Journal of Mathematical Imaging and Vision
https://doi.org/10.1007/s10851-024-01185-y

Finding Space-Time Boundaries with Deformable Hypersurfaces

Patrick M. Jensen[1] · J. Andreas Bærentzen[1] · Anders B. Dahl[1] · Vedrana A. Dahl[1]

Received: 8 December 2022 / Accepted: 11 March 2024

Abstract

Dynamic 3D imaging is increasingly used to study evolving objects. We address the problem of detecting and tracking simple objects that merge or split in time. Common solutions involve detecting topological changes. Instead, we solve the problem in 4D by exploiting the observation that if objects only merge or only split, they appear as a single component in 4D. This allows us to initiate a topologically simple 3D hypersurface and deform it to fit the surface of all objects at all times. This gives an extremely compact representation of the objects' evolution. We test our method on artificial 4D images and compare it to other segmentation methods. We also apply our method to a 4D X-ray data set to quantify evolving topology. Our method performs comparably to existing methods with better resource use and improved robustness.

Keywords 4D images · Deformable models · Segmentation · Topology

1 Introduction

4D images refer to a series of 3D volumes acquired over time, for example, via dynamic X-ray computed tomography (CT). Such images may be used to study the evolution of 3D objects and have applications in *e.g*. medicine [1] and materials science [2–4]. To use 4D images in quantitative studies, we need segmentation methods that can detect and track objects over time. Additionally, some systems contain objects which merge or split as part of their time evolution, and such events are often of interest on their own [2, 5]. Therefore, a segmentation method should handle topological changes and still obtain an accurate result.

Furthermore, due to advancements in imaging technology [6, 7], 4D images can now be acquired with a very high spatial and temporal resolution, i.e., a voxel size ≤ 5 μm at hundreds of 3D volumes per second. This places high demands on the computational efficiency of segmentation methods.

In this paper, we address the problem of 4D segmentation via a simplifying assumption, which we demonstrate leads to a simple, elegant, and highly efficient segmentation method. We assume that objects only perform splits or merges during their evolution, but *not* both. While this assumption is slightly limiting, many interesting systems exhibit this behavior. For example, biological cells divide but do not merge, and bubbles in a foam merge but do not split. Furthermore, for such systems, enforcing that objects are only allowed split or merge can indeed be a benefit, as it exploits our prior knowledge of the system.

To illustrate our method, we initially describe it in 2D+time for simplicity. Consider the case in Fig. 1a where two solid blobs grow until they merge together to a single object. The scenario of splitting is identical, as we can transform one to the other by reversing time. We can think of the objects at time t as the intersection of a 3D connected component and an xy-plane at time t, as shown in Fig. 1b. To segment the objects, we could aim to segment the full 2D+time connected component. However, in 4D, this will require a large amount of memory. Instead, we could just represent the boundary by segmenting the object contours at every time, as in Fig. 1c. Yet, this still uses excessive memory and does not exploit our assumption of only splits or merges.

Our core observation is that if we assume solid objects *only* merge (or *only* split) the boundary of their 2D+time con-

✉ Patrick M. Jensen
patmjen@dtu.dk

J. Andreas Bærentzen
janba@dtu.dk

Anders B. Dahl
abda@dtu.dk

Vedrana A. Dahl
vand@dtu.dk

[1] Department of Applied Mathematics and Computer Science, Technical University of Denmark, Richard Petersens Plads, 2800 Kgs. Lyngby, Denmark

Published online: 09 April 2024

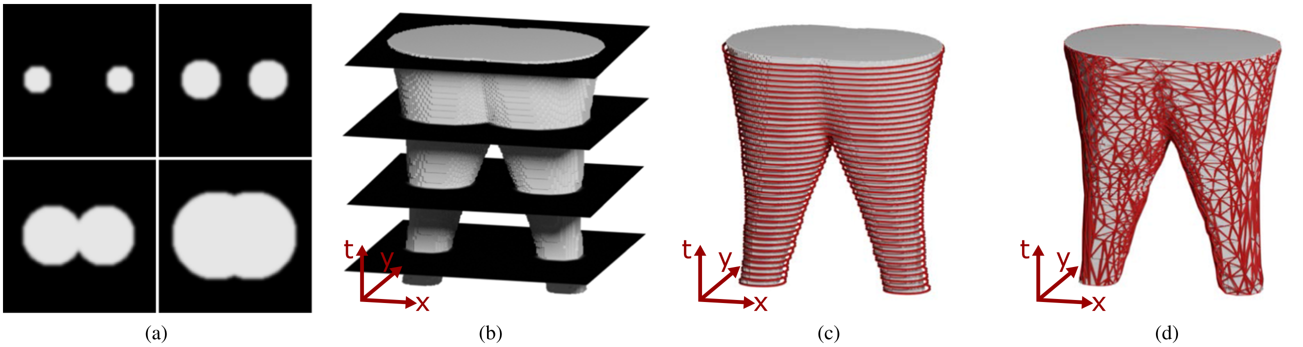


Fig. 1 The principle of the 4D segmentation approach illustrated on a lower-dimensional 3D (2D+time) example. **a** Time series of 2D images, where two disks grow and merge. **b** 2D images are stacked to form a 3D (2D+time) volume. Here, the merging disks appear as a single simply connected component. **c** Segmentation given by sequentially detecting the disks in each 2D image. **d** Segmentation given by detecting the 2D boundary of the 3D connected component

nected component is homeomorphic to a sphere. Thus, we can represent its boundary as a triangle mesh, $\mathcal{M}$, as shown in Fig. 1d. However, note that we do not have to close the mesh in the top (corresponding to time steps after the final image) which in practice means that the triangle mesh has n-disk topology rather than n-sphere topology. For 4D data, the mesh used for segmentation is a tetrahedral mesh embedded in 4D. Thus, we only have a *single* mesh whose cross sections represent the contours we seek. This is significantly less memory intensive than representing a segmentation for each time step.

We now proceed as follows. Given a sequence of input 2D images with merging objects, we segment the final image. This gives a contour which we fill in to produce a triangle mesh that represents the initial stage of $\mathcal{M}$. The surface mesh is now allowed to deform in 3D (i.e., 2D+time). Similar to classic active contours [8] or deformable surfaces [9, 10], this deformation is driven by an internal energy which keeps it smooth and external energy which attracts the mesh to object boundaries. When the computation stops, our spatiotemporal data has been segmented. We can obtain a segmentation at an arbitrary point in time, t, by intersecting $\mathcal{M}$ with a plane corresponding to that specific value of t. This results in a contour describing the boundary of the segmented object(s) at the given time.

The most significant benefit of our approach is that it allows us to precisely locate splits (or merges) in both time and space. Consider the mapping from a point $p \in \mathcal{M}$ to time $t(p)$. As shown in Fig. 1d, the points where splits occur correspond to saddle points of $t(p)$ where the gradient is zero. Further, the center of the initial objects corresponds to the minima of t. In other words, by finding the singularities of $t(p)$, as well as how they are connected, we can detect where specific objects split. This strategy also makes our approach robust to spurious self-intersections, since they do not affect the location of the singularities. In practice, we exploit this by computing the Reeb graph [11, 12] of $\mathcal{M}$ with respect to $t(p)$ since the Reeb graph contains information both about the placement and connectivity between the singularities.

From a computational standpoint, our approach also offers considerable benefits. As we only store a boundary, we have a very compact description of the evolution of an object. Partly because storing the boundary inherently requires significantly less space. But also because a tetrahedron in 4D can span multiple time values, which exploits the coherence between two time-adjacent 3D volumes. This compactness also makes our method fast, which allows us to segment very large 4D images in only a few minutes. To make our method accessible, we publish a Python implementation at https://github.com/patmjen/4dsurf.

1.1 Related Work

Generally, existing methods for segmenting 4D images fall into two categories: mesh-based methods and voxel-based methods. Mesh-based methods are mainly used for tracking non-interacting objects over time. In [10, 13], a sequential approach is used where the fit in one 3D volume is used to improve the fit in the next. Here, one needs to establish a correspondence between segmentations in consecutive 3D volumes in order to obtain object tracking. When objects merge or split, this can be highly non-trivial, although some works have suggested approaches for handling this, *e.g.* [14, 15]. Others [16–19] use a more global approach where a separate mesh is placed in every 3D volume and then all meshes are fitted simultaneously. Common for these methods is that the mesh connectivity is kept fixed and only vertex positions are changed. This provides a natural correspondence between meshes and makes it easy to track the development of an object over time. However, it does not allow for the object to split or merge.

Voxel-based approaches on the other hand can easily handle topology changes. Here, authors have used methods ranging from level sets [5, 20], Markov random fields [21, 22], to deep learning [13, 23]. However, while voxel-based approaches are more flexible regarding the evolution of the object, they do not (directly) give access to a mesh-based representation, which can make postprocessing more challenging. Furthermore, small errors in the segmentation can have large effects on the resulting topology as two objects may merge prematurely. And finally, since modern 4D images have sizes measured in the tens of gigabytes (GB) to several terabytes (TB), segmenting 4D images with voxel-based methods quickly becomes unwieldy unless the data is severely downsampled.

2 Method

Deformable models are widely used in 2D and 3D segmentation in the form of curves and surfaces. In 4D segmentation, hypersurfaces that deform in space-time are rarely encountered, and existing approaches treat the time dimension differently than the space dimensions. In this section, we propose the methodology needed to generalize parametric deformable surfaces to 4D, providing a combined treatment of the deformation in space and time. We begin by introducing the continuous formulation of 4D deformable models and then detail how we discretize the problem. In the subsequent sections, we then detail each component of our 4D deformable model approach.

2.1 Continuous 4D Deformable Model

Given a 4D image, $I : \Omega \subset \mathbb{R}^4 \rightarrow \mathbb{R}$, we aim to find a partition of the domain Ω into two disjoint regions, Ω_{in} and Ω_{out}, separated by a boundary Γ. The region Ω_{in} corresponds to the inside of the 4D connected component we wish to segment and Ω_{out} to the outside. As Γ uniquely determines the regions Ω_{in} and Ω_{out} we formulate our problem as a search for a boundary, Γ^*, that minimizes the following energy functional

$$\Gamma^* = \arg\min_{\Gamma} E_{\text{ext}}(\Gamma) + E_{\text{int}}(\Gamma). \tag{1}$$

Here, E_{ext} is an external energy term that ensures that Ω_{in} and Ω_{out} correspond to the image regions we wish to segment and E_{int} is an internal energy that acts as a regularizer by encouraging the boundary Γ to remain smooth.

For E_{ext}, we use the popular Chan-Vese energy [24], which assumes the image can be modeled as a piecewise constant function with value c_{in} in Ω_{in} and c_{out} in Ω_{out}. It is given as

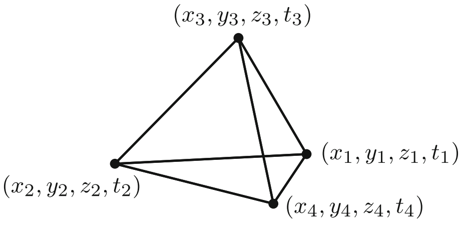


Fig. 2 Single element of the tetrahedral mesh. Each vertex position, $\mathbf{x}_i$, has four coordinates: x_i, y_i, z_i, and t_i

$$\begin{aligned} E_{\text{ext}}(\Gamma) = &\int_{\Omega_{\text{in}}} (I - c_{\text{in}})^2 \, \mathrm{d}\Omega \\ &+ \int_{\Omega_{\text{out}}} (I - c_{\text{out}})^2 \, \mathrm{d}\Omega. \end{aligned} \tag{2}$$

Generally, c_{in} and c_{out} may be unknown values which are also optimized over. However, as that adds a significant computational cost, we assume in this paper that c_{in} and c_{out} are known a priori.

For E_{int}, we use the Dirichlet energy

$$E_{\text{int}}(\Gamma) = \gamma \int_{\Gamma} \|\nabla \Gamma(\mathbf{x})\|_2^2 \, \mathrm{d}\mathbf{x}, \tag{3}$$

where $\gamma \geq 0$ is a fixed parameter that controls the degree of regularization.

2.2 Discrete 4D Deformable Model

To find the optimal boundary, we use a discretized version of Γ. Since Γ is the boundary between two 4D regions, it is a 3D hypersurface embedded in 4D [25]. We represent Γ with a tetrahedral mesh $\mathcal{T}$ with vertices $\mathbf{X} \in \mathbb{R}^{V \times 4}$ where vertex positions $\mathbf{x}_i$ are given by four coordinates as shown in Fig. 2.

Given an initial mesh $\mathcal{T}^{(0)}$, a solution to the discretized version of (1) can be found by gradient descent of the energy [8, 9]

$$\begin{aligned} &\nabla_{\mathbf{X}} E_{\text{ext}}(\mathcal{T}^{(i-1)}) + \nabla_{\mathbf{X}} E_{\text{int}}(\mathcal{T}^{(i-1)}) \\ &\quad = -\frac{1}{\tau}(\mathbf{X}^{(i)} - \mathbf{X}^{(i-1)}), \end{aligned} \tag{4}$$

where τ is a step size. The gradient of the Dirichlet energy $\nabla E_{\text{int}}(\Gamma)$ is given by the Laplacian $\Delta\Gamma$ [26]. This gives the update equation

$$(\mathbf{I} + \lambda \mathbf{L})\,\mathbf{X}^{(i)} = \mathbf{X}^{(i-1)} - \tau \nabla E_{\text{ext}}(\mathcal{T}^{(i-1)}), \tag{5}$$

where $\mathbf{L}$ is a matrix approximating the Laplacian of Γ, and $\lambda = \tau\gamma$. We describe our initialization strategy in Sect. 2.3.

As the mesh deforms from its initial configuration the tetrahedra will stretch, meaning the mesh will not be able to fit the image to sufficient detail. Therefore, we adaptively subdivide the mesh every n_{sub} iteration to prevent the mesh from becoming too coarse. We detail our subdivision strategy in Sect. 2.4.

For the external energy gradient in (5), it can be shown that the vertex displacements corresponding to $\nabla E_{\text{ext}}(\mathcal{T})$ are given by

$$\frac{\partial E_{\text{ext}}}{\partial \mathbf{x}_i} = 2(c_{\text{in}} - c_{\text{out}})(I(\mathbf{x}_i) - (c_{\text{in}} + c_{\text{out}})/2)\mathbf{n}_i, \tag{6}$$

where $\mathbf{n}_i$ is the normal for vertex i [27]. As we adaptively subdivide the mesh during deformation, we want our normals to be robust to this. Therefore, we have developed an extension of angle-weighted normals to 4D, which we describe in Sect. 2.5.

For the Laplacian matrix, we use the scale-dependent Laplacian [28], in order to be robust to irregularly sized tetrahedra [26]. This computes the Laplacian at the i'th vertex as

$$L(\mathbf{x}_i) = \frac{1}{E} \sum_{j \in N(i)} \frac{\mathbf{x}_j - \mathbf{x}_i}{e_{ij}}, \quad E = \sum_{j \in N(i)} e_{ij}, \tag{7}$$

where $e_{ij} = \left\| \mathbf{x}_j - \mathbf{x}_i \right\|$ and $N(i)$ are the neighbor vertices of vertex i. Since $\mathbf{L}$ is sparse, we can solve the linear system in (5) efficiently using conjugate gradient iteration. However, constructing $\mathbf{L}$ is relatively expensive, and we therefore only update it when we subdivide the mesh. While not strictly correct, we did not observe drawbacks with this strategy.

Finally, to extract the segmentation for a 3D volume at a given time, we compute a cross section of the fitted 4D mesh using the method detailed in Sect. 2.6.

2.3 Mesh Initialization

We assume objects only merge (similar to the example Fig. 1), since splitting can be viewed as merging in reverse. Thus, if the initial mesh is placed in the last 3D volume of the 4D image, it only needs to propagate backward in time. This makes a hyperdisk (*i.e.*, a solid 3D ball) a good candidate for an initial mesh. The boundary of a hyperdisk forms a 2D surface, which we can represent with a triangle mesh.

This leads to the following initialization approach. First, fit a triangle mesh to the 2D object boundary in the last 3D volume. Since we only consider a single 3D volume here, we can ignore the time coordinate and treat it as a traditional 3D segmentation task. As a result, we can use an existing 3D method such as deformable surfaces [10], graph cut approaches [29, 30], or an isosurface [31] from a voxel segmentation to find the triangle mesh. We now let this triangle mesh be the boundary of the hyperdisk, and then tetrahedralize it using the TetGen tool by Si [32]. Finally, we add the time coordinate to the tetrahedron vertices, which results in the initial mesh. An illustration of the initialization approach for a lower-dimensional (2D+time) example is shown in Fig. 3.

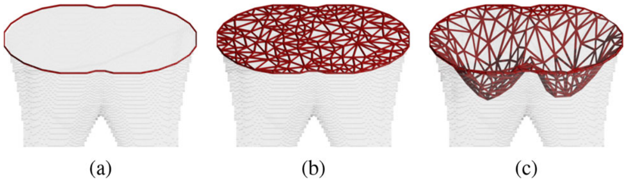


Fig. 3 The principle of initialization process illustrated on a 3D (2D+time) example. **a** First, the boundary of the object in a single image (volume in 4D) is located. **b** The interior of the boundary is triangulated (tetrahedralized in 4D), which gives the initial mesh. **c** The initial mesh is iteratively deformed to fit to the boundary of the connected component

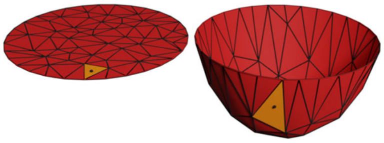


Fig. 4 Illustration of the mapping from the coordinate mesh (left) to the deformed mesh (right). A point inserted in the highlighted face in the coordinate mesh can be mapped to its corresponding position in the deformed mesh using barycentric coordinates

To avoid the mesh leaving the last 3D volume, we constrain the time coordinate of the tetrahedron vertices that correspond to the original triangle vertices to remain unchanged.

2.4 Adaptive 4D Mesh Subdivision

A considerable amount of research has been conducted on creating and subdividing tetrahedral meshes in 3D, and we want to leverage this. We will use an idea similar to that of coordinate charts from differential geometry [33], which are maps from manifolds to Euclidean space. In our case, we know the initial mesh is contained in a single 3D volume. After the mesh has been deformed in 4D, the initial mesh defines a natural mapping between 3D Euclidean space and the 4D mesh. For a point placed anywhere within the initial mesh, we can then use barycentric coordinates to find its corresponding position in 4D. See Fig. 4 for an illustration.

Specifically, we keep a copy of the initial mesh, $\mathcal{T}^{(0)}$, during deformation—henceforth referred to as the coordinate mesh, $\mathcal{T}^{C}$. To perform subdivision, we find all tetrahedra whose volume has grown larger than a threshold, s, and insert a new point at the barycenter of each tetrahedron. The volume of a tetrahedra with vertex positions $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$, and $\mathbf{x}_4$ is given by [34, 35]

$$\mathrm{vol}(\mathbf{T}) = \frac{1}{6}\sqrt{G(\mathbf{x}_2 - \mathbf{x}_1, \mathbf{x}_3 - \mathbf{x}_1, \mathbf{x}_4 - \mathbf{x}_1)}, \tag{8}$$

where G is the Gram determinant. Then, the original vertices of $\mathcal{T}^C$, along with the new points, are re-tetrahedralized with TetGen [32], which gives a new coordinate mesh. This is then re-mapped to 4D, after which the old coordinate and 4D mesh are thrown away.

While performing a full re-tetrahedralization is more expensive than updating the mesh, it has the benefits of resulting in a higher quality mesh, and is simple to implement with tools designed for 3D meshes.

2.5 Angle Weighted Normals in 4D

Analogous to surface meshes in 3D, vertex normals are defined as a weighted average of the adjacent tetrahedral face normals. These are in turn defined using a 4D analog of the cross product [36]; given 4D vectors **u**, **v**, and **w**, we can find an orthogonal vector, $\mathbf{n} = (n_x, n_y, n_z, n_t)^{\mathrm{T}}$, as

$$\begin{aligned} n_x &= \begin{vmatrix} u_y & u_z & u_t \\ v_y & v_z & v_t \\ w_y & w_z & w_t \end{vmatrix}, \quad n_y = \begin{vmatrix} u_x & u_z & u_t \\ v_x & v_z & v_t \\ w_x & w_z & w_t \end{vmatrix}, \\ n_z &= \begin{vmatrix} u_x & u_y & u_t \\ v_x & v_y & v_t \\ w_x & w_y & w_t \end{vmatrix}, \quad n_t = \begin{vmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ w_x & w_y & w_z \end{vmatrix}, \end{aligned} \tag{9}$$

where $|\cdot|$ is the determinant. For a tetrahedron with vertex positions $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$, and $\mathbf{x}_4$, we form the vectors as $\mathbf{u} = \mathbf{x}_2 - \mathbf{x}_1$, $\mathbf{v} = \mathbf{x}_3 - \mathbf{x}_1$, and $\mathbf{w} = \mathbf{x}_4 - \mathbf{x}_1$.

Previous works have used simple averaging or weighted the normal of each tetrahedron by its volume [37]. However, as for triangle meshes [38], these weighting schemes are sensitive to re-tessellation of the mesh. In this work, we extend angle-weighted normals to four dimensions by weighting the contribution of each tetrahedron by the solid angle spanned by the tetrahedron at **v**. Assume, without loss of generality, that the mesh is scaled such that all edges are longer than 1. The solid angle, ω, is then given by the area of the spherical triangle formed by the intersection of the tetrahedron and a unit sphere, as illustrated in Fig. 5.

The area of the spherical triangle ω is given by

$$\omega = \theta_1 + \theta_2 + \theta_3 - \pi, \tag{10}$$

where θ_1, θ_2, and θ_3 are the dihedral angles of the tetrahedron edges connected to **v** [39].

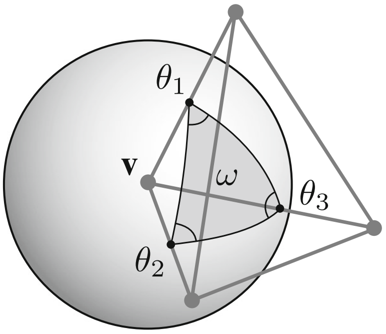


Fig. 5 The solid angle ω at vertex **v** is the area of the spherical triangle given by the intersection of the tetrahedron and a unit sphere centered at **v**. θ_1, θ_2, and θ_3 denote the dihedral angles along the tetrahedron edges connected to v

The normal at vertex i is then given by

$$\mathbf{n}_i = \frac{\sum_{T \in \mathcal{T}_i} \omega_T \mathbf{n}_T}{\left\| \sum_{T \in \mathcal{T}_i} \omega_T \mathbf{n}_T \right\|}, \tag{11}$$

where $\mathcal{T}_i$ are the incident tetrahedra for vertex i with normals $\mathbf{n}_T$ and incident solid angles ω_T. The denominator ensures that the resulting normal is unit length. Similar to angle-weighted normals for triangle meshes, this weighting scheme is invariant to re-tessellation of the mesh.

As a proof, consider re-tesselating a tetrahedron meaning splitting it into multiple sub-tetrahedra where the original vertices remain in place and all new vertices are convex combinations of the original vertices. In this case, the face normals of the new tetrahedra will be equal to the original face normal. Furthermore, the solid angles of the sub-tetrahedra incident with **v** will sum to the original solid angle ω. Therefore, the contribution of the incident sub-tetrahedra to the vertex normal of **v** will be equal to the contribution of the original tetrahedron.

2.6 Cross sections of Tetrahedral Meshes in 4D

Computing a cross section is equivalent to finding the intersection with a hyperplane or extracting an isosurface. We use a simplified version of the marching tetrahedron method [40], which has been used previously for visualizing 4D tetrahedral meshes [36].

We assume without loss of generality that we compute the intersection with the xyz-hyperplane at $t = 0$. If a tetrahedron intersects a hyperplane, the intersection will always be one of the eight cases illustrated in Fig. 6. The last three cases are problematic as the intersections are not surface elements. To avoid these, we perturb the time coordinate of all vertices which lie on the hyperplane by a small $\varepsilon \ll 1$, which guarantees we will only encounter the first two cases.

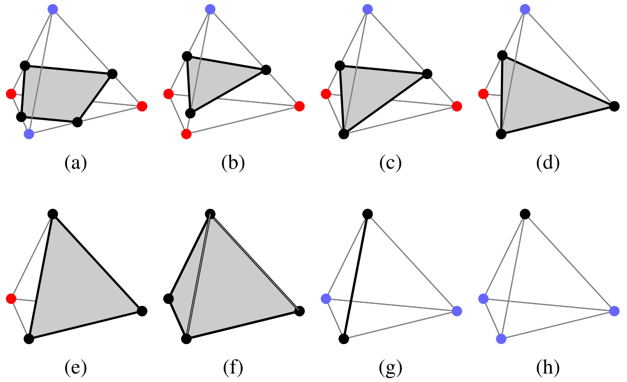


Fig. 6 All possible ways a tetrahedron in 4D may intersect a hyperplane. Black vertices are on the hyperplane ($t = 0$), red vertices are 'below' ($t < 0$), and blue vertices are 'above' ($t > 0$). The intersection can be **a** a quadrilateral (which may be split into two triangles), **b**–**e** a triangle, (f) the entire tetrahedron, **g** a line, or **h** a point

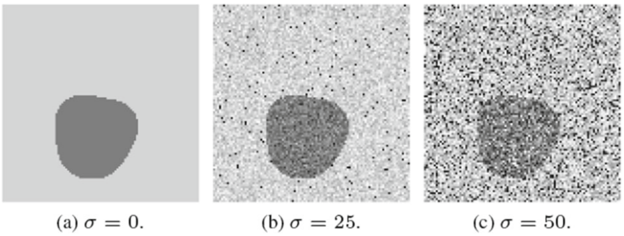

(a) $\sigma = 0$. (b) $\sigma = 25$. (c) $\sigma = 50$.

Fig. 7 2D cross sections of the last 3D volume in the artificial 4D image for different values of the noise level σ

3 Results

To assess the performance of our method, we perform a series of numerical experiments, where we segment computer-generated 4D images with known ground truth. This allows a quantitative assessment and comparison with other methods. Next, we apply the method to a 4D dataset from dynamic X-ray CT. Here, a ground truth segmentation is not known, so we can only do a qualitative evaluation. Finally, we demonstrate how our method allows for tracking the evolution of splitting/merging objects through time by computing the Reeb graph [11, 12] of the fitted tetrahedral mesh.

3.1 Numerical experiments

We created an artificial 4D image containing five organic-looking blobs, shown in the top row of Fig. 8. As time progresses, the blobs move toward each other until they collide and merge to a new blob whose volume grows to equal the sum of the previous two blobs. Furthermore, as the blobs move, they rotate and deform to make the data more challenging to segment.

The 4D image consists of 200 binary label volumes of size 100 × 100 × 100 voxels. Before segmenting, the intensity of the 4D image is transformed so the blobs have an intensity of 100 and the background an intensity of 200. After that, we add zero-mean Gaussian noise with standard deviations of 25 and 50 to create two additional 4D images. Figure 7 shows a cross section of the last 3D volume in each of the 4D images.

We segment the artificial images with the proposed method and compare with three other segmentation methods. These methods were chosen because they are well established and form the basis of many segmentation approaches [41].

1. Sequential 3D Markov Random Fields (MRFs). We segment each 3D image independently in sequence. The image segmentation is modeled as a graph minimum cut problem as detailed in [42]. Voxels are nodes in a graph and are connected to a special source and sink node. Each source/sink edge has a cost proportional to the background/foreground probability. Additionally, each node is connected to its six spatial neighbors with an edge whose cost controls the segmentation smoothness. We now seek a partition of the nodes into two disjoint sets by cutting edges of the graph until there is no path from the source to the sink node. The cost of the partition is the sum of the cut edges and we find the partition of minimum cost via the popular algorithm by Boykov and Kolmogorov (BK) [42].
2. 4D Markov Random Fields (MRFs). We use the same construction as for 3D MRFs, but compute a single segmentation for the entire 4D images at once. As such, we also add edges between graph nodes and their temporal neighbors (in addition to their six spatial neighbors). Again, the segmentation is computed with BK algorithm [42]. We note that one could also use smaller overlapping temporal windows, instead of considering the entire 4D image, in order to reduce the computational costs. To be able to compare results with a global method, we chose not to do this.
3. Surface Detection via Graph Cut (GC). An initial mesh is placed at a user-defined location and deformed to fit the image contours, as detailed in [29, 30, 43]. New candidate positions for each mesh vertex are sampled equidistantly in a column along the outward vertex normals. These candidates are graph nodes connected to a special source and sink node. Each source/sink edge has an associated cost, computed as in [44], related to the probability of being inside/outside the deformed mesh. Nodes in neighboring column are connected with edges to enforce that vertex displacement varies by a maximum of Δ steps to ensure a smooth surface. We then compute an optimal node partition into 'inside' and 'outside' with the BK algorithm [42] and move each vertex to the outermost 'inside' node in its column.

The initial mesh is an 8-frequency subdivided icosahedron approximating a sphere. We process each volume

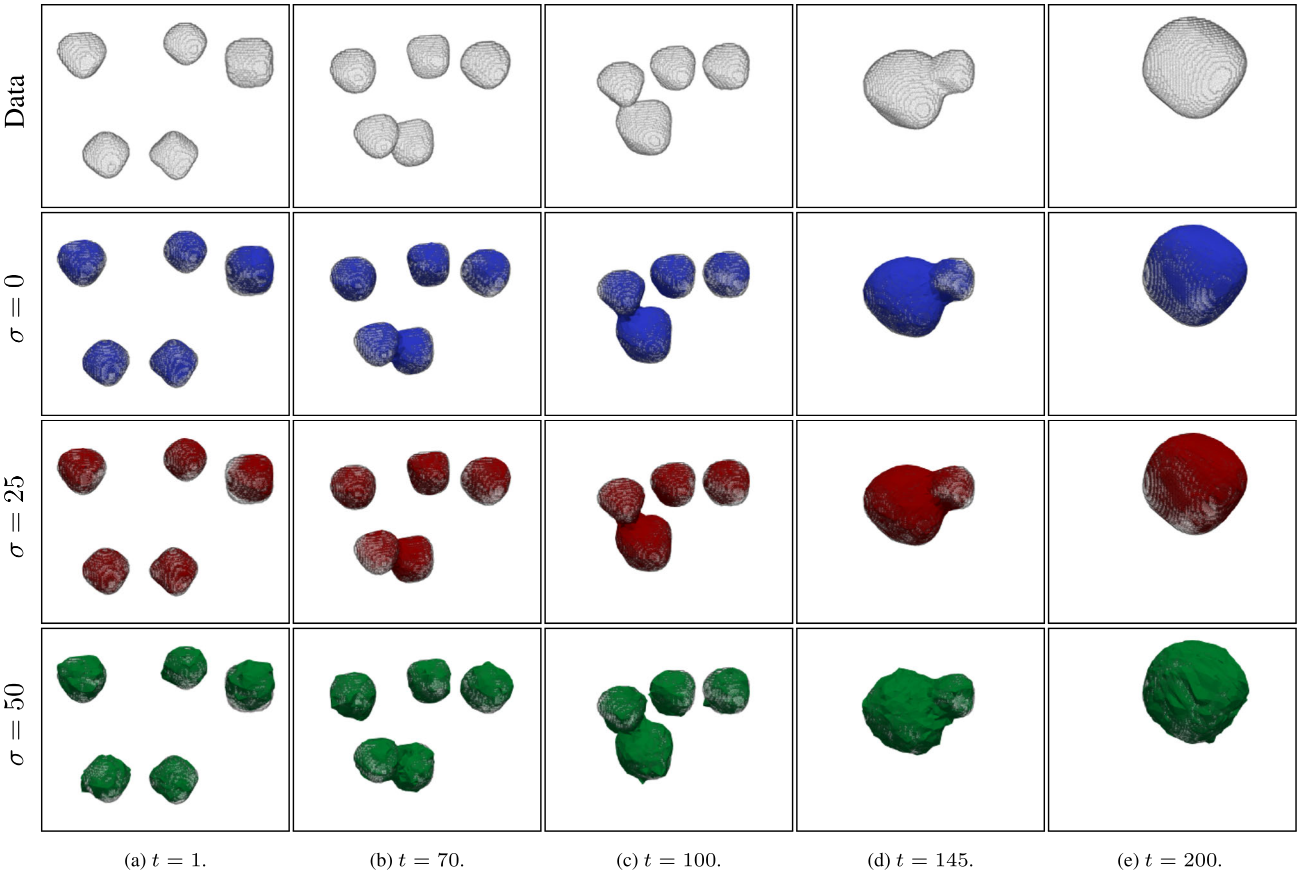


Fig. 8 Segmentation results for the artificial 4D image using the proposed method. The rows show 3D renderings of (top to bottom): the data, segmentation for $\sigma = 0$, segmentation for $\sigma = 25$, and segmentation for $\sigma = 50$. The segmentations have been overlaid on the data

sequentially and place a mesh at the centroid of each connected component of the ground truth label volume. Note that this represents a best-case scenario for this method, as it requires either a good pre-segmentation of the data or manual annotation at every time step. Furthermore, the sequential nature of this method means that we do not have any correspondence between segmentations in different 3D volumes as we do with our method. The correspondences would have to be established afterward which is non-trivial. We apply the graph cut method for two values of the max. difference parameter, Δ. A large value gives the method more freedom to fit elongated shapes, while a smaller value makes it more robust to noise.

For our method, we scale each spatial axis to $[-1, 1]$ and the time axis to $[-2, 2]$. We scale the time axis to ensure that the image features have roughly equal curvature. This allows us to maintain a more regular mesh during the deformation, which results in a better fit. We use the ground truth segmentation of the last volume to initialize as in Sect. 2.3. Furthermore, the max. tet. volume, s, starts at 4× the final value to perform a rough initial fit and is then set to the final value for the last 10 iterations to refine the fit.

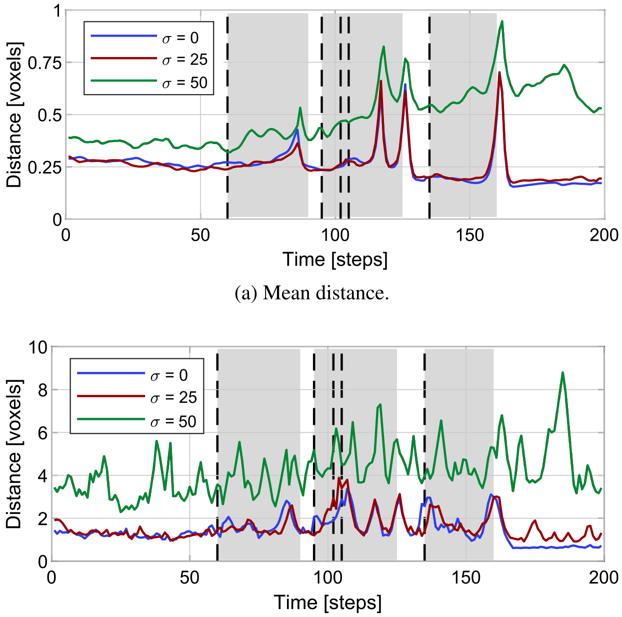
The parameters for all methods are shown in Table 1, and all experiments were performed with an AMD Ryzen 7 3700X processor. For the noiseless image, we only apply our method. The segmentation results for our method are shown in Fig. 8. Visually, the segmentations correspond well with the data, with some noise observable for $\sigma = 50$.

To provide a quantitative measure of the quality of the segmentation, a ground truth segmentation boundary—stored as a surface mesh—was created for each 3D volume. These were then compared with cross sections of the tetrahedral mesh at the corresponding times. The comparison was done with the mean and maximum distance between the surface meshes as computed with the MESH tool by Aspert et al. [45]. The results are shown in Fig. 9.

The plots support what is seen in Fig. 8 as the mean distances remain small, i.e., below one voxel. There is a spike in the error at the end of the merges, since there is a slight discontinuity in the data when blobs merge. Thus, the temporal part of the chosen regularization introduces a larger error at these times. The same behavior occurs for the maximum dis-

Table 1 Parameters used to segment the artificial 4D images

Method	Parameter	$\sigma = 0$	$\sigma = 25$	$\sigma = 50$
Ours	Means, c_{in}, c_{out}	100, 200	100, 200	100, 200
	#iter., N	50	50	50
	Step size, τ	0.2	0.2	0.3
	Smoothing, λ	0.004	0.004	0.004
	Max. volume, s	16	16	16
	n_{sub}	10	10	10
MRF 3D/4D	Source term	–	$(I(\mathbf{x})-200)^2$	$(I(\mathbf{x})-200)^2$
	Sink term	–	$(I(\mathbf{x})-100)^2$	$(I(\mathbf{x})-100)^2$
	Neighbor term	–	10^4	10^4
Graph cut (GC/Δ)	Means, c_{in}, c_{out}	–	100, 200	100, 200
	Std. dev., σ	–	100	100
	#Column samples	–	150	150
	Coumn. sample step	–	0.5	0.5
	Max. diff., Δ	–	2 and 8	2 and 8



(a) Mean distance.

(b) Max. distance.

Fig. 9 Plots of mean and max. distance between cross sections of the fitted tetrahedral meshes and ground truth segmentation boundary. The gray regions signify times when merges are ongoing and the vertical dotted lines mark the beginning of a new merge

tance, although the errors are larger since a single spurious vertex can significantly affect the value.

For the other methods, we also use the distance to the ground truth segmentation boundary. For the GC methods, this can be done directly. For the MRF methods, we extract an isosurface from each 3D label volume. The results are shown in Fig. 10. Additionally, we also compare the resource use of the methods. We measured runtime, peak memory use, and how much memory was needed to store the final segmentations. Table 2 shows the results along with the theoretical scaling of each measured value w.r.t. the image size. Our method performs as well as or better than the compared approaches while resulting in a more compact representation of the final segmentation.

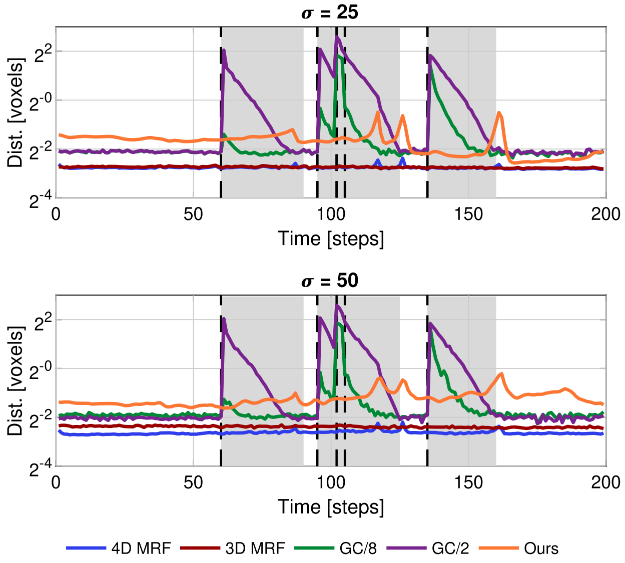


Fig. 10 Plots of the mean distance between the computed segmentation boundary for each method and ground truth segmentation boundary. The gray regions signify times when merges are ongoing and the vertical dotted lines mark the beginning of a new merge

3.2 Application to Metal Foam

We now apply the method to a 4D synchrotron X-ray CT image. The dataset is a time series of foaming metal, where blowing agent powders have been mixed into a block of aluminum. As the aluminum sample is heated to its melting point

Table 2 Resource use for the segmentation approaches. The memory uses do not include the size of the image data (200 MB). The scaling w.r.t. the image size of each value is also shown. S represents the size of each 3D volume, and T is the number of 3D volumes in the 4D image

Noise	Method	Time		Memory use (peak)		Memory use (seg.)	
		Meas.	Scaling	Meas.	Scaling	Meas	Scaling
$\sigma = 25$	3D MRF	45 s	S, T	408.3 MB	S	200.0 MB	S, T
	4D MRF	205 s	S, T	57,500.0 MB	S, T	200.0 MB	S, T
	GC/2	37 s	T	62.6 MB	1	9.2 MB	T
	GC/8	39 s	T	50.3 MB	1	9.2 MB	T
	Ours	36 s	1	40.5 MB	1	4.4 MB	1
$\sigma = 50$	3D MRF	45 s	S, T	421.5 MB	S	200.0 MB	S, T
	4D MRF	232 s	S, T	57,500.0 MB	S, T	200.0 MB	S, T
	GC/2	38 s	T	40.1 MB	1	9.2 MB	T
	GC/8	40 s	T	48.3 MB	1	9.2 MB	T
	Ours	38 s	1	41.0 MB	1	5.3 MB	1

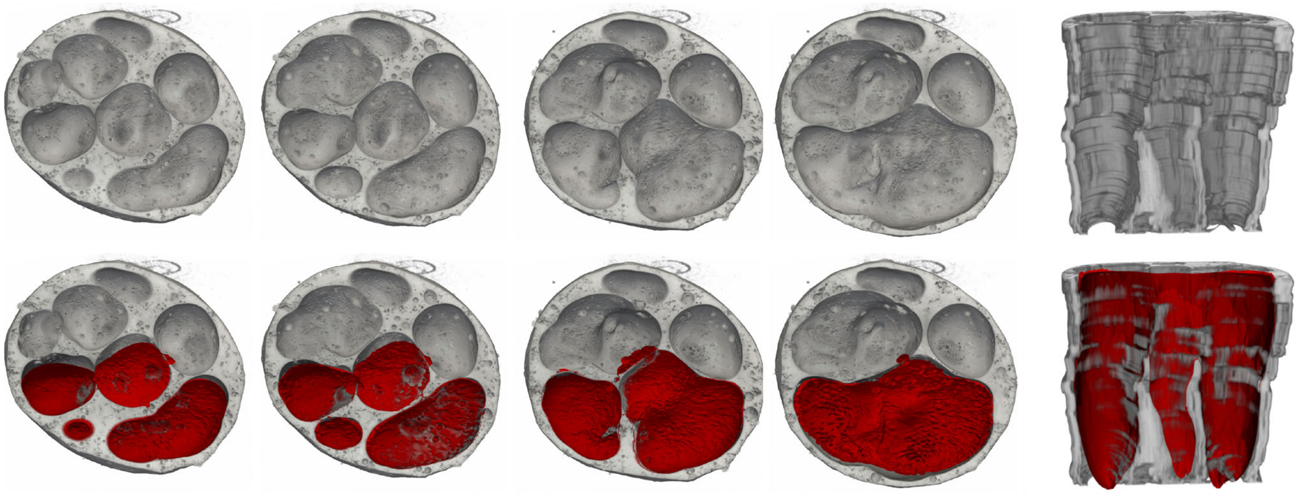

(a) xyz-volume at $t = 25$ (b) xyz-volume at $t = 35$ (c) xyz-volume at $t = 45$ (d) xyz-volume at $t = 55$ (e) xyt-volume at $x = 232$

Fig. 11 Segmentation results for the 4D synchrotron X-ray CT data. The top row shows a 3D rendering of the 4D data at different times. The bottom row shows the segmentations overlaid on the data. Note that the last column shows an xyt-slice where the vertical axis represents time (bottom = t_0 and top = t_{95})

the powders release gas, which causes bubbles to form. Over time, these bubbles expand and merge to form even larger bubbles. More details can be found in [2]. The data consists of 55 volumes of size 888 × 888 × 600 and uses 26 GB of memory. It is visualized in the top row of Fig. 11.

We initialize the mesh at the large bubble in the bottom of the last volume (see Fig. 11d). We again scale the spatial axes to $[-1, 1]$ and the temporal to $[-0.25, 0.25]$. The goal of the segmentation is to detect which bubbles merged together to form the final bubble. The parameters are shown in Table 3 and the segmentation itself took 5 min using an Intel Xeon Gold 6142 processor.

The result of the segmentation is shown in the bottom row of Fig. 11. The method has detected the merging of different bubbles. Furthermore, the 4D cross sections provide a good match with the data, even though the quality of the segmentation degrades somewhat the further we are from the initial mesh. Also, there are some minor errors in the last volumes where the tetrahedral mesh has moved into an adjacent bubble.

Table 3 Parameters used to segment the 4D synchrotron X-ray CT data. Please see Table 1 for more descriptive parameter names

c_{in}	c_{out}	N	τ	λ	s	n_{sub}
25	70	342	0.03	0.0003	$5.2 \cdot 10^{-6}$	10

3.3 Tracking Evolving Topology

We now demonstrate how our method allows the quantification of splitting/merging by computing the Reeb graph [11, 12] of the fitted tetrahedral mesh. Reeb graphs are used to describe the evolution of level sets of a function defined on

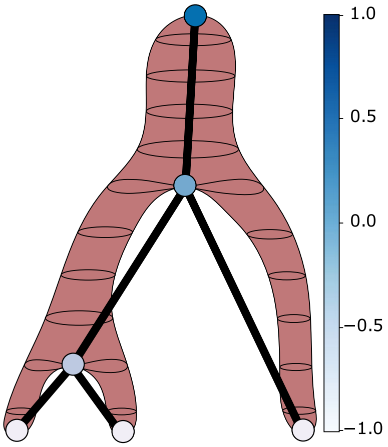


Fig. 12 Illustration of the Reeb graph for a simple 2D+time example. The level sets are shown as black rings and the Reeb graph has nodes where the level sets change topology. The nodes are colored according to their time coordinate

a manifold—in our case the time coordinate of our fitted tetrahedral mesh—making them ideal for detecting topological event such as splitting or merging [46]. As illustrated in Fig. 12, nodes are placed where topological changes occur such as local extrema and saddle points. If the level sets between two points form a connected component, the points are joined by an edge. As a result, the Reeb graph provides a compact description of the evolving topology of an object and also encodes when and where topological changes occur.

We use the Topology Toolkit [47, 48] to compute the Reeb graph for the (noiseless) segmentation of our 4D test image and the segmentation of the metal foam image. Prior to computation, we smooth the tetrahedral mesh with Laplacian smoothing and afterward we prune edges of the Reeb graph which are shorter than 10% of the mean edge length.

The results are shown in Fig. 13. For both segmentations, the Reeb graph provides a good description of where and how the objects merged together. Note especially how, even though a level set might have some slight self-intersections, e.g., in Fig. 13h, objects are not counted as having merged since the Reeb graph considers the full spatiotemporal structure.

4 Discussion

The method presented in this paper can segment merging or splitting objects in 4D. As demonstrated in Sect. 3.1, this can be achieved even in the presence of severe noise, although with a reduced quality. When objects are merging, our method also significantly outperformed the fixed-topology GC methods while being faster and using the same amount of memory. The MRF methods gave the most accurate segmentation, which is expected given that this is a binary segmentation problem with Gaussian noise. However, they used significantly more time and memory.

Furthermore, the time and memory benefits of our method will only improve as the image size increases, since the resource use of our method only depends on the complexity of the object to segment and *not* on the size of the input image. This is not the case for the compared methods. Finally, storing the segmentation as a tetrahedral mesh also results in an extremely compact description of the segmented objects—indeed, of the tested methods, ours provided the most compact segmentation. For the foaming aluminum data, which took up 26 GB of space, the resulting mesh only uses around 8.5 MB which is a reduction factor of more than 3,000.

Our method also offers increased robustness regarding quantifying the evolution of object topology. As we explicitly model the hypersurface boundary in 4D, splits/merges can be automatically detected as saddle points and do not need manual heuristics or collision detection like sequential methods. Furthermore, this also means that if some level sets intersect due to segmentation errors or imaging artifacts it is easily discarded as an error and not a merge/split—something which is not possible with sequential or 3D/4D voxel-based approaches. These benefits are unique to our object representation.

The main challenge with our method is how to maintain the mesh quality during deformation. With too high a regularization, the method cannot achieve a sufficient quality of fit due to over-smoothing. With too low, the tetrahedral mesh may start to self-intersect which results in a degenerate fit. Currently, finding the suitable balance between step size and regularization strength requires manual tuning. Promising avenues for addressing this are collision detection or displacing the vertices along non-intersecting search lines as in [49, 50]. However, as implementing these in a computationally efficient way is non-trivial, we relegate it to future work.

Furthermore, while Laplacian smoothing works well to regularize the spatial components of the fit, the spatiotemporal saddle points, that form at splits/merges (see Fig. 11e), are sensitive to over-smoothing. Additionally, for more complex image data (*e.g.* medical scans), an intensity-based deformation force may not be adequate. In these cases, performance may be improved by adapting approaches from [51–54] where the output of a neural network is used to guide the deformation of a contour in 2D images. Finally, our method does not attempt to detect if the topological assumptions are broken. As a result, fitting to image data where both splits and merges happen can result in degenerate fits.

5 Conclusion

We presented a method for simultaneously segmenting and tracking merging or splitting objects in 4D images. This

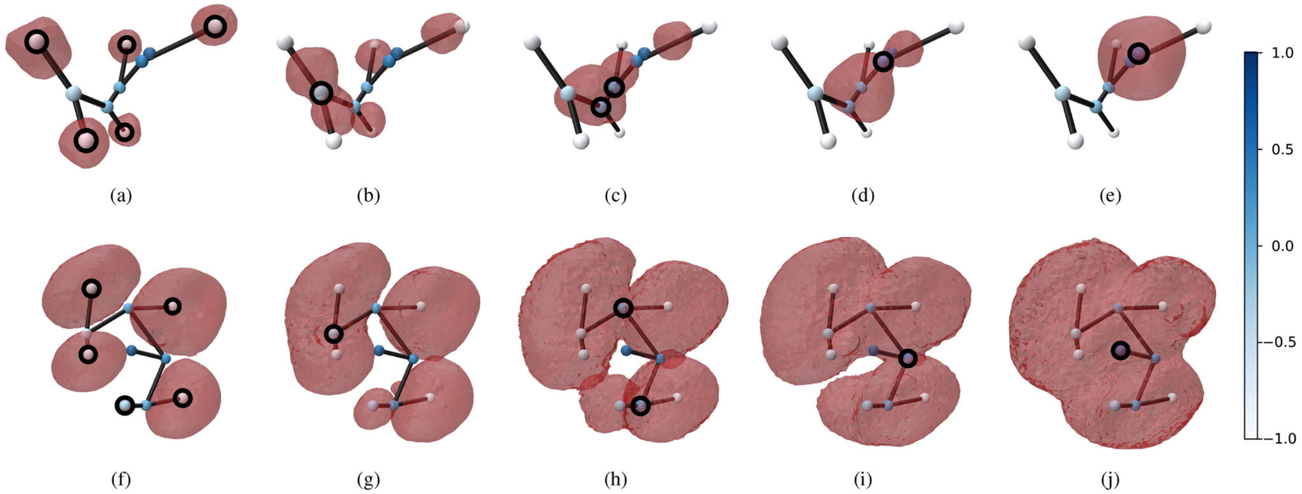


Fig. 13 Reeb graphs for the artificial (top row) and metal foam (bottom row) images. Each column shows a 3D cross section of the fitted 4D tetrahedral mesh with the Reeb graph overlaid. The Reeb graph has been projected to 3D and its nodes are colored according to their time coordinate (scaled to be in $[-1, 1]$) as in Fig. 12. The Reeb graph nodes corresponding to the merge in each subfigure have been highlighted with a black circle

was achieved by fitting a discretized hypersurface to the 3D boundary of the 4D simply connected component comprising the object of interest. Applications to artificial and real 4D images showed that the method was able to successfully segment, track, and quantify merging or splitting objects, even in the presence of noise. The method achieved comparable accuracy to existing methods with a resource use equal to or better than sequential 3D methods. Furthermore, our method only scales with object complexity and not image size making it suitable for the analysis of very large images.

Acknowledgements We want to thank Rajmund Mokso and Francisco García-Moreno for providing the 4D image of foaming metal.

Author Contributions All authors contributed to the method conception and design. Method implementation, experiments, and analysis were performed by PMJ. The first draft of the manuscript was written by PMJ. All authors commented on previous versions of the manuscript and suggested improvements. All authors read and approved the final manuscript.

Funding Open access funding provided by Technical University of Denmark This work was partly supported by a research grant (VIL50425) from VILLUM FONDEN.

Data Availability Source code for a Python implementation of the method can be found at https://github.com/patmjen/4dsurf. This also includes the artificial 4D image used for the experiments in Sect. 3.1. For the 4D image of foaming metal used in Sect. 3.2, please contact the authors of [2].

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Peng, P., Lekadir, K., Gooya, A., Shao, L., Petersen, S.E., Frangi, A.F.: A review of heart chamber segmentation for structural and functional analysis using cardiac magnetic resonance imaging. Magn. Reson. Mater. Phys., Biol. Med. **29**(2), 155–195 (2016)
2. García-Moreno, F., Kamm, P.H., Neu, T.R., Bülk, F., Mokso, R., Schlepütz, C.M., Stampanoni, M., Banhart, J.: Using x-ray tomoscopy to explore the dynamics of foaming metal. Nat. Commun. **10**(1), 1–9 (2019)
3. Maire, E., Withers, P.J.: Quantitative x-ray tomography. Int. Mater. Rev. **59**(1), 1–43 (2014)
4. Raufaste, C., Dollet, B., Mader, K., Santucci, S., Mokso, R.: Three-dimensional foam flow resolved by fast x-ray tomographic microscopy. Europhys. Lett. **111**(3), 38004 (2015)
5. Mikula, K., Peyriéeras, N., Remešíková, M., Smíšek, M.: 4D numerical schemes for cell image segmentation and tracking. In: Finite Volumes for Complex Applications VI Problems & Perspectives, pp. 693–701 (2011)
6. García-Moreno, F., Kamm, P.H., Neu, T.R., Bülk, F., Noack, M.A., Wegener, M., von der Eltz, N., Schlepütz, C.M., Stampanoni, M., Banhart, J.: Tomoscopy: time-resolved tomography for dynamic processes in materials. Adv. Mater. **33**(45), 2104659 (2021)

7. Mokso, R., Schlepütz, C.M., Theidel, G., Billich, H., Schmid, E., Celcer, T., Mikuljan, G., Sala, L., Marone, F., Schlumpf, N., et al.: GigaFRoST: the gigabit fast readout system for tomography. J. Synchrotron Radiat. **24**(6), 1250–1259 (2017)
8. Kass, M., Witkin, A., Terzopoulos, D.: Snakes: active contour models. Int. J. Comput. Vis. (IJCV) **1**(4), 321–331 (1988)
9. Cohen, I., Cohen, L.D., Ayache, N.: Using deformable surfaces to segment 3-D images and infer differential structures. Comput. Vis. Graph. and Image Process. **56**(2), 242–263 (1992)
10. McInerney, T., Terzopoulos, D.: A dynamic finite element surface model for segmentation and tracking in multidimensional medical images with application to cardiac 4d image analysis. Comput. Med. Imaging Graph. **19**(1), 69–83 (1995)
11. Shinagawa, Y., Kunii, T.L., Kergosien, Y.L.: Surface coding based on Morse theory. IEEE Comput. Graph. Appl. **11**(05), 66–78 (1991)
12. Reeb, G.: Sur les points singuliers d'une forme de pfaff completement integrable ou d'une fonction numerique [on the singular points of a completely integrable pfaff form or of a numerical function]. C. R. Acad. Sci. Paris **222**, 847–849 (1946)
13. Gao, Y., Phillips, J.M., Zheng, Y., Min, R., Fletcher, P.T., Gerig, G.: Fully convolutional structured LSTM networks for joint 4D medical image segmentation. In: IEEE International Symposium on Biomedical Imaging, pp. 1104–1108 (2018). IEEE
14. Benninghoff, H., Garcke, H.: Segmentation of three-dimensional images with parametric active surfaces and topology changes. J. Sci. Comput. **72**, 1333–1367 (2017)
15. Foteinos, P., Chrisochoides, N.: 4D space-time Delaunay meshing for medical images. Eng. Comput. **31**, 499–511 (2015)
16. Kashyap, S., Zhang, H., Sonka, M.: Accurate fully automated 4D segmentation of osteoarthritic knee MRI. Osteoarthr. Cartil. **25**, 227–228 (2017)
17. Montagnat, J., Delingette, H.: Space and time shape constrained deformable surfaces for 4D medical image segmentation. In: International Conference on Medical Image Computing and Computer-Assisted Intervention, pp. 196–205 (2000). Springer
18. Oguz, I., Abramoff, M.D., Zhang, L., Lee, K., Zhang, E.Z., Sonka, M.: 4D graph-based segmentation for reproducible and sensitive choroid quantification from longitudinal oct scans. Investig. Ophthalmol. Vis. Sci. **57**(9), 621–630 (2016)
19. Wu, X., Chen, D.Z.: Optimal net surface problems with applications. In: International Colloquium on Automata, Languages, and Programming, pp. 1029–1042 (2002). Springer
20. Wang, Y., Zhang, Y., Xuan, W., Kao, E., Cao, P., Tian, B., Ordovas, K., Saloner, D., Liu, J.: Fully automatic segmentation of 4D MRI for cardiac functional measurements. Med. Phys. **46**(1), 180–189 (2019)
21. Lorenzo-Valdés, M., Sanchez-Ortiz, G.I., Elkington, A.G., Mohiaddin, R.H., Rueckert, D.: Segmentation of 4d cardiac MR images using a probabilistic atlas and the EM algorithm. Med. Image Anal. **8**(3), 255–265 (2004)
22. Wolz, R., Heckemann, R.A., Aljabar, P., Hajnal, J.V., Hammers, A., Lötjönen, J., Rueckert, D., Initiative, A.D.N., et al.: Measurement of hippocampal atrophy using 4D graph-cut segmentation: application to ADNI. Neuroimage **52**(1), 109–118 (2010)
23. Choy, C., Gwak, J., Savarese, S.: 4D spatio-temporal convnets: Minkowski convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3075–3084 (2019)
24. Chan, T.F., Vese, L.A.: Active contours without edges. IEEE Trans. Image Process. **10**(2), 266–277 (2001)
25. Guillemin, V., Pollack, A.: Differential Topology, vol. 370. American Mathematical Society, Providence (2010)
26. Desbrun, M., Meyer, M., Schröder, P., Barr, A.H.: Implicit fairing of irregular meshes using diffusion and curvature flow. In: Proceedings of SIGGRAPH, pp. 317–324 (1999)
27. Dahl, V.A., Dahl, A.B., Hansen, P.C.: Computing segmentations directly from x-ray projection data via parametric deformable curves. Meas. Sci. Technol. **29**(1), 014003 (2017)
28. Fujiwara, K.: Eigenvalues of Laplacians on a closed Riemannian manifold and its nets. Proc. Am. Math. Soc. **123**, 2585–2594 (1995)
29. Egger, J., Bauer, M.H., Kuhnt, D., Carl, B., Kappus, C., Freisleben, B., Nimsky, C.: Nugget-cut: a segmentation scheme for spherically- and elliptically-shaped 3D objects. In: Joint Pattern Recognition Symposium, pp. 373–382 (2010). Springer
30. Wang, Y., Beichel, R.: Graph-based segmentation of lymph nodes in CT data. In: International Symposium on Visual Computing, pp. 312–321 (2010). Springer
31. Lorensen, W.E., Cline, H.E.: Marching cubes: a high resolution 3D surface construction algorithm. In: Proceedings of SIGGRAPH, vol. 21, pp. 163–169 (1987). ACM
32. Si, H.: TetGen, a Delaunay-based quality tetrahedral mesh generator. ACM Trans. Math. Softw. **41**(2), 11 (2015)
33. Lee, J.M., Chow, B., Chu, S.-C., Glickenstein, D., Guenther, C., Isenberg, J., Ivey, T., Knopf, D., Lu, P., Luo, F., et al.: Manifolds and differential geometry. Topology **643**, 658 (2009)
34. Axler, S.J.: Linear Algebra Done Right, vol. 2. Springer, New York (1997)
35. Horn, R.A., Johnson, C.R.: Matrix Analysis. Cambridge University Press, Cambridge (2012)
36. Chu, A., Fu, C.-W., Hanson, A., Heng, P.-A.: GL4D: a GPU-based architecture for interactive 4D visualization. IEEE Trans. Visual Comput. Graph. **15**(6), 1587–1594 (2009)
37. Southern, R.: Animation manifolds for representing topological alteration. Technical report, University of Cambridge, Computer Laboratory (2008)
38. Bærentzen, J.A., Aanaes, H.: Signed distance computation using the angle weighted pseudonormal. IEEE Trans. Vis. Comput. Graph. **11**(3), 243–253 (2005)
39. Todhunter, I.: Spherical Trigonometry, for the Use of Colleges and Schools: With Numerous Examples. Macmillan, Cambridge (1863)
40. Zhou, Y., Chen, W., Tang, Z.: An elaborate ambiguity detection method for constructing isosurfaces within tetrahedral meshes. Comput. Graph. **19**(3), 355–364 (1995)
41. Chen, X., Pan, L.: A survey of graph cuts/graph search based medical image segmentation. IEEE Rev. Biomed. Eng. **11**, 112–124 (2018)
42. Boykov, Y., Kolmogorov, V.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. IEEE Trans. Pattern Anal. Mach. Intell. (PAMI) **26**(9), 1124–1137 (2004)
43. Li, K., Wu, X., Chen, D.Z., Sonka, M.: Optimal surface segmentation in volumetric images-a graph-theoretic approach. IEEE Trans. Pattern Anal. Mach. Intell. (PAMI) **28**(1), 119–134 (2005)
44. Haeker, M., Wu, X., Abràmoff, M., Kardon, R., Sonka, M.: Incorporation of regional information in optimal 3-D graph search with application for intraretinal layer segmentation of optical coherence tomography images. In: Biennial International Conference on Information Processing in Medical Imaging, pp. 607–618 (2007). Springer
45. Aspert, N., Santa-Cruz, D., Ebrahimi, T.: Mesh: Measuring errors between surfaces using the Hausdorff distance. In: Proceedings of the IEEE International Conference on Multimedia and Expo, vol. 1, pp. 705–708 (2002). IEEE
46. Weber, G., Bremer, P.-T., Day, M., Bell, J., Pascucci, V.: Feature tracking using Reeb graphs. In: Topological Methods in Data Analysis and Visualization, pp. 241–253 (2011)
47. Masood, T.B., Budin, J., Falk, M., Favelier, G., Garth, C., Gueunet, C., Guillou, P., Hofmann, L., Hristov, P., Kamakshidasan, A., et al.: An overview of the topology toolkit. In: Proceedings of Topological Methods in Data Analysis and Visualization (2019)

48. Tierny, J., Favelier, G., Levine, J.A., Gueunet, C., Michaux, M.: The topology toolkit. IEEE Trans. Visual Comput. Graph. **24**(1), 832–842 (2017)
49. Yin, Y., Song, Q., Sonka, M.: Electric field theory motivated graph construction for optimal medical image segmentation. In: International Workshop on Graph-Based Representations in Pattern Recognition, pp. 334–342 (2009)
50. Yin, Y., Sonka, M.: Electric field theory based approach to search-direction line definition in image segmentation: application to optimal femur-tibia cartilage segmentation in knee-joint 3-D MR. In: Medical Imaging, vol. 7623, pp. 309–317 (2010)
51. Acuna, D., Ling, H., Kar, A., Fidler, S.: Efficient interactive annotation of segmentation datasets with polygon-RNN++. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 859–868 (2018)
52. Castrejon, L., Kundu, K., Urtasun, R., Fidler, S.: Annotating object instances with a polygon-RNN. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 5230–5238 (2017)
53. Ling, H., Gao, J., Kar, A., Chen, W., Fidler, S.: Fast interactive object annotation with curve-GCN. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 5257–5266 (2019)
54. Peng, S., Jiang, W., Pi, H., Li, X., Bao, H., Zhou, X.: Deep snake for real-time instance segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 8533–8542 (2020)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Patrick M. Jensen is a postdoc at the Department of Applied Mathematics and Computer Science (DTU Compute) at the Technical University of Denmark and a guest scientist at the Institute of Mathematics and Image Computing at the University of Lübeck. His research interests are segmentation and quantitative analysis methods for large-scale 3D images with a focus on materials science and pre-clinical medical data.

J. Andreas Bærentzen is Professor of Computer Graphics at the Department of Applied Mathematics and Computer Science (DTU Compute) at the Technical University of Denmark from which he received his M.Sc. and Ph.D. degrees in 1998 and 2003, respectively. Andreas Bærentzen conducts research in the area of computer graphics with emphasis on geometric modeling. His specific interests include surface reconstruction, skeletonization, methods for interactive manipulation of digital shapes, methods for synthesis of 3D geometry in the context of shape and topology optimization, and generative methods for geometry synthesis. Andreas is also interested in many aspects of real-time graphics.

Anders B. Dahl is professor in 3D image analysis, and a head of the Section for Visual Computing at the Department of Applied Mathematics and Computer Science, Technical University of Denmark (DTU), Kgs. Lyngby, Denmark. He is heading The Center for Quantification of Imaging Data from MAX IV, focusing on quantitative analysis of 3D images. His research is focused on image segmentation and its applications.

Vedrana A. Dahl is an associate professor at the Department of Applied Mathematics and Computer Science, Technical University of Denmark (DTU), Kgs. Lyngby, Denmark. Her primary research interest is in the use of geometric models for the analysis of volumetric data. This includes volumetric segmentation and methods based on deformable meshes. She developed analysis tools with applications in material science, industrial inspection, and biomedicine.