbf{c}_i^S on a contour can easily be estimated using the adjacent points and the normal \mathbf{n}^C of the plane in which the contour is defined:

$$\mathbf{n}_i = \frac{(\mathbf{n}^C \times (\mathbf{c}_{i+1}^S - \mathbf{c}_i^S)) + (\mathbf{n}^C \times (\mathbf{c}_i^S - \mathbf{c}_{i-1}^S))}{\|(\mathbf{n}^C \times (\mathbf{c}_{i+1}^S - \mathbf{c}_i^S)) + (\mathbf{n}^C \times (\mathbf{c}_i^S - \mathbf{c}_{i-1}^S))\|}. \quad (7)$$

For deciding whether \mathbf{c}_i^N is located inside or outside the contour, we have to perform a *point-in-polygon test*. In our implementation we use a simple *crossing test* [34]. To speed up this test, all edges of the polygon are inserted into bins where each bin corresponds to a voxel row in the image. This way only the edges within the bin in which the point is located have to be considered for the crossing test.

The final step of the segmentation process is a *voxelization* of the resulting implicit function. A naive solution to this is an evaluation of Eq. 2 for each voxel. Because of the discretization, no voxel will lie exactly on the surface, i.e., $f(\mathbf{x}^V)$ will not be 0 with \mathbf{x}^V being the position of the voxel's center. Instead, we have to evaluate Eq. 2 for each of the eight corner points $f(\mathbf{x}_i^V)$ of the voxel. A voxel is located on the surface if there is at least one corner point that is inside ($f(\mathbf{x}_i^V) > 0$) and at least one corner point that is outside the object ($f(\mathbf{x}_i^V) < 0$).

Variational interpolation does not use image information for segmentation but only considers the object geometry given by the user-drawn contours. A segmentation algorithm that uses variational interpolation is therefore able to properly segment objects also if there is no contrast to the background, i.e., if the

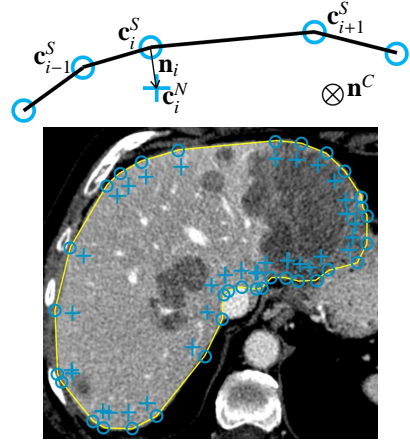


Figure 3: Top: Computation of a normal constraint \mathbf{c}_i^N (indicated by a blue +) for a surface constraint \mathbf{c}_i^S (indicated by a blue circle). The normal \mathbf{n}^C of the plane in which contour is defined is pointing inwards. Bottom: Normal constraints for one contour of the liver example.

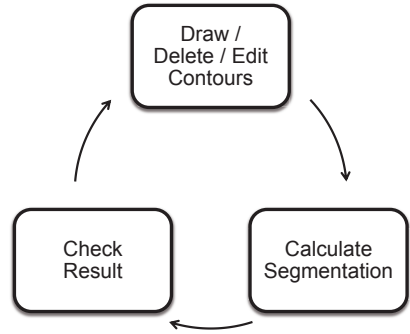


Figure 4: Contour-based interactive segmentation process with variational interpolation. Contours can be drawn manually or by more advanced image-based algorithm like live-wire or SketchSnakes for example.

object does not exhibit distinct boundaries. This is a common problem for many objects in medical images, at least at certain parts of the object, which is one reason why automatic algorithms frequently fail. Regarding the smoothness of the reconstructed surface it should be emphasized that objects in medical images are rather smooth as biological structures generally do not have sharp edges.

4.2. Interactive segmentation

In an interactive segmentation process, the user can simply add, remove or edit contours using proper contouring tools if the segmentation is not yet sufficient. The segmentation is automatically calculated after each modification to the contours as shown in Fig. 4. Contouring can be done manually or by more advanced image-based algorithms like live-wire or SketchSnakes for example [4, 6]. This way, an interactive contour-based algorithm would be used for segmenting the object in 2D and these contours could then be used as input to our algorithm to generate a 3D segmentation. In our implementation, contours are drawn manually in 2D viewers that show the image data in different orientations in a slice-wise manner and in which the segmentation result is visualized as a semi-transparent overlay (see Sec. 8 for an example). A computer

mouse and a touch-screen display have been employed as input devices.

Because it is a global interpolation algorithm, variational interpolation cannot take advantage of local changes, i.e., the whole object needs to be reconstructed after each user interaction. Therefore, the segmentation algorithm needs to be fast enough to allow a real-time response for this use case. We will address the performance issues in the next section.

5. Towards real-time response

A segmentation algorithm that uses variational interpolation consists of two computationally intensive steps. The first step is building and solving the linear system given in Eq. 6. The second step is the voxelization of the resulting implicit function.

5.1. Building and solving the linear system

Building the matrix given in Eq. 6 takes n^2 evaluations of the RBF given in Eq. 4, where n is the size of the matrix that is given by the total number of constraints + 4 (because of the linear and constant portions). This operation can easily and efficiently be performed in parallel using multiple threads. We use *OpenMP*¹ for parallelization.

Turk and O’Brien use an LU decomposition to solve Eq. 6, which needs about $\frac{2}{3}n^3$ operations. Fortunately, because the matrix in Eq. 6 is symmetric, a more efficient algorithm developed by Bunch and Kaufman can be used, which only takes about $\frac{1}{3}n^3$ operations [35]. The *Bunch-Kaufman algorithm* has a complexity comparable to a Cholesky decomposition, but it is applicable to all symmetric matrices. The algorithm is available as part of the LAPACK (Linear Algebra PACKage) library, which we use for solving the linear system.

There are different LAPACK implementations available. The basic implementation *CLAPACK*² is a rather slow implementation, because it does not take advantage of modern CPU features, such as multiple cores and 64-bit as well as vector instruction sets like SSE (Streaming SIMD Extensions). Optimized implementations that are much faster compared to CLAPACK are available in the *Intel Math Kernel Library (MKL)*³, the *AMD Core Math Library (ACML)*⁴ and the *Accelerate Framework*⁵ (only available on Mac OS X since 10.3). We have compared the 64-bit versions of CLAPACK 3.0, ACML 4.3.0 and MKL 10.2.2 using the Bunch-Kaufmann based solver for the linear system. The function that implements this algorithm in double precision is called *dsysv* in LAPACK. A detailed overview on the results is given in Sec. 7.

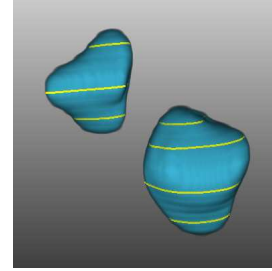


Figure 5: Example where two tumors have been segmented. This shows that it is necessary to start the surface tracking for at least one point of each contour to cover all objects.

5.2. Voxelization

A naive voxelization of the implicit function $f(\mathbf{x})$ has to evaluate Eq. 2 for each corner point of each voxel. Let n be the total number of constraints plus the linear and constant portions (i.e., the size of the matrix in Eq. 6) and let m be the total number of voxels of the resulting segmentation (the resolution of the segmentation can differ from the resolution in which the user has drawn the contours). The number of operations necessary for a voxelization is $O(nm)$ for the naive algorithm. This can be optimized by calculating each corner value only once and not for each voxel and by restricting the voxelization to a specific axis-aligned bounding box around the contours. Unfortunately, the real bounding box is only given by the implicit function, so it is not known until all voxels have been evaluated.

We use a voxelization method based on *surface tracking* that generates the surface of the object given by $f(\mathbf{x})$ with only $O(n\tilde{m})$ operations, where \tilde{m} is the number of voxels on the surface of the object in the final segmentation. The tracking is based on a scheme similar to the marching cubes algorithm [36]. A similar algorithm has also been used by Wyvill *et al.* for representation and visualization of soft objects [37]. In real world examples $\tilde{m} \ll m$ holds. Therefore, this method is much faster compared to the brute-force approach. For example, only about 2.22% (metastasis) and 1.58% of the voxels (liver) are located on the surfaces of the examples used in this paper (see Tab. 4). Another advantage is, that the voxelization time only depends on the number of surface voxels and thus, it is independent of the actual size of the image. For voxelization a starting point is needed that is known to be located on the surface of the object. This is true for each surface constraint. As a consequence, the voxel in which a contour’s first surface constraint is located is used. If the starting point is not exactly located on the surface, e.g., due to discretization or numerical issues, its neighboring voxels, i.e., all voxels in the 26-neighborhood, are tested as well. Because the contours might define several objects, we need to use one starting point for each contour (see Fig. 5). If a starting point is already part of a voxelized surface, it is rejected. Otherwise, the voxelization of the already processed surface would be repeated. We use the 6-neighborhood of each voxel for tracking the surface.

Computation times can be reduced by using multiple threads. In our implementation, $f(\mathbf{x})$ is evaluated in parallel for each of the eight corners of a voxel when scanning the surface. Because

¹OpenMP: <http://openmp.org>

²CLAPACK: <http://www.netlib.org/clapack>

³MKL: <http://software.intel.com/en-us/intel-mkl>

⁴ACML: <http://developer.amd.com/cpu/Libraries/acml>

⁵Accelerate: <http://developer.apple.com/performance/accelerateframework.html>

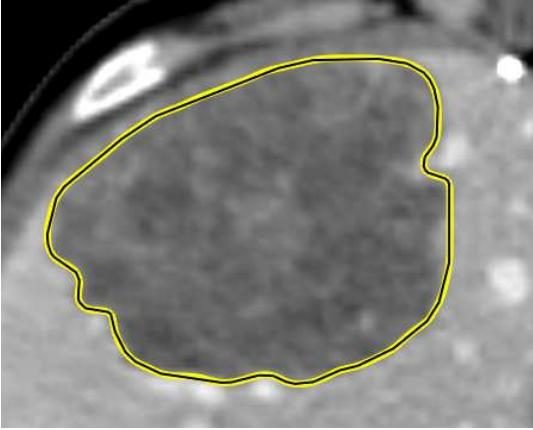


Figure 6: Example of a contour before (thick yellow polygon) and after reduction (thin black polygon) with a quality factor of $q = 0.2$. The initial contour consists of 256 points, the reduced one has 50 points.

two voxels share four corner points, these results are cached to prevent redundant computations. Therefore, four or six of the corner values have already been evaluated when moving to the next voxel and only two or four values have to be evaluated. Because of this, the theoretical maximum speedup of our parallelization strategy for the voxelization is between two and four. As for building the linear system, we have used OpenMP for the parallelization of the algorithm.

The surface of the segmented object (or objects) is filled in a successive step using a *scan-line algorithm* in x-direction for each row (y-coordinate) of each slice (z-coordinate). Because start and end voxels for the scan-line are known by their configuration (i.e., the values of the implicit function in each corner of the voxel), $f(\mathbf{x})$ does not have to be evaluated when filling the object.

5.3. Constraint reduction

For a segmentation algorithm that is based on variational interpolation, the greatest impact on the overall computation time of is given by the number of constraints n . The computational complexity for finding the solution of the linear system has a complexity of $O(n^3)$, while the voxelization is $O(n\tilde{m})$. Therefore, using only half of the constraints (i.e., contour points) decreases the calculation time for solving the linear system by a factor of 8. Also the voxelization would only take half as much time. Hence, a reduction of the number of contour points significantly decreases the overall calculation time.

In the context of object reconstruction from unorganized point clouds, this step is typically referred to as *center reduction*. Because there is no additional geometry information available in this case, Carr *et al.* suggest using a greedy algorithm to iteratively fit an RBF to a subset from the given points within a desired fitting accuracy [13]. Fortunately, we have additional geometry information, because in our case the point clouds are generated from contours.

To achieve real-time calculation times, we remove contour points \mathbf{c}_i that have no or almost no influence on the contours'

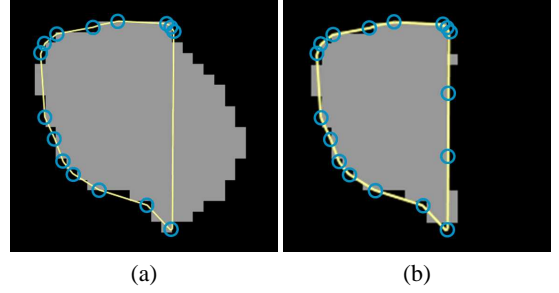


Figure 7: Example for variational interpolation of a contour after reducing the number of constraints: (a) before and (b) after inserting additional surface constraints (blue circles).

geometry which reduces the number of constraints while preserving the overall shape of the contour, especially on parts with high curvature (see Fig. 6). This is carried out in a pre-processing step. As described by Latecki and Lakämper we use two attributes to measure the influence of a contour point on the overall geometry [38]: The angle between its adjacent edges ω_i^α and the length of the adjacent edges ω_i^l . These weights are given by:

$$\omega_i^\alpha = \frac{1}{2} \left(1 - \frac{(\mathbf{c}_i - \mathbf{c}_{i-1}) \cdot (\mathbf{c}_{i+1} - \mathbf{c}_i)}{\|\mathbf{c}_i - \mathbf{c}_{i-1}\| \|\mathbf{c}_{i+1} - \mathbf{c}_i\|} \right) \quad (8)$$

$$\omega_i^l = \frac{\|\mathbf{c}_i - \mathbf{c}_{i-1}\| \|\mathbf{c}_{i+1} - \mathbf{c}_i\|}{\|\mathbf{c}_i - \mathbf{c}_{i-1}\| + \|\mathbf{c}_{i+1} - \mathbf{c}_i\|}. \quad (9)$$

In contrast to Latecki and Lakämper, we use the angle between the edges given by \mathbf{c}_{i-1} , \mathbf{c}_i and \mathbf{c}_{i+1} for computation of ω_i^α , which is $\pi - \beta$ when looking at the relevance measure used by the authors. Moreover, we shift ω_i^α to $[0, 1]$ such that collinear edges get a weight of 0. The total weight ω_i of a contour point is calculated by:

$$\omega_i = \omega_i^\alpha \omega_i^l. \quad (10)$$

This way, high weights are assigned to contour points with a high curvature and long adjacent edges, while low weights are assigned to points with almost no curvature and short edges.

Our reduction algorithm iteratively removes the points with the lowest weight ω_i until a specific number of points is reached. This number is given by a quality factor $q \in (0, 1]$ multiplied by the initial number of points. I.e., using a quality of $q = 0.5$ removes about half of the points, while $q = 0.2$ removes about 80% of the points. Because we do not want to remove a contour completely, the reduction stops if less than 6 points are left.

During this constraint reduction step, collinear points are replaced by one single line segment. Such contours are visually equal to the original contours. However, the surface defined by the reduced contours might be different from the original surface, because of the loss of information, which can be seen in Fig. 7a. To solve this, we insert additional surface constraints on long line segments if the length of the segment is twice as long as the average length of all segments after the reduction step. The constraints are inserted such that the length of the resulting segments equals the average length (see Fig. 7b). Instead of this postprocessing step, long line segments could al-

	Global RBFs	Compactly supported RBFs
Storage	$O(n^2)$	$O(n)$
Linear system building	$O(n^2)$	$O(n \log n)$
Linear system solving	$O(n^3)$	$O(n^{1.5})$
Evaluation per point	$O(n)$	$O(\log n)$
Effect of a single point	Global	Local

Table 2: Complexity of different steps of the algorithm when using global RBFs and RBFs with compact support according to Morse *et al.* [19].

ready be avoided in the reduction step using an adjusted weight ω_i for each surface constraint.

5.4. Radial basis functions with compact support

Another approach for fast creation of an implicit surface of a complex model from a point cloud are *compactly supported RBFs* as described by Morse *et al.* [19]. These RBFs allow a local interpolation instead of a global one. In 3D a C^2 -continuous RBF with compact support is defined by

$$\phi_{\mathbf{c}}(r) = \begin{cases} (1-r)^4(4r+1) & , \text{if } r < 1 \\ 0 & , \text{otherwise,} \end{cases} \quad (11)$$

with $r = \|\mathbf{x} - \mathbf{c}\|$ being the distance of the evaluated point \mathbf{x} to the RBF's center \mathbf{c} . The support radius of this RBF equals 1. Scaling this function, i.e., by calculating $\phi_{\mathbf{c}}(\frac{r}{\alpha})$, allows any support radius α .

Compactly supported RBFs result in a linear system with a sparse matrix, which reduces both the memory and the computational complexity for solving the linear system, as long as the radius of support is small enough. Because the matrix is sparse, iterative methods can efficiently be used for solving the system of equations. Table 2 gives an overview on the complexity for different steps of the algorithm compared to global RBFs as used in the variational interpolation algorithm. For efficiently building the linear system and for an efficient evaluation of the implicit function, a fast algorithm for finding all points within a given sphere is essential. In computational geometry this is known as *range searching*. A range searching can efficiently be performed by using a spatial subdivision data structure [39]. Although it is not the most efficient data structure in theory, a commonly used spatial subdivision algorithm is the *kD-tree*. Other common spatial subdivision data structures are *bins*, *octrees*, *bounding-volume-hierarchies* and *R-trees*.

For fast range searching we use a kD-Tree as well. Because the voxelization is the bottleneck of our algorithm, we have focused our performance measurements on this step (see. Sec. 7).

6. Making the segmentation more robust

Self-intersecting contours cannot always be handled correctly by the basic algorithm. Moreover, an interpolation of the user-drawn contours is not always desired, because of inaccuracies that might occur, especially if the user is allowed to draw in different views.

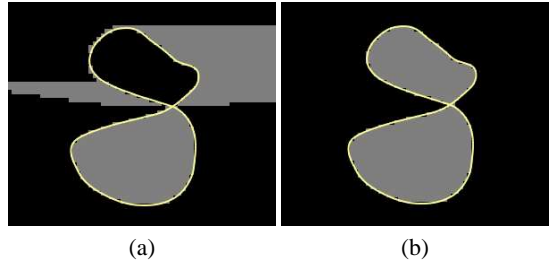


Figure 8: Example for variational interpolation of a self-intersecting contour: (a) before and (b) after normal correction.

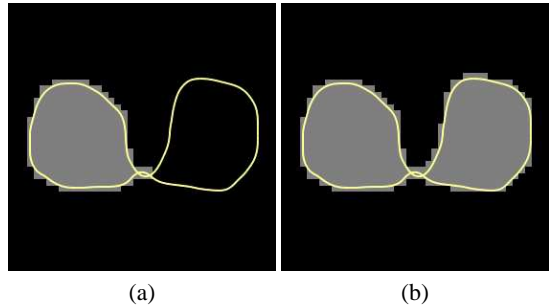


Figure 9: Example for a more complex self-intersecting contour: (a) with only the first point and the intersection points as starting points for surface tracking and (b) with 10 equally distributed points of the contour as additional starting points.

6.1. Handling self-intersecting contours

Normal constraints allow a better segmentation result, because the surface interpolates the contours more accurately (recall Sec. 4.1). The normal can be calculated according to Eq. 7, followed by a point-in-polygon test. However, for self-intersecting contours this test might not detect the intersection properly, resulting in an incorrect surface as shown in Fig. 8a. As a solution, we calculate the normal constraint in a more robust way.

By default we assume that the normal constraint is located outside the contour. To ensure this, we start the computation of the normal using three points of the contour that are known to be convex. This is true for the point with minimum x-, y- or z-coordinate (depending on the plane in which the contour is defined) and its adjacent points. Based on these minimum points we calculate the normal according to Eq. 7. If this normal points inside the contour, we invert the sign of the normal. Because a contour is an ordered set of points, we can now iteratively scan it, starting from its minimum point and calculate the normal in each point while setting the sign according to the sign of the normal in the minimum point. This way all normals point outside, until two adjacent points are involved in a self-intersection, i.e., until the line segment defined by these points intersects one or more other line segments of the contour. If the number of intersections with other line segments is odd, we need to invert the sign of this normal and all following normals, until the next self-intersection line segment is found. This way we don't have to split self-intersecting polygons, which is for example necessary for active contour models or when reconstructing the object in the surface-mesh domain [40, 5].

In addition to this normal calculation, we also need additional starting points for the voxelization of the surface, because the surfaces surrounded by such contours are not necessarily coherent. We use the voxels of 10 points from the self-intersecting contour as starting points. These points are equally distributed over the contour. In addition, we use the voxels in which the intersection points are located as additional starting points, because these points are known to be located on a new surface, apart from discretization and rounding errors. This way we get a consistent surface even for self-intersecting contours, as shown in Fig. 8b. Because of the additional equally distributed starting points, even complex self-intersections are handled properly (see Fig. 9). However, complex self-intersections are rather unusual in practice when segmenting biological objects in medical image data but they can occur due to an inaccurate drawing of contours along thin structures for example.

6.2. From interpolation to approximation

As the name indicates, variational interpolation is an *interpolating* algorithm, i.e., all surface and normal constraints are included in the implicit function as given in the linear system (see Eq. 6). Typically, this is intended because the user-drawn contours shall be exactly included in the segmentation by the algorithm. But in some cases, for example if the user draws contradictory contours as shown in Fig. 15b, the interpolated surfaces become degenerated in the sense that it is not what the user expects and that the surface has a high curvature in some points. This can be addressed by moving from an interpolating to an *approximating* character, which is also known as *regularization*.

As already discussed by some authors, an approximation can be achieved by replacing the elements ϕ_{ii} in Eq. 6 by $\phi_{ii} + \lambda_i$, with $\lambda_i > 0$ being the *regularization parameter* or *regularization strength* for each constraint [14, 16]:

$$\begin{bmatrix} \phi_{11} + \lambda_1 & \phi_{12} & \cdots & \phi_{1k} & 1 & c_1^x & c_1^y & c_1^z & w_1 \\ \phi_{21} & \phi_{22} + \lambda_2 & \cdots & \phi_{2k} & 1 & c_2^x & c_2^y & c_2^z & w_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \phi_{k1} & \phi_{k2} & \cdots & \phi_{kk} + \lambda_k & 1 & c_k^x & c_k^y & c_k^z & w_k \\ 1 & 1 & \cdots & 1 & 0 & 0 & 0 & 0 & p_0 \\ c_1^x & c_2^x & \cdots & c_k^x & 0 & 0 & 0 & 0 & p_1 \\ c_1^y & c_2^y & \cdots & c_k^y & 0 & 0 & 0 & 0 & p_2 \\ c_1^z & c_2^z & \cdots & c_k^z & 0 & 0 & 0 & 0 & p_3 \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_k \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \quad (12)$$

Because we only add values ≥ 0 to the principal diagonal of the matrix from Eq. 6, this matrix is still symmetric and positive semi-definite. The larger λ_i is, the more the algorithm approximates the i -th constraint. The size and the unit of the regularization parameter correspond to the coordinates of the constraints. In our case, these are given in millimeters. Best results can be achieved if different regularization strengths are used for surface and normal constraints. We found values of $\lambda_i^S = 2$ for surface and $\lambda_i^N = 20$ for normal constraints a good compromise between segmentation accuracy and regularization strength in our examples.

Example	t	Build	CLAPACK	ACML	MKL
Metastasis $n = 6246$	1	0.61s	52.51s	12.04s	9.92s
	2	0.32s	-	7.37s	5.69s
	4	0.19s	-	5.04s	3.55s
	8	0.14s	-	5.58s	2.91s
Liver $n = 13798$	1	3.16s	561.73s	117.32s	103.13s
	2	1.73s	-	68.26s	55.68s
	4	0.96s	-	44.84s	31.59s
	8	0.72s	-	41.18s	23.38s

Table 3: Computation times for building and solving the linear system described in Eq. 6 using CLAPACK, the AMD Core Math Library (ACML) and the Intel Math Kernel Library (MKL) with a different number of threads (t). The metastasis example is shown in Fig. 1, the liver example is shown in Fig. 2. n is the size of the matrix (surface constraints + normal constraints + 4).

Example	t	Voxelization
Metastasis $n = 6246$ $m = 1058508$ $\tilde{m} = 23541$	1	4.41s
	2	2.48s
	4	2.19s
Liver $n = 13798$ $m = 10711151$ $\tilde{m} = 169296$	1	64.18s
	2	35.49s
	4	29.10s
	8	34.74s

Table 4: Computation times for voxelization of the implicit function using the surface tracking approach and different numbers of threads (t). n is the size of the matrix. m is the number of voxels of the corresponding dataset, while \tilde{m} is the number of voxels on the surface of the object.

The surface constraints are not necessarily part of the implicit surface anymore because they are approximated instead of interpolated. This can be seen in Fig. 13, where the surface does not always contain the contours. As a consequence, the starting point for surface tracking might not be on the surface as it has been defined in Sec. 5.2. In this case we need to look in the 26-neighborhood of the voxel associated with the starting point for a voxel on the surface of the implicit function and use it as starting point for surface tracking.

7. Results

The results in Tab. 3 show that using an optimized LAPACK implementation has a significant impact on the time necessary for solving the linear system. All measurements were performed on an 8-core system (2x Intel Xeon X5550 (2.66GHz), Turbo Boost and Hyper-Threading disabled, 12 GB RAM, Windows 7 64-bit). Intel's MKL has shown to be the fastest library for our problem, at least on an Intel CPU. The ACML is much faster than CLAPACK as well and it might be even faster on AMD CPUs.

Measurements of the voxelization times are given in Tab. 4. The measurements only include the voxelization of the surface. The time for filling the object is negligible ($\ll 1$ s). As expected, our parallelization strategy for the voxelization has its maximum speedup when using four threads. The maximum speedup is about 2.21. Using more threads slows down the computation,

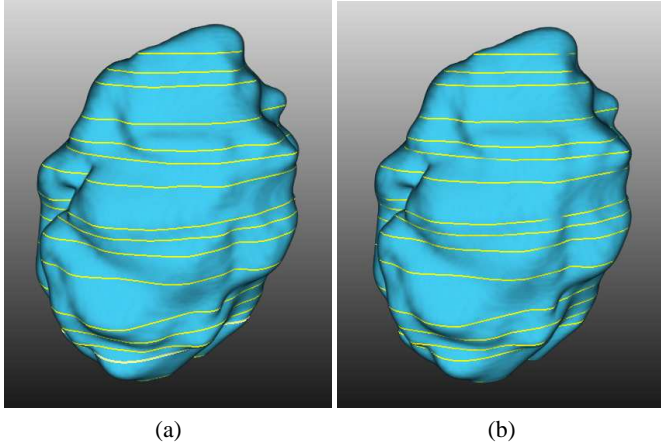


Figure 10: Example for reducing the number of constraints: (a) shows the original contours and the result of the interpolation using all 6246 constraints, (b) shows the result after reduction with $q = 0.2$ resulting in 1390 constraints. The resulting masks are visually identical.

because the overhead for handling the threads is larger than the speedup of the parallel computation as typically only two or four threads are involved in computations. This is due to the fact that the function values at four or six corners of the voxel have already been computed during the surface tracking.

However, the results in Tab. 3 and 4 show, that the algorithm does not allow real-time response in an interactive segmentation process. Especially in the liver example the segmentation algorithm is far too slow even with the fastest LAPACK library and with parallel voxelization.

Using a proper reduction of the number of constraints yields a significant speedup in both solving the linear system and the voxelization, while resulting in an almost similar overall quality of the segmentation as shown in Tab. 5 and Fig. 10. Calculation times for this preprocessing are negligible ($\ll 1s$) compared to the overall computations and are not included in Tab. 5. The maximum distance between the surface of initial segmentation (i.e., with $q = 1.0$) and the surface with $q = 0.1$ corresponds to 2.6% of the maximum diameter for the metastasis example (which has a maximum diameter of 100.8mm) and to 3.3% for the liver example (which has a maximum diameter of 228.643mm). The size of one voxel is $(1.086mm, 1.086mm, 1.0mm)$ in the metastasis example and $(0.66mm, 0.66mm, 2.0mm)$ in the liver example, so the maximum distance in Tab. 5 corresponds to about 2.45 voxels for the metastasis and about 11.31 voxels for the liver. The large maximum distance in the liver example results from an incorrect reconstruction due to the use of orthogonal contours (see Fig. 11). Therefore, the bad results for the maximum distance do not reflect a loss in segmentation quality.

Our measurements show that a real-time response can be achieved for small objects, which allows an interactive segmentation. It has also shown that in our current implementation the voxelization is the main limiting part of the segmentation algorithm, which is the reason why the algorithm is not interactive for large objects. Solving the linear system is already fast enough, although it has a computational complexity of $O(n^3)$.

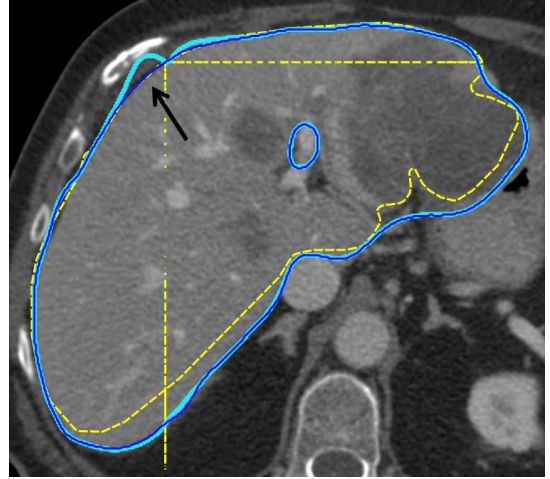


Figure 11: Differences in the segmentation result for the liver example when using $q = 1$ (thick light blue contour) and $q = 0.1$ (thin dark blue contour). Here the maximum distance between the surfaces is large (about 11 voxels) because of a reconstruction issue due to using orthogonal contours (dashed yellow). The result using less contour points (i.e., with $q = 0.1$) is the more accurate segmentation.

Example	Global RBFs	Radius	Compactly supported RBFs
Metastasis $n = 6246$	1.92s	10mm	0.73s
		20mm	1.40s
		30mm	2.06s
Liver $n = 13798$	28.95s	30mm	12.45s
		50mm	20.58s
		70mm	34.82s

Table 6: Voxelization times with 4 threads using compactly supported RBFs with different radii compared to global RBFs for $q = 1$. A kD-tree was used for finding all points within the radius of support.

The strength of reduction, i.e., the value of q , depends on the actual segmentation task and on the demand for real-time response during the segmentation process. It is a trade-off between reconstruction accuracy and overall calculation time. In our tests we found a quality of $0.2 \leq q \leq 0.5$ can be used in most real-world examples without a loss in *subjective* segmentation accuracy.

Results using compactly supported RBFs are shown in Fig. 12 and Tab. 6. These results suggest that this class of RBFs is not suitable for solving our segmentation task. The main reason for this is, that the constraints generated from the user-drawn contours are rather sparse and thus, a large radius of support is necessary to generate proper results, which reduces the theoretical performance of the algorithm significantly in terms memory consumption, calculation times and segmentation quality. Moreover, the objects are not filled correctly if the radius is too small, because $f(\mathbf{x})$ might evaluate to zero for points inside the object if their distance to the surface is greater than the radius of support. Therefore, a strategy for an automatic calculation of the support radius depending on the size of the object and the density of the constraints is necessary. However, when using a proper radius, the voxelization step is even

Example	q	n	Time	Volume overlap	Maximum surface distance	Voxel with surface distance > 0
Metastasis	1.0	6246	3.05 /2.19 /5.34s	100%	0.0mm	0
	0.5	3552	0.71 /1.25 /2.02s	99.94%	1.09mm	98
	0.2	1390	0.08 /0.57 /0.69s	99.32%	1.09mm	1030
	0.1	708	0.02 /0.39 /0.43s	97.35%	2.63mm	4038
Liver	1.0	13798	24.10/29.10/53.64s	100%	0.0mm	0
	0.5	7892	5.31 /18.69/24.18s	99.50%	2.75mm	8916
	0.2	3152	0.55 /7.66 /8.33s	98.72%	7.08mm	22770
	0.1	1580	0.11 /4.49 /4.71s	97.70%	7.54mm	41012

Table 5: Computation times (linear system building and solving / voxelization / overall) and segmentation accuracy for different quality settings q . n is the size of the resulting matrix. For all measurements the MKL with 8 threads and the voxelization with 4 threads were used. Volume overlap (*Jaccard index*) and surface distance are given relative to the interpolation result with $q = 1$. The overlap is calculated by the number of intersecting voxels divided by the number of voxels in the union of two segmentations. The minimum surface distance was 0mm for all cases.

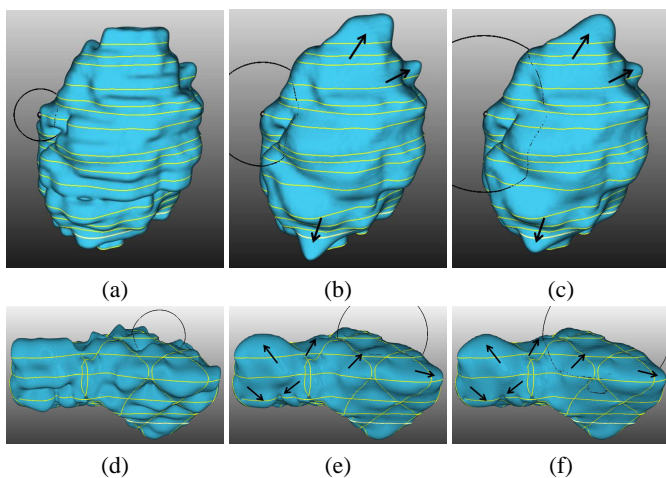


Figure 12: Results using RBFs with compact support of different radii (indicated by the black circle) for $q = 1$: (a) Metastasis with $\alpha = 10$ mm, (b) $\alpha = 20$ mm and (c) $\alpha = 30$ mm. (d) Liver with $\alpha = 30$ mm, (e) $\alpha = 50$ mm and (f) $\alpha = 70$ mm. Using a smaller radius, the surface becomes more “bumpy”. Only (c) and (f) have been filled correctly by the voxelization. The arrows indicate some main differences of the resulting segmentations.

slower for our segmentation problems when using RBFs with compact support compared to the global interpolation approach.

As shown in the examples in Fig. 13, the robustness of the segmentation algorithm can further be improved by moving from an interpolating to an approximating reconstruction scheme. By using approximation instead of interpolation, contradictory user-drawn contours and contours with high curvature can be segmented more robustly for instance. Therefore, approximation generates better segmentations than interpolation in our opinion. Because we only add constants to the elements on the principal diagonal of the matrix in Eq. 12, this approach does not affect the time necessary for solving the linear system. Because with approximation a little less voxels are on the surface of the segmentation, the voxelization time is slightly reduced in practice.

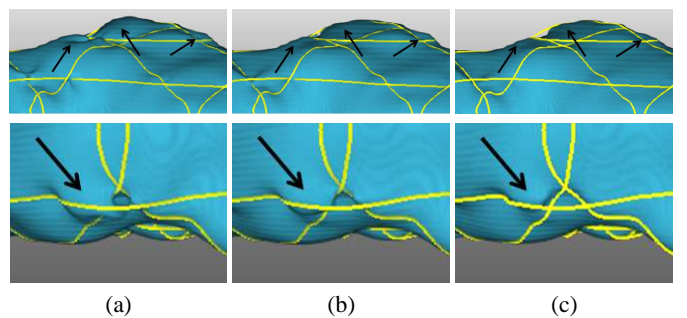


Figure 13: Comparison between interpolation and approximation with different regularization strengths for the liver example: (a) interpolation, (b) approximation with $\lambda_i^S = 2$ and $\lambda_i^N = 20$ and (c) approximation with $\lambda_i^S = 50$ and $\lambda_i^N = 100$. The arrows indicate the main differences of the resulting segmentations.

8. Evaluation

For evaluation of our algorithm, we have developed a tool using MeVisLab⁶ that allows a slice-wise manual segmentation in axial view, a segmentation using variational interpolation with only parallel axial contours and a segmentation using variational interpolation with contours in axial, sagittal and coronal orientation. Figure 14 shows this tool. Our evaluation tool measures the time that is necessary for segmentation and it tracks the number of contours that are needed. Moreover, it allows to rate the segmentation quality and the segmentation time on a scale between 1 and 5, with 1 being “bad” and 5 being “perfect”.

Our algorithm was evaluated by two experienced radiology technicians on 15 liver metastases of different size (ranging from 0.05ml to 53.26ml) and complexity and 1 liver. The PC used for evaluation was a Windows 7 64-bit system with an Intel Core 2 Duo E6850 (3.0GHz) and 8GB RAM. Thus, only two threads were used for computation. For contouring, a touch-screen display with a pencil-like input device was used. The quality factor for constraint reduction was set to $q = 0.2$, while the regularization strengths were set to $\lambda_i^S = 2$ and $\lambda_i^N = 20$.

⁶MeVisLab: <http://www.mevislab.de>

	Manual (axial)	Variational interpolation (axial)	Variational interpolation (axial, sagittal, coronal)
Metastases	Time	111.4s	70.5s
	Number of contours	20.6	5.2
	Number of steps	29.3	5.4
	Quality rating	4.9	4.0
	Time rating	3.8	4.1
	Overlap to manual segmentation	100%	71.54%
	Maximum surface distance	0	3.69mm
	Number of voxels with distance > 0	0	3340
Liver	Time	1272.5s	734.5s
	Number of contours	106	20.5
	Number of steps	148.5	25
	Quality rating	5	4.5
	Time rating	2.5	4.5
	Overlap to manual segmentation	100%	92.68%
	Maximum surface distance	0	10.12mm
	Number of voxels with distance > 0	0	128250

Table 7: Results of the evaluation of our algorithm for segmentation of 15 liver metastases and 1 liver by 2 experienced radiology technicians. All results are given as average over all cases and participants. The overlap and distance measures are given relative to the manual segmentation.

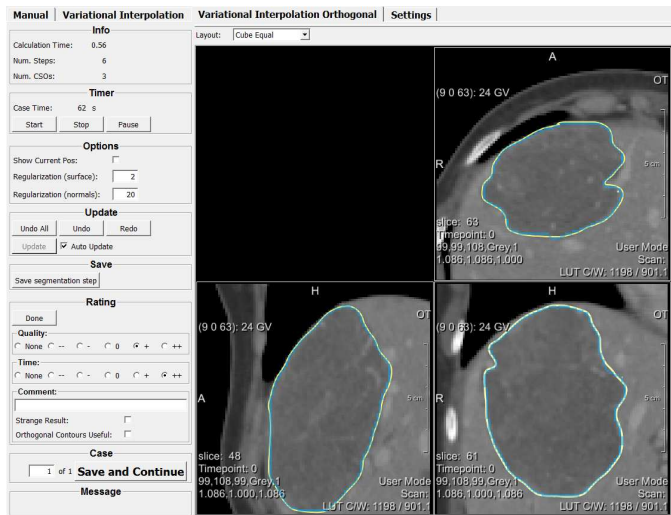


Figure 14: The tool used for evaluation of our algorithm. The user-drawn contours are shown in yellow. The surface of the interpolated segmentation is shown in blue.

Table 7 gives an overview on the results of our evaluation. A step refers to drawing, deleting or editing a contour.

Our evaluation shows that using variational interpolation, the segmentation times can be reduced by a factor of about 2. Compared to the number of contours used for the manual segmentations, only one fifth to one third of the contours are necessary when using our interpolation algorithm, depending on the segmentation task. The fact that the segmentation takes only half as long although about one fourth of the contours are necessary can be explained by the time the variational interpolation needs for computation and by the interactive process. This process requires an assessment of the current segmentation result and the user might need to correct or even delete already drawn contours. The assessment is known to take almost as much time

as manually drawing contours. Unfortunately, the segmentation quality has been rated slightly worse compared manual segmentations. This also corresponds with the accuracy of our segmentation algorithm in terms of the volume overlap and the maximum surface distance compared to the manual segmentations. By using orthogonal contours instead of only parallel axial contours, the number of contours necessary for segmentation can further be reduced, although the overall segmentation time increases, probably due to the more time intensive task of looking at different views. The benefit of using contours in different views was emphasized by the radiology technicians in particular for sphere-shaped structures. For objects with a more complex and irregular shape, drawing contours in different views was assessed as rather counterproductive.

Although we have not yet compared our algorithm to more advanced contour-based semi-automatic segmentation algorithms like live-wire, we expect a speedup in the segmentation time that is comparable to the speedup measured for pure manual segmentations, because variational interpolation can be combined with such approaches as well. However, it would be interesting to compare variational interpolation with the combination of live-wire and a shape-based interpolation as suggested by Schenk *et al.* [31].

9. Discussion and future work

As shown in the examples, variational interpolation allows a smooth segmentation of objects in medical images from a sparse set of contours (see Fig. 1 and Fig. 2). The algorithm is computationally expensive and needs much memory ($O(n^2)$), because it is a global interpolation. Therefore, the number of constraints is limited by the available memory. Moreover, computation times are slow for a large number of constraints. This can be solved by reducing the constraints resulting in computation times that allow a real-time response for segmentation of

	Direct	Fast multipole
Storage	$O(n^2)$	$O(n)$
Setup	-	$O(n \log n)$
Linear system solving	$O(n^3)$	$O(n \log n)$
Evaluation per point	$O(n)$	$O(1)$

Table 8: Complexity of different steps of the algorithm when using the direct method and the fast multipole method according to Carr *et al.* [13].

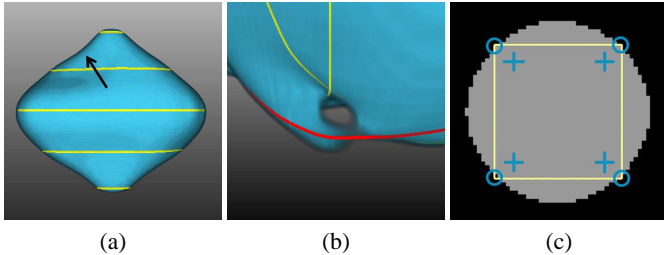


Figure 15: Limitations of our algorithm: (a) 2D normal constraints using interpolation (which can be seen at the uneven silhouette), (b) contradictory contour (red) in the liver example and (c) a square-shaped contour being interpolated to a sphere.

small objects like a tumor in an interactive segmentation process. However, in our current implementation and for large objects like the liver, the voxelization becomes the bottleneck after constraint reduction and the algorithm does no longer allow a real-time response.

Our goal is to allow an interactive segmentation even for large objects like the liver. Therefore, future work will focus on reducing the voxelization times by using an improved parallelization strategy. We will also evaluate, whether GPU-based implementations can improve computation times significantly. For further improving the performance and reducing the memory consumption of the algorithm we will also investigate the *fast multipole method* which has been suggested by Carr *et al.* [13]. The fast multipole method allows solving the variational interpolation problem with reduced complexity in both the required memory and the calculation time (see Tab. 8). Because this method can be seen as a global reconstruction approach it does not have the drawbacks of compactly supported RBFs. Therefore, it should be better suited for solving our 3D segmentation problem.

The presented algorithm still has some limitations concerning the segmentation quality. Normal constraints are currently computed based on a single contour, i.e., in 2D. But neighboring contours influence the normal of the surface as well (see Fig. 15a). This can be slightly compensated when using approximation instead of interpolation. Another issue are contradictory contours, i.e., contours that define different surfaces, although they should be located on the same surface. This can in particular happen for non-parallel contours like in the liver example, as shown in Fig. 15b (the figure shows the same part of the liver segmentation that is shown in the bottom row of Fig. 13 but from another point of view). A similar example that results from a combination of both of these issues has already been shown in Fig. 11. Although this can be compensated by an ap-

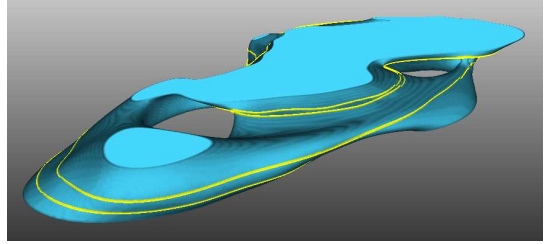


Figure 16: Erroneous reconstruction of an object with high curvature with an insufficient number of constraints because of parallel contours with low overlap and low distance. The example was taken from a 3D microscopy dataset.

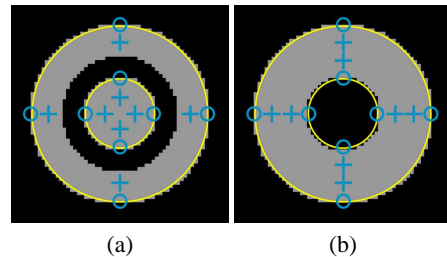


Figure 17: Example for cutting out a part of the segmentation where the algorithm fails because of its single-contour-based normal calculation: (a) result of our algorithm and (b) the expected result.

proximation as well (see Sec. 6.2), a more advanced handling, like an additional preprocessing step that solves such inconsistencies or user-guidance during contour-drawing, is essential for consistent results according to the expectations of the user. We will investigate this in the future. Furthermore, the regularization parameters of the approximation need to be further evaluated. For example, it might be advantageous to compute an individual regularization strength for each constraint based on a contours shape or complexity.

A conceptual drawback of the smooth interpolation is that the resulting surface does not always look like how the user would expect it if the number of constraints is too low. For example, a square-shaped contour that is only defined by its corner points will be interpolated to a sphere using the presented algorithm (see Fig. 15c). For objects with high curvature the reconstruction might fail, if the surface is sampled too sparse, i.e., with not enough contours. This can also be the case if parallel contours are close to each other and the overlap between the contours is low. Figure 16 shows such a situation.

Because of the way how normal constraints are computed, which is done per contour (see Sec. 4.1), it is not possible to cut out an inner part of an object as shown in Fig. 17 because the reconstruction algorithm has no information on what should be included into the segmentation and what should be excluded. Instead, this needs to be solved on the interface level, so the user has the possibility to explicitly define parts that should be removed from the segmentation.

Finally, we think of combining the reconstruction algorithm with image information to allow more accurate segmentations, e.g., by image-adaptive RBFs described by Mory and Ardon [30].

10. Conclusion

Variational interpolation allows an accurate and robust reconstruction of the surface of an object defined by a set of planar contours that can be arbitrarily oriented in 3D space. A surface generated using variational interpolation is smooth, it is guaranteed that all used contour points are part of the surface and the resulting surface is plausibly extrapolated into regions where no contours have been drawn. Thus, the algorithm is well suited for supporting the segmentation of objects in medical images in a contour-based workflow. Because it is independent of image information, the algorithm can be used for any 3D modality (e.g., CT, MRI, 3D US) and it can also be used for segmentation of objects with no contrast to the background. In addition, the interpolation can easily be extended to more dimensions, which would allow an interpolation of an object over several time points as well for example.

Although the algorithm is time and memory consuming, calculation times can be significantly decreased using modern CPU features and a reduction of the number of constraints. Because segmentation is done using contours, a fast and robust reduction strategy can be used, that preserves the shape of each contour. This way, the differences in the interpolated results are visually not distinguishable from the result using all contour points. As a result, the algorithm allows a real-time response for small objects, which enables an interactive segmentation process. Using approximation instead of interpolation improves the robustness of the segmentation algorithm in the liver example while it does not increase calculation times.

The local interpolation scheme, which uses RBFs with compact support, is not suitable for the presented segmentation problems, because the constraints generated from the user-drawn contours are too sparse and thus the radius of support needs to be too large.

Our evaluation has shown that using variational interpolation can reduce segmentation times by a factor of about 2. We assume that this factor increases as the user gets more familiar with the segmentation tool and, especially for large objects like the liver, as the algorithm itself becomes faster.

Acknowledgment

We thank Christiane Engel and Ulrike Kayser from Fraunhofer MEVIS for evaluating our algorithm. Parts of this work were funded by Siemens AG, Healthcare Sector, Imaging & Therapy Division, Computed Tomography, Forchheim, Germany.

Appendix A. Algorithm overview

Algorithm 1: Segmentation using variational interpolation

Data: Contours, quality factor, reference image
Result: Segmentation
 Perform constraint reduction;
 Perform variational interpolation;
 Perform voxelization;

Function Constraint reduction

Data: Contours, quality factor
Result: Contours
foreach *User drawn contour* C_i **do**
 foreach *Contour point* c_i **do**
 | Compute weight ω_i ;
 end
 while *#points exceeds quality factor* q **do**
 | Remove point with lowest weight;
 | Update weights of the adjacent points;
 end
 Compute average line segment length;
 foreach *Line segment* **do**
 | **if** *Length* $> 2 \times$ *average length* **then**
 | Insert additional points;
 end
 end
end

Function Variational interpolation

Data: Contours
Result: Implicit function, starting points
foreach *User drawn contour* C_i **do**
 Find point with minimum x-, y- or z-coordinate;
 Initialize sign for normal;
 foreach *Contour point* c_i^S *starting from minimum point* **do**
 | Check adjacent line segment for self-intersections;
 | **if** *Number of self-intersections is odd* **then**
 | Invert sign for normal;
 end
 Compute normal \mathbf{n}_i and normal constraint c_i^N ;
 end
end
 Build matrix for Eq. 6 in parallel;
 Solve Eq. 6 using LAPACK to get $f(\mathbf{x})$;

Function Voxelization

Data: Implicit function, starting points, reference image
Result: Segmentation
foreach *Starting point* **do**
 Evaluate $f(\mathbf{x})$ at each corner point;
 if *Voxel of starting point is not on surface of* $f(\mathbf{x})$ **then**
 | Check voxels in 26-neighborhood;
 end
 if *Starting voxel is not on any already voxelized surface* **then**
 repeat
 | Evaluate $f(\mathbf{x})$ at each corner of the voxel in parallel
 | (= configuration);
 | Move to next voxel in 6-neighborhood depending on
 | the configuration;
 until *Surface is voxelized*;
 end
end
foreach *Slice of the resulting image* **do**
 foreach *Row of the slice* **do**
 | Fill surface;
 end
end

References

- [1] Hahn, H.K., Peitgen, H.O.. IWT - interactive watershed transform: A hierarchical method for efficient interactive and automated segmentation of multidimensional gray-scale images. In: Sonka, M., Fitzpatrick, J.M., editors. *SPIE Medical Imaging: Image Processing*; vol. 5032. SPIE; 2003, p. 643–653. doi:10.1117/12.481097.
- [2] Grady, L.. Random walks for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2006;28(11):1768–1783. doi:10.1109/TPAMI.2006.233.
- [3] Cohen, L.D.. On active contour models and balloons. *CVGIP: Image Understanding* 1991;53:211–218. doi:10.1016/1049-9660(91)90028-N.
- [4] Barrett, W.A., Mortensen, E.N.. Interactive live-wire boundary extraction. *Medical Image Analysis* 1997;1:331–341.
- [5] Bredno, J., Lehmann, T.M., Spitzer, K.. A general discrete contour model in two, three, and four dimensions for topology-adaptive multi-channel segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2003;25:550–563. doi:10.1109/TPAMI.2003.1195990.
- [6] McInerney, T.. SketchSnakes: Sketch-line initialized snakes for efficient interactive medical image segmentation. *Computerized Medical Imaging and Graphics* 2008;32(5):331–352. doi:10.1016/j.compmedimag.2007.11.004.
- [7] Turk, G., Dinh, H.Q., O'Brien, J., Yngve, G.. Implicit surfaces that interpolate. In: *International Conference on Shape Modeling and Applications*. IEEE Computer Society; 2001, p. 62–71. doi:10.1109/SMA.2001.923376.
- [8] Raya, S.P., Udupa, J.K.. Shape-based interpolation of multidimensional objects. *IEEE Transactions on Medical Imaging* 1990;9(1):32–42. doi:10.1109/42.52980.
- [9] Olabariaga, S.D., Smeulders, A.W.M.. Interaction in the segmentation of medical images: A survey. *Medical Image Analysis* 2001;5:127–142.
- [10] Preim, B., Dachselt, R.. *Interaktive Systeme 1: Grundlagen, Graphical User Interfaces, Informationsvisualisierung, Mobile Interaktion*. Springer; 2 ed.; 2010.
- [11] Turk, G., O'Brien, J.F.. Shape transformation using variational implicit functions. In: *ACM SIGGRAPH*. ACM. ISBN 0-201-48560-5; 1999, p. 335–342. doi:10.1145/311535.311580.
- [12] Turk, G., O'Brien, J.F.. Modelling with implicit surfaces that interpolate. *ACM Transactions on Graphics* 2002;21(4):855–873. doi:10.1145/571647.571650.
- [13] Carr, J.C., Beatson, R.K., Cherrie, J.B., Mitchell, T.J., Fright, W.R., McCallum, B.C., et al. Reconstruction and representation of 3D objects with radial basis functions. In: *ACM SIGGRAPH*. ACM. ISBN 1-58113-374-X; 2001, p. 67–76. doi:10.1145/383259.383266.
- [14] Dinh, H.Q., Turk, G., Slabaugh, G.. Reconstructing surfaces by volumetric regularization using radial basis functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2002;24(10):1358–1371. doi:10.1109/TPAMI.2002.1039207.
- [15] Carr, J.C., Beatson, R.K., McCallum, B.C., Fright, W.R., McLennan, T.J., Mitchell, T.J.. Smooth surface reconstruction from noisy range data. In: *International conference on Computer graphics and interactive techniques in Australasia and South East Asia*. ACM. ISBN 1-58113-578-5; 2003, p. 119–126. doi:10.1145/604471.604495.
- [16] Yngve, G., Turk, G.. Robust creation of implicit surfaces from polygonal meshes. *IEEE Transactions on Visualization and Computer Graphics* 2002;8(4):346–359.
- [17] Ohtake, Y., Belyaev, A., Seidel, H.P.. 3D scattered data interpolation and approximation with multilevel compactly supported RBFs. *Graphical Models* 2005;67(3):150–165. doi:10.1016/j.gmod.2004.06.003. Special Issue on SMI 2003.
- [18] Walder, C., Schölkopf, B., Chapelle, O.. Implicit surface modelling with a globally regularised basis of compact support. In: *Computer Graphics Forum (Proceedings of Eurographics)*. Eurographics Association; 2006, p. 635–644.
- [19] Morse, B.S., Yoo, T.S., Rheingans, P., Chen, D.T., Subramanian, K.R.. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In: *ACM SIGGRAPH Courses*. ACM; 2005, p. 78–87. doi:10.1145/1198555.1198645.
- [20] Süßmuth, J., Meyer, Q., Greiner, G.. Surface reconstruction based on hierarchical floating radial basis functions. *Computer Graphics Forum* 2010;29(6):1854–1864. doi:10.1111/j.1467-8659.2010.01653.x.
- [21] Rohr, K.. *Landmark-Based Image Analysis: Using Geometric and Intensity Models*. Springer Netherlands; 2001. ISBN 0792367510.
- [22] Modersitzki, J.. *Numerical Methods for Image Registration*. Oxford University Press; 2003.
- [23] Morse, B., Liu, W., Yoo, T., Subramanian, K.. Active contours using a constraint-based implicit representation. In: *IEEE Conference on Computer Vision and Pattern Recognition*; vol. 1. IEEE Computer Society; 2005, p. 285–292. doi:10.1109/CVPR.2005.59.
- [24] Slabaugh, G., Dinh, Q., Unal, G.. A variational approach to the evolution of radial basis functions for image segmentation. In: *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society; 2007, p. 1–8. doi:10.1109/CVPR.2007.383013.
- [25] Freedman, D., Radke, R., Zhang, T., Jeong, Y., Lovelock, D., Chen, G.. Model-based segmentation of medical imagery by matching distributions. *IEEE Transactions on Medical Imaging* 2005;24(3):281–292. doi:10.1109/TMI.2004.841228.
- [26] Gelas, A., Bernard, O., Friboulet, D., Prost, R.. Compactly supported radial basis functions based collocation method for level-set evolution in image segmentation. *IEEE Transactions on Image Processing* 2007;16(7):1873–1887. doi:10.1109/TIP.2007.898969.
- [27] Wimmer, A., Soza, G., Hornegger, J.. Two-stage semi-automatic organ segmentation framework using radial basis functions and level sets. In: *3D Segmentation in The Clinic: A Grand Challenge, MICCAI Workshop*. Springer; 2007, p. 179–188.
- [28] Carr, J.C., Fright, W.R., Beatson, R.K.. Surface interpolation with radial basis functions for medical imaging. *IEEE Transactions on Medical Imaging* 1997;16:96–107.
- [29] Masutani, Y.. RBF-based representation of volumetric data: Application in visualization and segmentation. In: Dohi, T., Kikinis, R., editors. *International Conference on Medical Image Computing and Computer Assisted Intervention*; vol. 2489 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg; 2002, p. 300–307. doi:10.1007/3-540-45787-9_38.
- [30] Mory, B., Ardon, R., Yezzi, A., Thiran, J.P.. Non-euclidean image-adaptive radial basis functions for 3D interactive segmentation. In: *IEEE International Conference on Computer Vision*. IEEE Computer Society; 2009, p. 787–794. doi:10.1109/ICCV.2009.5459245.
- [31] Schenk, A., Prause, G.P.M., Peitgen, H.O.. Efficient semiautomatic segmentation of 3D objects in medical images. In: *International Conference on Medical Image Computing and Computer Assisted Intervention*. Springer-Verlag. ISBN 3-540-41189-5; 2000, p. 186–195.
- [32] de Bruin, P.W., Dercksen, V.J., Post, F.H., Vossepoel, A.M., Streekstra, G.J., Vos, F.M.. Interactive 3D segmentation using connected orthogonal contours. *Computers in Biology and Medicine* 2005;35(4):329–346. doi:10.1016/j.compbiomed.2004.02.006.
- [33] Liu, L., Bajaj, C., Deasy, J.O., Low, D.A., Ju, T.. Surface reconstruction from non-parallel curve networks. In: *Computer Graphics Forum (Proceedings of Eurographics)*; vol. 27. Eurographics Association; 2008, p. 155–163.
- [34] Haines, E.. *Graphics Gems IV*; chap. Point in Polygon Strategies. Academic Press, Inc.; 1994, p. 24–48.
- [35] Bunch, J.R., Kaufman, L., Parlett, B.N.. Decomposition of a symmetric matrix. *Numerische Mathematik* 1976;27:95–109.
- [36] Newman, T.S., Yi, H.. A survey of the marching cubes algorithm. *Computers & Graphics* 2006;30(5):854–879. doi:10.1016/j.cag.2006.07.021.
- [37] Wyvill, G., McPheeters, C., Wyvill, B.. Data structure for sof objects. *The Visual Computer* 1986;2:227–234. doi:10.1007/BF01900346.
- [38] Latecki, L.J., Lakämper, R.. Convexity rule for shape decomposition based on discrete contour evolution. *Computer Vision and Image Understanding* 1999;73:441–454.
- [39] Goodman, J.E., O'Rourke, J., editors. *Handbook of Discrete and Computational Geometry*. Chapman & Hall; 2 ed.; 2004.
- [40] Horritt, M.S.. A statistical active contour model for SAR image segmentation. *Image and Vision Computing* 1999;17(3–4):213–224. doi:10.1016/S0262-8856(98)00101-2.