Eurographics Workshop on Visual Computing for Biology and Medicine (2010)
D. Bartz, C. P. Botha, J. Hornegger, R. Machiraju, A. Wiebel, and B. Preim (Editors)

Fast and Smooth Interactive Segmentation of Medical Images Using Variational Interpolation

F. Heckel[1], O. Konrad[2] and H.-O. Peitgen[1]

[1]Fraunhofer MEVIS, Germany
[2]MeVis Medical Solutions AG, Germany

Abstract

We present a fast and interactive segmentation method for medical images that allows a smooth reconstruction of an object's surface from a set of user drawn, three-dimensional, planar contours that can be arbitrarily oriented. Our algorithm uses an interpolation based on variational implicit functions.
Because variational interpolation is computationally expensive, we show how to speed up the algorithm to achieve an interactive calculation time while preserving the overall segmentation quality. The performance improvements are based on a quality preserving reduction of the number of contour points and a fast voxelization strategy for the resulting implicit function. A huge speedup is achieved by the parallelization of the algorithms, utilizing modern 64-bit multi-core CPUs. Finally, we discuss how to make the interpolation algorithm more robust to self-intersecting and reduced contours.

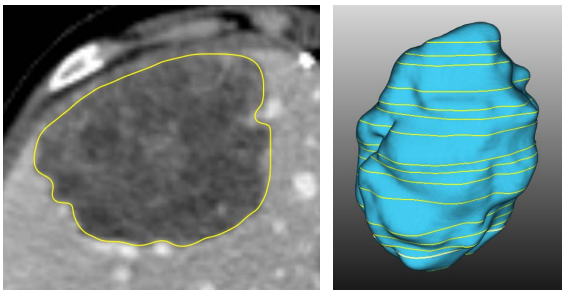
Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.5]: Curve, surface, solid, and object representations—

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. In contrast, manual segmentation is time-consuming and thus not applicable in clinical routine. Therefore, semi-automatic segmentation methods, i.e., methods which require user interactions, can be used in cases where automatic algorithms fail.

Based on a set of three-dimensional, planar contours a smooth surface of an object can be reconstructed using variational implicit functions, resulting in an implicit representation of an object's surface. This is called a variational interpolation. For a variational interpolation, the contours do not have to be parallel. This allows the user to manually segment an object in a few arbitrarily oriented slices. The algorithm guarantees that all contours given by the user are part of the surface and it also extrapolates beyond the contours in a plausible way (see Fig. 1).

Unfortunately, variational interpolation is computationally expensive and there are some special cases which are not handled correctly using a straight forward implementa-



(a) (b)

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 where drawn there.*

DOI: 10.2312/VCBM/VCBM10/009-016

tion. However, for replacing a manual segmentation, the algorithm needs to be both fast and robust.

In this paper we discuss the variational interpolation for a segmentation of medical images with respect to applications in tumor and liver segmentation from CT scans. Moreover, we show how to speed up the algorithm to achieve an interactive calculation time while preserving the overall segmentation quality. The performance improvements are based on a quality preserving reduction of the number of contour points and a fast voxelization strategy for the resulting implicit function. A huge speedup is achieved by the parallelization of the algorithms, utilizing modern 64-bit multicore CPUs. In addition, we discuss how to make the interpolation algorithm more robust to self-intersecting and reduced contours.

2. Related work

A wide range of interactive segmentation methods exist that can roughly be classified into *voxel-based* and *surface-based* methods.

Voxel-based methods try to detect voxels that belong to an object. One such algorithm is the *interactive watershed algorithm* by Hahn and Peitgen that allows the user to influence the result by defining include and exclude markers [HP03]. Another interesting algorithm of this class is the *random walker* algorithm developed by Grady, where the user defines for some of the voxels whether they belong to the background or to the object [Gra06].

Surface-based methods use some kind of surface reconstruction. User interaction is typically performed by drawing contours along the border of an object. Because of the interaction, this is typically a 2D approach. If the surface has been generated, a segmentation can be computed by filling all voxels inside the surface. A wide-spread 2D algorithm is *live-wire* introduced by Barrett and Mortensen, where the user sets points that are dynamically connected by a path that is aligned at edges in the image [BM97]. The main drawback of this algorithm is that the segmentation has to be done on each slice. As a solution, a shape-based interpolation between adjacent contours can be performed to speed up the segmentation process as proposed by Schenk *et al.* [SPP00]. Wolf *et al.* have proposed an interpolation of arbitrarily oriented 2D segmentations based on *Coons-Patches* [WEV*03].

Our method is based on planar contours as well, but, in contrast to many other algorithms, the contours can be arbitrarily oriented. The contours can be drawn freehand, by using algorithms like live-wire or combinations. Based on these contours a smooth, three-dimensional surface is generated that contains the user drawn contours using variational interpolation. Variational interpolation has first been suggested for segmentation by Turk and O'Brien [TO99]. However, to our knowledge, the only work in the context of medical imaging that uses variational interpolation was published by Freedman *et al.*, who have used it for an automatic model-based segmentation of the prostate [FRZ*05].

3. Variational interpolation

In computer graphics, implicit functions are a well known way to model 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* [TO02]. Variational implicit functions are C^1-continuous, i.e., they are rather smooth. Using variational implicit functions, the interpolation problem can be solved in any dimension [TO99]. We call this *variational interpolation*, while in 2D it is called *thin-plate interpolation*. This means, given a set of so called *constraint points*, which are points on the surface of the object, an implicit surface can be created using variational interpolation that passes through each constraint point. This was used by Turk and O'Brien to model shape transformations, i.e., transformations between different 3D objects over time, which is achieved by a 4D interpolation [TO99]. In this paper we will focus on the 3D interpolation.

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

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

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 *radial basis function* $\phi(\mathbf{x})$, the interpolation function $f(\mathbf{x})$ can be written as

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

where $\mathbf{c}_j = (c_j^x, c_j^y, c_j^z)$ are the locations of the constraints, w_j are the weights of each constraints 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.* [CBC*01], a commonly used radial basis function in 3D that minimizes Eq. 1, is the *biharmonic spline*

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

We use the *triharmonic spline* in this paper, which is another commonly used three-dimensional radial basis function. It is defined by

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

The triharmonic spline results in a C^2-continuous and thus smoother interpolation [Roh01].

$f(\mathbf{x})$ must fulfill the constraints $\mathbf{c}_i$ whose values are given

by h_i, i.e.,

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

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

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

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.

3.1. Segmentation using variational interpolation

As already proposed by Turk and O'Brien, variational interpolation can be used for segmentation of medical images based on a set of contours [TO99]. In medical imaging, three-dimensional anatomical data is often acquired as a set of parallel *slices* using CT or MR. An object can be manually segmented by drawing contours along its border in all slices. Using 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 and sagittal), which makes a more accurate segmentation possible, as shown in Fig. 2. Another advantage is that the user can simply add, remove or edit contours if the segmentation is not yet sufficient.

The points of the contours are boundary constraints $\mathbf{c}_i^S$. But for defining the implicit function, we need additional constraints that define which points should be located inside or outside of the object. Turk and O'Brien distinguish between *interior*, *exterior* and *normal constraints* (see Fig. 3) [TDOY01]. The latter allow defining the normal in each boundary constraint. Hence, normal constraints are the best choice for segmentation based on a set of contours, because the contours are interpolated more accurately this way.

The location of a normal constraint $\mathbf{c}_i^N$ is computed by adding the normal to the corresponding boundary constraint $\mathbf{c}_i^S$, which is done for all boundary constraints (i.e., for all points of all contours). 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 of 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 computed using the adjacent points and the normal

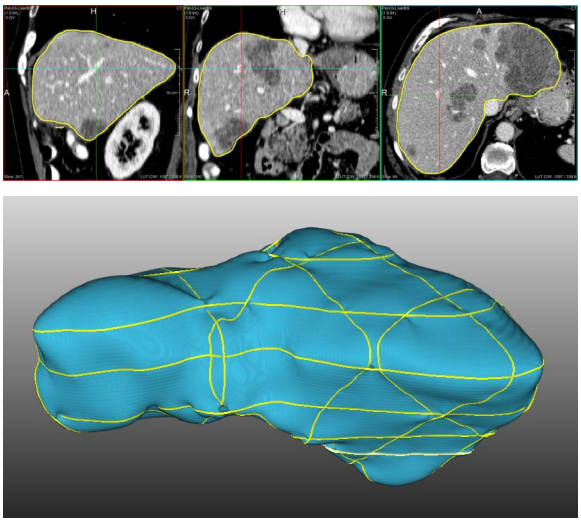
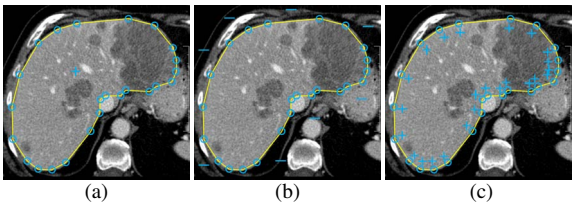


Figure 2: *Example for segmentation of a liver in CT using 14 orthogonal contours.*



(a) (b) (c)

Figure 3: *Different types of constraints for variational interpolation: (a) interior, (b) exterior and (c) normal constraints. Boundary constraints are indicated by blue circles.*

$\mathbf{n}^C$ of the plane in which the contour is defined:

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

For deciding whether $\mathbf{c}_i^N$ is located inside or outside of the contour, we have to perform a point-in-polygon test. In our implementation we use a simple ray casting approach with a binning strategy for efficient calculation.

If there is only one contour or all contours are in the same plane, we add one additional contour for the first input contour that is slightly shifted in the normal-direction of the plane in which the contours are defined. Otherwise the solution of Eq. 6 results in the plane of the contours.

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 never be 0 with $\mathbf{x}^V$ being the position of the voxel's center. Instead, we have to evaluate Eq. 2 for each of

Example	Threads	CLAPACK	ACML	MKL
Metastasis $n = 6246$	1	52.75s	12.26s	10.01s
	2	-	7.41s	5.74s
	4	-	5.07s	3.57s
	8	-	5.28s	2.89s
Liver $n = 13796$	1	548.74s	119.44s	103.97s
	2	-	68.38s	55.98s
	4	-	45.68s	31.84s
	8	-	41.47s	23.42s

Table 1: *Computation times for solving the linear system described in Eq. 6 using different LAPACK implementations. The metastasis example is shown in Fig. 1, the liver example is shown in Fig. 2. n is the size of the matrix (boundary constraints + normal constraints + 4).*

Example	Threads	Voxelization
Metastasis $n = 6246$ $m = 1058508$ $\tilde{m} = 23541$	1	4.15s
	2	2.49s
	4	1.92s
	8	2.13s
Liver $n = 13796$ $m = 10711151$ $\tilde{m} = 169300$	1	62.45s
	2	33.64s
	4	28.95s
	8	32.18s

Table 2: *Computation times for voxelization of the implicit function using the marching cubes scheme. 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 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 of the object ($f(\mathbf{x}_i^V) < 0$).

4. Speeding up computations

As described in the previous section, a segmentation algorithm that uses variational interpolation consists of two computationally intensive steps. The first step is solving the linear system given in Eq. 6. The second step is the voxelization of the resulting implicit function. In the following we will discuss the computation of those two steps. Furthermore, we will describe how to speed up computations while preserving the overall segmentation quality by reducing the number of constraints.

4.1. Solving the linear system

Turk and O'Brien use an LU decomposition to solve Eq. 6, which needs about $\frac{2}{3}n^3$ operations, 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). Fortunately, because the matrix in Eq. 6 is symmetric, a more efficient algorithm developed by Bunch and Kaufman can be used [BKP76] which only takes about $\frac{1}{3}n^3$ operations. The Bunch-Kaufman algorithm has a complexity comparable to a Cholesky decomposition, but it is more stable for matrices that are not positive definite. The algorithm is available as part of the LAPACK library, which we use for solving the linear system.

There are different LAPACK implementations available. The "basic" implementation *CLAPACK* (http://www.netlib.org/clapack) 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. Optimized implementations that are much faster compared to CLAPACK are available in terms of the *Intel Math Kernel Library* (MKL) (http://software.intel.com/en-us/intel-mkl), the *AMD Core Math Library* (ACML) (http://developer.amd.com/cpu/Libraries/acml) and the *Accelerate Framework* (only available on Mac OS X since 10.3) (http://developer.apple.com/performance/accelerateframework.html). 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. The results are shown in Tab. 1. All measurements were performed on an 8-core system (2x Intel Xeon X5550, Turbo Boost and Hyper-Threading disabled, 12 GB RAM, Windows 7 64-bit). We use Intel's MKL in this paper, because it is the fastest library, at least on an Intel CPU. The ACML is much faster than CLAPACK as well and it might be even faster on AMD CPUs.

4.2. Voxelization

As already mentioned, 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, 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 are evaluated.

We use a voxelization method that generates the surface of the object given by $f(\mathbf{x})$, which only needs $O(n\tilde{m})$ operations, where $\tilde{m}$ is the number of voxels on the surface of the object. It is based on a scheme similar to the marching cubes algorithm [NY06]. In real world examples $\tilde{m} \ll m$ holds. Therefore, this method is much faster compared to the naive 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. 2). For voxelization a starting point is needed that is known to be located on the surface of the object. This is true for each boundary constraint. As a consequence, the voxel in which a contour's first boundary constraint is located is used. If the start point is not exactly located on the surface, e.g., due to numerical issues, its neighboring voxels are tested as well. Because the contours might define several objects, we need to use one start point for each contour. If a start point is already part of a voxelized surface, it is rejected. Otherwise, the voxelization of the already processed surface would be repeated.

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 need to be evaluated when filling the object.

Computation times can be reduced by using multiple threads. In our implementation, one thread is used for the evaluation of $f(\mathbf{x})$ for each of the eight corners of a voxel when scanning the surface by moving to the next neighboring voxel. Because typically four or six of the corner values have already been computed when the previous voxels have been evaluated, the theoretical maximum speedup of this parallelization strategy for the voxelization is between two and four. Measurements of the voxelization times are given in Tab. 2. We have used *OpenMP* for the parallelization of the algorithm. The measurements only include the voxelization of the surface. The time for filling the object is negligible ($\ll$ 1s).

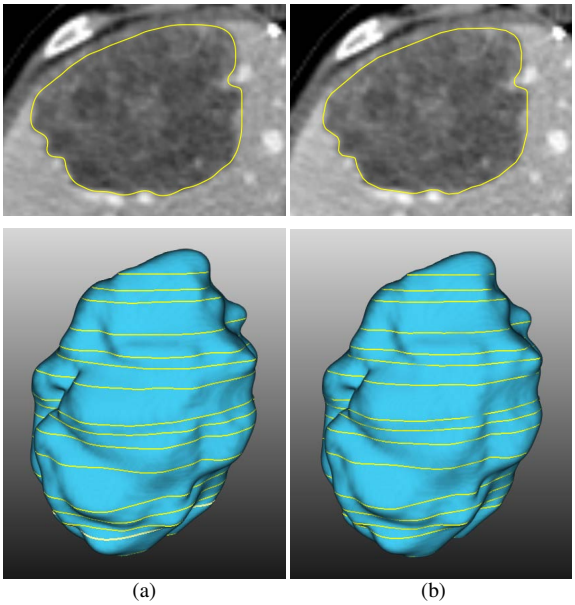
4.3. Constraint reduction

Looking at the results in Tab. 1 and 2 shows that only the computation times of the metastasis example might be called "interactive". The liver example is far too slow even with the fastest LAPACK library and with parallel voxelization. As already stated, the bottlenecks of a segmentation algorithm that is based on variational interpolation are: firstly, finding the solution of the linear system and secondly, the voxelization. The first problem has a complexity of $O(n^3)$, the second one $O(n\tilde{m})$. As described in the Sec. 4.2, we have already minimized the number of voxels $\tilde{m}$ that are evaluated during voxelization. But the greatest impact on the overall computation time is given by the number of constraints. In particular, using only half as much 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 dramatically decreases the overall calculation time.

To achieve an interactive calculation time, we remove contour points $\mathbf{c}_i^S$ that have no or almost no influence on the contours' geometry, which reduces the number of constraints while preserving the visual quality of the contour. This is done in a preprocessing step. As described by Latecki and Lakämper we use two attributes to measure the influence of a contour point on the overall geometry [LL99]: The angle between its adjacent edges ω_i^α and the length of

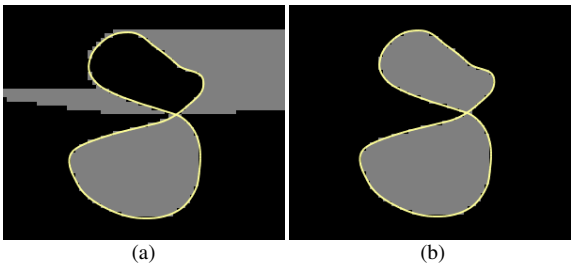
Example	q	n	Time	Overlap
Metastasis	1.0	6246	2.89 / 1.92 / 5.13s	100%
	0.5	3608	0.75 / 1.18 / 2.08s	99.94%
	0.2	1410	0.08 / 0.55 / 0.67s	99.32%
	0.1	710	0.02 / 0.36 / 0.41s	97.05%
Liver	1.0	13796	23.42 / 28.95 / 53.28s	100%
	0.5	7916	4.94 / 16.79 / 21.94s	99.50%
	0.2	3170	0.5 / 7.3 / 7.82s	98.74%
	0.1	1594	0.1 / 8.3 / 8.41s	97.68%

Table 3: *Computation times (linear system solving / voxelization / overall) for different quality settings q. For all measurements the MKL with 8 threads and the voxelization with 4 threads were used. The volume overlap (*Jaccard index*) is given relative to the interpolation result with $q = 1$. It is calculated by the number of intersecting voxels divided by the number of voxels in the union of two segmentations.*



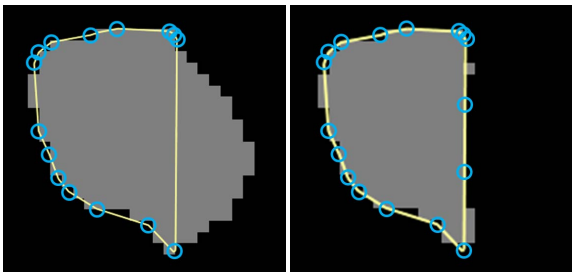
(a) (b)

Figure 4: *Example for reduction of the contours: (a) shows the original contours and the result of the interpolation using all 6.246 constraints, (b) shows the result after reduction with $q = 0.2$ resulting in 1410 constraints. The resulting masks are visually almost identical.*



(a) (b)

Figure 5: *Example for the variational interpolation of a self-intersecting contour: (a) before and (b) after normal correction.*



(a) (b)

Figure 6: *Example for the variational interpolation of a contour after reduction of the number of constraints: (a) before and (b) after inserting additional boundary constraints (blue circles).*

the adjacent edges ω_i^l. These weights are given by:

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

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

The total weight ω_i of a contour point can be calculated by:

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

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. Fig. 4 and Tab. 3 show results of our reduction algorithm. Calculation times for this preprocessing are negligible ($\ll$ 1s) compared to the overall computations and are not included in Tab 3.

5. Making the interpolation more robust

There are some special cases that cannot be handled correctly by the basic algorithm. These cases will be discussed in the following.

5.1. Handling self-intersecting contours

As already described in Sec. 3.1, normal constraints allow a better segmentation result, because the surface interpolates the contours more accurately. 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. 5a.

As a solution, we calculate the normal constraint in a more robust way. By default we assume that the normal constraint is located outside of 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 again, until the next self-intersection line segment is found.

In addition to this normal calculation, we also need additional start points for the voxelization of the surface, because the surfaces surrounded by such contours are typically not coherent. We use the voxels in which the intersection points are located as additional start points, because these points are known to be located on a "new" surface. This way we get a consistent surface even for self-intersecting contours, as shown in Fig. 5b.

5.2. Handling reduced contours

For a fast calculation of the interpolation, the number of constraints is reduced as described in Sec. 4.3. This way, 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. 6a.

We solve this by inserting additional boundary and normal constraints on long line segments if the length of the segment is twice as long as the average length of all segments. The constraints are inserted such that the lengths of the resulting

segments equal the average length (see Fig. 6b). Instead of this "postprocessing", long line segments could already be avoided in the reduction step using an adjusted weight ω_i for each surface constraint.

6. Results

As shown in the examples, variational interpolation allows an accurate and smooth segmentation of objects in medical images (see Fig. 1 and Fig. 2).

The results in Tab. 1 show that using an optimized LAPACK implementation has a huge impact on the time necessary for solving the linear system. Using a voxelization strategy that successively moves along the surface of the object allows a fast conversion from the implicit function to a discrete image. This process can be sped up by using multiple threads as shown in Tab. 2. As expected, our parallelization strategy for the voxelization has its maximum speedup of about 2.16 when using four threads. Using more threads slows down the computation, because the overhead for handling the threads is larger than the speedup of the parallel computation.

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. 3 and Fig. 4. In the liver example, it seems that for $q = 0.1$ the overhead for parallelization increases compared to the overall computations, which results in a higher voxelization time. The overall time measurements in Tab. 3 also include all the robustness optimizations described in Sec. 5. As can be seen, the overhead for improving the robustness is negligible compared to the main bottlenecks of the algorithm.

Our measurements show that interactive segmentation times can be achieved for small objects. However, in our current implementation the voxelization is the main limiting part of the segmentation algorithm, especially for large objects. Thus, the algorithm is not interactive for large objects by now.

7. Limitations

Variational interpolation 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 interactive computation times for small objects. However, in our current implementation and for large objects like the liver, the voxelization becomes the bottleneck after reduction. An improved parallelization strategy is necessary to further speed up the segmentation.

The presented algorithm also has some limitations concerning the segmentation quality. Normal constraints are currently computed based on a single contour, i.e., in 2D. However, neighboring contours influence the normal of the surface as well (see Fig. 7a). Another issue are contradictory contours, i.e., contours that define different surfaces, although they should be located on the same surface. This can happen for non parallel contours like in the liver example, as shown in Fig. 7b. For such contours an additional preprocessing step is necessary that solves such inconsistencies. Another solution might be a morphological postprocessing of the resulting segmentation. In some cases the correct handling of self-intersecting contours fails, because the intersection point used as additional starting point for the voxelization is located on the already found surface instead of the additional surface (see Fig. 7c). A more advanced selection of starting points should solve this. 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 to 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. 7d). Finally, numerical instabilities can arise during the intersection and point-in-polygon tests, which can result in a wrong interpolation result for self-intersecting contours.

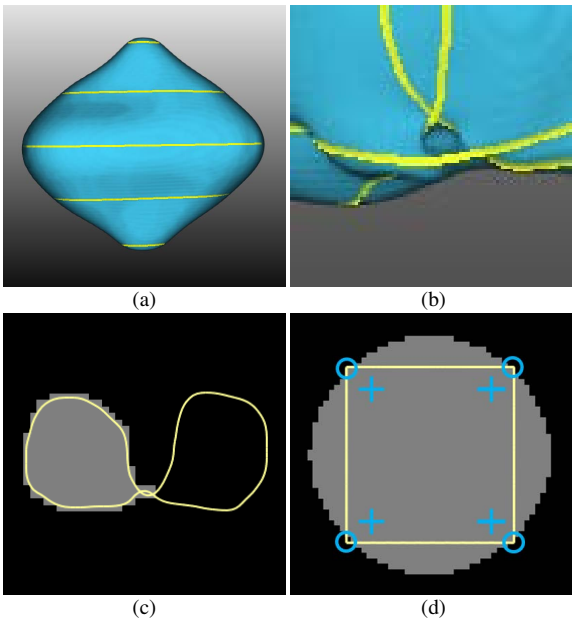


Figure 7: *Limitations of our algorithm: (a) 2D normal constraints (which can be seen at the uneven silhouette), (b) contradictory contours in the liver example, (c) incomplete voxelization of some self-intersecting contours and (d) a square-shaped contour being interpolated to a sphere.*

8. Conclusion and future work

Variational interpolation allows an accurate and robust reconstruction of the surface and thus a segmentation of an object defined by a set of contours. 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. In addition, the algorithm can be used for any three-dimensional modality (e.g., CT, MRI, 3D US) and the interpolation can easily be extended to more dimensions, which allows an interpolation of an object over several time points as well.

Although the algorithm is time and memory consuming, calculation times can dramatically be decreased using modern CPU features and a reduction of the number of constraints. Furthermore, a fast, parallel voxelization method has been presented that only evaluates the voxels on the surface of the object. This way, interactive segmentation times can be achieved for small objects. Using a proper reduction strategy, the differences in the interpolated result are visually not distinguishable from the result using all contour points.

Future work will focus on further reducing the voxelization times by using an improved parallelization strategy. We will also investigate, whether GPU based implementations can improve computation times significantly. For handling self-intersecting contours more robustly, we are going to improve the voxelization by using more appropriate starting points, which is also necessary for a more efficient parallelization. Instead of using a global variational interpolation, local schemes could be used for solving the interpolation problem. For example, Morse *et al.* suggest *compactly supported radial basis functions* that only need $O(n)$ memory and $O(n^{1.5})$ time for solving the interpolation [MYR*05]. Also the voxelization could be sped up this way, because these radial basis functions can be evaluated in $O(\log n)$. Other aspects that have to be evaluated are the number of contours and their orientation as well as the number of constraints that are necessary for an acceptable segmentation.

The presented algorithm will be available in MeVisLab 2.1 (http://www.mevislab.de) as part of the CSO (Contour Segmentation Objects) library, where it is called *CSOConvertTo3DMask*. But because MeVisLab only ships with CLAPACK, this module calculates slower compared to the results presented in this paper.

Acknowledgment

Parts of this work were funded by Siemens AG Healthcare Sector Imaging & IT Division Computed Tomography, Forchheim.

References

[BKP76] BUNCH J. R., KAUFMAN L., PARLETT B. N.: Decomposition of a symmetric matrix. *Numerische Mathematik 27* (1976), 95–109. 4

[BM97] BARRETT W. A., MORTENSEN E. N.: Interactive live-wire boundary extraction. *Medical Image Analysis 1* (1997), 331–341. 2

[CBC*01] CARR J. C., BEATSON R. K., CHERRIE J. B., MITCHELL T. J., FRIGHT W. R., MCCALLUM B. C., EVANS T. R.: Reconstruction and representation of 3D objects with radial basis functions. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2001), ACM, pp. 67–76. 2

[FRZ*05] 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 24*, 3 (March 2005), 281–292. 2

[Gra06] GRADY L.: Random walks for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 28*, 11 (2006), 1768–1783. 2

[HP03] 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 *SPIE Medical Imaging* (2003), Sonka M., Fitzpatrick J. M., (Eds.), vol. 5032, SPIE, pp. 643–653. 2

[LL99] LATECKI L. J., LAKÄMPER R.: Convexity rule for shape decomposition based on discrete contour evolution. *Computer Vision and Image Understanding 73* (1999), 441–454. 5

[MYR*05] 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 2005 Courses* (New York, NY, USA, 2005), ACM, p. 78. 8

[NY06] NEWMAN T. S., YI H.: A survey of the marching cubes algorithm. *Computers & Graphics 30*, 5 (2006), 854 – 879. 4

[Roh01] ROHR K.: *Landmark-Based Image Analysis: Using Geometric and Intensity Models*. Kluwer Academic Publishers, Norwell, MA, USA, 2001. 2

[SPP00] SCHENK A., PRAUSE G. P. M., PEITGEN H.-O.: Efficient semiautomatic segmentation of 3D objects in medical images. In *MICCAI '00: Proceedings of the Third International Conference on Medical Image Computing and Computer-Assisted Intervention* (London, UK, 2000), Springer-Verlag, pp. 186–195. 2

[TDOY01] TURK G., DINH H. Q., O'BRIEN J., YNGVE G.: Implicit surfaces that interpolate. In *Shape Modeling and Applications, SMI 2001 International Conference on.* (May 2001), pp. 62–71. 3

[TO99] TURK G., O'BRIEN J. F.: Shape transformation using variational implicit functions. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1999), ACM Press/Addison-Wesley Publishing Co., pp. 335–342. 2, 3

[TO02] TURK G., O'BRIEN J. F.: Modelling with implicit surfaces that interpolate. *ACM Transactions on Graphics 21*, 4 (2002), 855–873. 2

[WEV*03] WOLF I., EID A., VETTER M., HASSENPFLUG P., MEINZER H.-P.: Segmentierung dreidimensionaler objekte durch interpolation beliebig orientierter, zweidimensionaler segmentierungsergebnisse. In *Bildverarbeitung für die Medizin* (2003), pp. 343–347. 2