Journal of Real-Time Image Processing
https://doi.org/10.1007/s11554-017-0740-1

ORIGINAL RESEARCH PAPER

Interactive GPU active contours for segmenting inhomogeneous objects

Chris G. Willcocks[1] · Philip T. G. Jackson[1] · Carl J. Nelson[1] · Amar V. Nasrulloh[1] · Boguslaw Obara[1]

Received: 13 April 2017 / Accepted: 27 November 2017

Abstract
We present a segmentation software package primarily targeting medical and biological applications, with a high level of visual feedback and several usability enhancements over existing packages. Specifically, we provide a substantially faster GPU implementation of the local Gaussian distribution fitting energy model, which can segment inhomogeneous objects with poorly defined boundaries as often encountered in biomedical images. We also provide interactive brushes to guide the segmentation process in a semiautomated framework. The speed of our implementation allows us to visualize the active surface in real time with a built-in ray tracer, where users may halt evolution at any time step to correct implausible segmentation by painting new blocking regions or new seeds. Quantitative and qualitative validation is presented, demonstrating the practical efficacy of our interactive elements for a variety of real-world datasets.

Keywords Segmentation · Active contours · Level set methods · GPU · Medical applications · Biological applications

1 Introduction

Image segmentation is a large research field with many practical applications, including but not limited to:

- Biosciences:
  - Cellular, developmental and cancer biology.
  - Plant biology, including plant–pathogen interactions.
  - Animal biology, including virus–host interactions and bacterial infections.
  - Microbiology, including food safety.
  - Neuroscience, including connectome projects and developmental neuroscience.
- Medicine:
  - Automated differential diagnosis.
  - Diagnostic measurements, shape, and volume, of:
    - Macular holes in retinal degeneration.
    - Aneurysms, clotting and infarction.
    - Tumors, neoplasia and dermatological moles.
    - MRI segmentation in dementia and Alzheimer's.
  - Computer-assisted surgery:
    - Pre-surgical planning and surgery simulation.
    - Guided surgical navigation.

The primary problems with current segmentation approaches are that they are either: (1) too limited, e.g., only able to segment objects by simple criteria, such as objects with consistent mean intensity [18, 36], (2) using too much memory or too slow, taking several hours to segment large 2D or 3D objects [47], (3) lacking in interactivity with the segmentation process in response to visual feedback [54], (4) requiring copious training data [22], or (5) difficult to use, requiring large interfaces, and multiple algorithms [50].

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s11554-017-0740-1) contains supplementary material, which is available to authorized users.

✉ Boguslaw Obara
boguslaw.obara@durham.ac.uk

Chris G. Willcocks
christopher.g.willcocks@durham.ac.uk

Philip T. G. Jackson
p.t.g.jackson@durham.ac.uk

Amar V. Nasrulloh
amar.v.nasrulloh@durham.ac.uk

[1] Department of Computer Science, Durham University, South Road, Durham DH1 3LE, UK

Published online: 26 December 2017

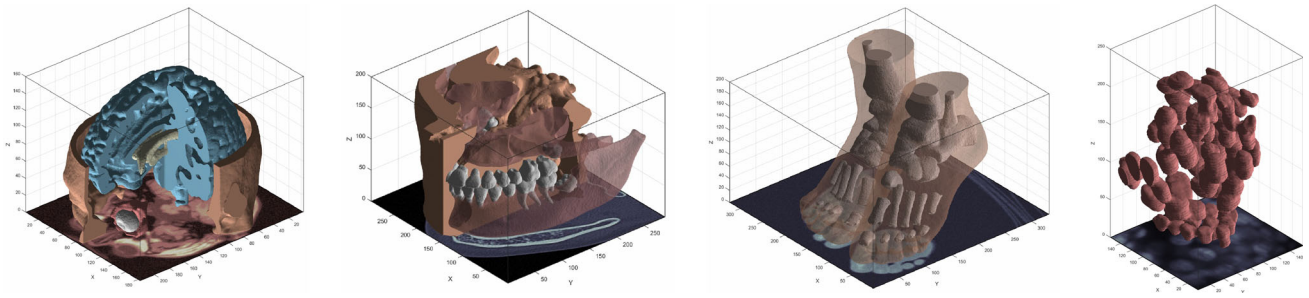


Fig. 1 A selection of 3D objects segmented by our software. Our interactive method allows users to efficiently capture specific objects (colored separately) within the data, such as the teeth in (**b**). Image **a** is a simulated brain MRI [8], images **b**, **c** are CT scans [37], and **d** shows selective plane illumination microscopy (SPIM) of zebrafish eye lens cells [17]. **a** Brain and ventricles. **b** Dental scan. **c** Foot bones and tissues. **d** Zebrafish cells

The oldest and most widely cited segmentation approaches are active contours [20]; these are variational frameworks which allow users to define an initial open or closed curve that deforms so as to minimize a energy functional, outlining or surrounding the object of interest. While active contours have been realized as fully automatic approaches without initial contours [25], their original foundation as an assisted approach is still important today as it allows users, such as clinicians, to extract precise measurements from specific objects of interest within a complex image. However, such interactivity relies on real-time visual feedback; therefore, they must also be computationally efficient.

Graphics processing units (GPUs) provide energy-efficient parallel computing and enable real-time interactive segmentation for larger 2D or 3D datasets [10, 43], but existing GPU segmentation methods currently rely on simple segmentation criteria restricting their usage and applications. The popular local Gaussian distribution fitting (LGDF) energy model [47] is much more powerful and able to segment a wider variety of general objects. However, it requires several intermediate processing steps that must be implemented sequentially, making it challenging to efficiently implement on graphics hardware. The current implementation of the LGDF energy model can segment small 2D images (99×120 in 27.37 s), but requires several hours of processing for larger 2D or 3D images [47]. For a 3D image of size $256 \times 256 \times 160$, this would take 6.6 hours if the implementation were available for 3D, preventing usage in many practical applications.

1.1 Contributions

In our approach, we: (1) significantly increase the performance of the LGDF energy model through an optimized GPU implementation, handling much larger 2D images and even 3D images at interactive performance, (2) introduce a novel set of interactive brush functions that are integrated into the GPU kernels such as to modify and constrain the evolving level set in real time, (3) provide a ray tracer to view the segmentation results at each time step, and (4) expose a simpler and more intuitive parameter space to the user, with suggested values and ranges. The combination of these four enhancements greatly improves the practicality of what is already considered a state-of-the-art level set method of particular relevance to the biomedical image processing communities. Our software is shown to be stable with respect to its input parameters and robust to noise through a large experiment on synthetic data and is further evaluated through segmenting a wide variety of real-world images, such as those shown in Fig. 1.

2 Related work

The field of active contours first gained mainstream adoption with the ‘active snakes’ model published by [20]. This seminal work proposes iterative evolution of an initial spline curve, with the evolution being governed by the minimization of an energy functional, the local minima of which correspond to curves that fit along prominent edges in the image. Level set methods (core theory explained in [30]) model contours implicitly as the zero-crossing of a scalar field. Originally they were proposed in [31] to model the evolution of inter-region boundaries in physical simulations. Malladi et al. [26] applied level sets to active contours, with the evolution of the contour being governed by its local mean curvature and the intensity gradient magnitude of the image, in such a way that local curvature is reduced and the motion of the contour stops as it approaches an edge. In [5], the authors develop a level set-based active contour framework in which the energy functional is based on the Mumford–Shah model, rather than image edges, which in practice are often faint, blurred,

or broken. The Mumford–Shah energy model [28] is minimized by an optimal partition of an image into piecewise smooth segments, and high-quality implementations exist on the GPU [33]. The global optimum can be found using a primal-dual algorithm [4] resulting in a cartoon-like rendering of the original image. Local solutions, such as with a trust-region approach [14], have applications in interactive segmentation, where local edits need to be made frequently.

Deep convolutional neural networks are the state of the art in image segmentation, where millions of parameters of deeply layered convolutions are learned using backpropagation [22]. These models are capable of learning abstract features in the data; however, their current reliance on such large datasets makes them unusable for a number of applications.

The influential public datasets with ground-truth segmentations (such as BSDS, MSRC, iCoseg, FlickrMFC, SegTrack) include RGB videos or 2D images such as cars, chairs, and people. Of these, the interactive approaches take as input a set of scribbles where objects follow similar color distributions [53]. Graph cut segmentation is popular in this field, where Grady [15] and Vineet and Narayanan [46] propose GPU implementations. For interactive segmentation in the biosciences, we find the main limitations being (1) the initialization of the foreground–background scribbles in 3D datasets such as networks and (2) the opaque intermediate steps of the cutting algorithm making it difficult to obtain a high level of visual feedback. While popular and easy to validate, these approaches address a different problem to grayscale 3D segmentation as with imaging modalities (such as CT, PET, SPECT, MRI, fMRI, ultrasound, optical imaging and microscopy) in the biosciences [10]. There is still a need for benchmark medical datasets with well-defined interactive performance evaluation [51].

Accelerating image segmentation with GPUs is a large research field with several comprehensive surveys [10, 34, 41, 43]. The survey by [10] covers a broad range of algorithms and different imaging modalities, whereas Smistad et al. [43] focuses more on GPU segmentation with a detailed discussion on the current GPU architecture.

The GPU level set methods in the literature focus on limiting the active computational domain to a small region near the zero-crossing of the level set function, such as the traditional narrow band algorithm [1]. More recent extensions classify the active region using simple operations on the spatial and temporal derivatives of the level set function [36] and then discard unimportant regions through parallel stream compaction. While limiting the active computational domain produces excellent performance with lower memory usage, the current implementations all use simple speed functions that attract the level set to make it grow and/or shrink within a fixed intensity range [18, 23, 36]. In contrast, the LGDF model proposed by [47] is able to segment much more challenging images, in which objects exhibit intensity inhomogeneity or even have the same mean intensity as their background, being distinguished only by intensity variance. However, to date the only existing implementation runs on the CPU, likely due to the sequential dependency of convolutions in the intermediate steps. Further, the LGDF model is derived from [5] who introduce C^∞ regularization of the Heaviside and Dirac functions which are nonzero everywhere, unlike the C^2 regularized Heaviside (proposed in [52]) which is nonzero only in the vicinity of the contour. C^∞ regularization restrains the algorithm from converging on local minima, but precludes traditional narrow band or sparse field algorithms because it requires the level set to update at all points on each time step.

GPU active contour methods parallelize the calculation of the energy forces described in the original snakes paper [20]. Traditional methods rely on simple intensity gradients and are prone to converging on local minima; however, [49] introduced a diffusion of the gradient vectors called gradient vector flow (GVF) to address this problem. [16] were one of the first GPU active contour implementations using GVF, and more recent optimizations in OpenCL exploit cached texture memory which has spatial locality in multiple dimensions [42]. The active contour can also be approximated by a surface mesh, such as in [39] who use Laplacian smoothing on local neighborhoods in conjunction with driving mesh vertices with gradient and intensity forces. However, these approaches still rely on the image gradient being a reliable indication of object boundaries, which is not the case in many real-world images [5].

Ever since the original snakes paper, active contours have gained popularity through being able to interactively edit the contour, or set up constraints to guide its motion [20]. Region-based active contour methods provide the option to initialize with a simple primitive shape, or sketch a starting region [7]. The more advanced approach by [27] introduces non-Euclidean radial-basis functions, which are weighted by the image features and blended to form an implicit function whose sign can be fixed at user-defined control points. The tool by [50] provides an interactive interface with geodesic active contours [3] and region competition [55]. Region competition favors a well-defined intensity range, whereas the geodesic approach is better suited for images with clear edges; by combining both approaches, [50] can segment a broad range of images, yet it requires significant tuning and can still fail in complex images with neither a well-defined intensity range nor clear edges.

There are several GPU approaches that produce segmentation without relying on initialization of a seed region [25]. Clustering methods join regions of a high-dimensional feature space [13], and superpixel approaches [35] form clusters that are deliberately over-segmented into more manageable regions. These approaches are good at simplifying complex images, yet they do not capture specific objects. In contrast, active shape and appearance methods fit a model to the data based on prior knowledge; however, this inherently makes assumptions of the overall shape of the objects and fails when these assumptions are not met.

3 Method

The LGDF model, originally proposed in [47], builds on existing active contour literature by introducing a new energy functional based on the local Gaussian distributions of image intensity. This functional drives a variational level set approach which is able to segment objects whose intensity mean and variance are inhomogeneous. Rather than creating segments whose intensity is as uniform as possible, this algorithm allows slow changes in intensity across an object, penalizing only sudden changes within it, without relying on a gradient based edge detector [5].

The segmentation is represented by a level set function $\phi(\mathbf{x})$. The foreground region is the set of points $\{\mathbf{x} : \phi(\mathbf{x}) < 0\}$, and the exterior (or background) is $\{\mathbf{x} : \phi(\mathbf{x}) \geq 0\}$. The contour itself (or surface in 3D) is thus defined implicitly as the zero level set, $\{\mathbf{x} : \phi(\mathbf{x}) = 0\}$. Segmentation is achieved by minimizing a global energy functional:

$$E = E^{\mathrm{LGDF}}(I, \phi) + \mu \mathcal{P}(\phi) + \nu \mathcal{L}(\phi) \tag{1}$$

where $\mu, \nu > 0$ are weighting constants, E^{LGDF} is the LGDF energy term which drives the contour to fit along salient image edges, $\mathcal{P}$ avoids the need to periodically re-initialize ϕ to a signed distance function [24], and $\mathcal{L}$ penalizes the contour length to ensure smoothness. The E^{LGDF} term is the sum of the individual LGDF energies for each pixel $\mathbf{x}$:

$$\begin{aligned} E^{\mathrm{LGDF}}(I, \phi, \mathbf{x}) = &- \int_{\Omega} \omega(\mathbf{y} - \mathbf{x}) \log(p_{1,\mathbf{x}}(I(\mathbf{y}))) M_1(\mathbf{y}) \, \mathrm{d}\mathbf{y} \\ &- \int_{\Omega} \omega(\mathbf{y} - \mathbf{x}) \log(p_{2,\mathbf{x}}(I(\mathbf{y}))) M_2(\mathbf{y}) \, \mathrm{d}\mathbf{y} \end{aligned} \tag{2}$$

where $\omega(\mathbf{y} - \mathbf{x})$ is a Gaussian weighting function centered on $\mathbf{x}$, $p_{1,\mathbf{x}}$ is a Gaussian approximation of the intensity distribution for the part of the neighborhood of $\mathbf{x}$ lying outside the contour (and inside for $p_{2,\mathbf{x}}$), and M_1 equals one outside the contour, zero inside (vice-versa for M_2). This quantity is smaller when the intensity distributions in the parts of the neighborhood of $\mathbf{x}$ lying outside and inside the contour are well approximated as Gaussian distributions, which can only be achieved by deforming the contour so that it separates regions of different intensity mean and variance.

The mean and variance parameters for these local Gaussian distributions are denoted $u_i(\mathbf{x})$, $\sigma_i(\mathbf{x})$ where $i \in \{1, 2\}$ for regions outside and inside the contour, respectively:

$$u_i(\mathbf{x}) = \frac{\int \omega(\mathbf{y} - \mathbf{x}) I(\mathbf{y}) M_i(\phi(\mathbf{y})) \, \mathrm{d}\mathbf{y}}{\int \omega(\mathbf{y} - \mathbf{x}) M_i(\phi(\mathbf{y})) \, \mathrm{d}\mathbf{y}} \tag{3}$$

$$\sigma_i(\mathbf{x})^2 = \frac{\int \omega(\mathbf{y} - \mathbf{x}) (u_i(\mathbf{x}) - I(\mathbf{y}))^2 M_i(\phi(\mathbf{y})) \, \mathrm{d}\mathbf{y}}{\int \omega(\mathbf{y} - \mathbf{x}) M_i(\phi(\mathbf{y})) \, \mathrm{d}\mathbf{y}} \tag{4}$$

Specifically, they express for each pixel the mean and variance of neighboring gray values that lie outside and inside the contour (for pixels whose entire neighborhood lies on one side of the contour, only one pair of these values is defined). The size of each pixel's neighborhood is determined by the standard deviation of the Gaussian weighting function, ω. This is a user-defined parameter, denoted σ. A larger neighborhood increases the range from which a pixel may influence the contour. This results in faster evolution, greater capture range, and a greater tendency to produce segments whose boundaries separate large regions of different mean intensity.

The internal energy term $\mathcal{P}$ penalizes the contour's deviation from a signed distance function [24] to ensure numerical stability [32]:

$$\mathcal{P}(\phi) = \int_{\Omega} \frac{1}{2} (|\nabla \phi(\mathbf{x})| - 1)^2 \, \mathrm{d}\mathbf{x} \tag{5}$$

and $\mathcal{L}$ penalizes the contour length to ensure smoothness:

$$\mathcal{L}(\phi) = \int_{\Omega} |\nabla H(\phi(\mathbf{x}))| \, \mathrm{d}\mathbf{x} \tag{6}$$

where H is the C^{∞} regularized Heaviside function, discretized to operate on a regular grid, first proposed by [5]:

$$H(x) = \frac{1}{2} \left[1 + \frac{2}{\pi} \arctan(\mathrm{x}) \right] \tag{7}$$

The total energy functional (Eq. 1) can be minimized by applying the calculus of variations [47] yielding the following PDE:

$$\frac{\partial \phi}{\partial t} = -\delta(\phi)(\lambda_1 e_1 - \lambda_2 e_2) + \mu\left(\nabla^2 \phi - \kappa\right) + \nu \delta(\phi) \kappa \tag{8}$$

where δ is the regularized Dirac function $\delta(x) = H'(x)$ [5], λ_1, λ_2, ν and μ are parameters controlling the weight of the terms, and κ is the contour's local curvature [31]:

$$\kappa = \operatorname{div}\left(\frac{\nabla\phi}{|\nabla\phi|}\right) \tag{9}$$

and $-\delta(\phi)(\lambda_1 e_1 - \lambda_2 e_2)$ is the force due to E^{LGDF}:

$$e_i(\mathbf{x}) = \int_\Omega \omega(\mathbf{y}-\mathbf{x})\left[\log(\sigma_i(\mathbf{y})) + \frac{(\mathrm{u_i}(\mathbf{y}) - \mathrm{I}(\mathbf{x}))^2}{2\sigma_i(\mathbf{y})^2}\right]\mathrm{d}\mathbf{y} \tag{10}$$

The data fitting term $e_1(\mathbf{x})$ quantifies how badly the pixel $\mathbf{x}$ would fit with the outside-contour parts of its neighbors' neighborhoods. When e_1 is high and $\mathbf{x}$ does not belong outside, $\frac{\partial\phi}{\partial t}$ is made more negative, so ϕ lowers at that point and the contour grows outwards, swallowing $\mathbf{x}$. The same applies in reverse for e_2.

Due to the smooth form of the C^∞ regularized Heaviside (Eq. 7), $\delta(\phi) = H'(\phi)$ is nonzero everywhere. This allows ϕ some freedom to change at any point in the image, not just in a narrow band around the contour. This helps prevent convergence on local energy minima [5].

3.1 GPU implementation

The goal of the implementation is to iteratively solve Eq. 8 for $\phi(\mathbf{x}, t)$ and visualize the results at each iteration. This is done by discretizing ϕ with respect to time and applying numerical integration: starting with $\phi(\mathbf{x}, t=0)$ (which is specified by the user), an update loop computes $\phi(\mathbf{x}, t+\Delta t)$ by computing $\frac{\partial\phi}{\partial t}$ according to Eq. 8 and assuming this quantity stays constant during the short time step Δt. Existing GPU level set methods implement their update rule inside a single kernel function; however, E^{LGDF} is more challenging as relies on intermediate stages with neighborhood operations, such as convolutions and derivatives, whose sequential dependencies must be considered such as to avoid race conditions.

The update rule in Eq. 8 requires convolutions (Eq. 10) of intermediate variables that themselves rely on other convolutions (Eqs. 3–4). The relationships of these variables are shown in Fig. 2, where an arrow from A to B indicates that A is required in the computation of B. Wherever they appear, I denotes the input image and H the smooth Heaviside function (Eq. 7). All variables of the form GX represent the n-dimensional Gaussian convolution of X.

We compute the means and variances (Eqs. 3–4) from GIH, GH, GI^2H, GI and GI^2 using the following formulas:

$$u_1 = \frac{GIH}{GH} \qquad \sigma_1^2 = \frac{GI^2H}{GH} - u_1^2 \tag{11}$$

$$u_2 = \frac{GI - GIH}{1 - GH} \qquad \sigma_2^2 = \frac{GI^2 - GI^2H}{1 - GH} - u_2^2 \tag{12}$$

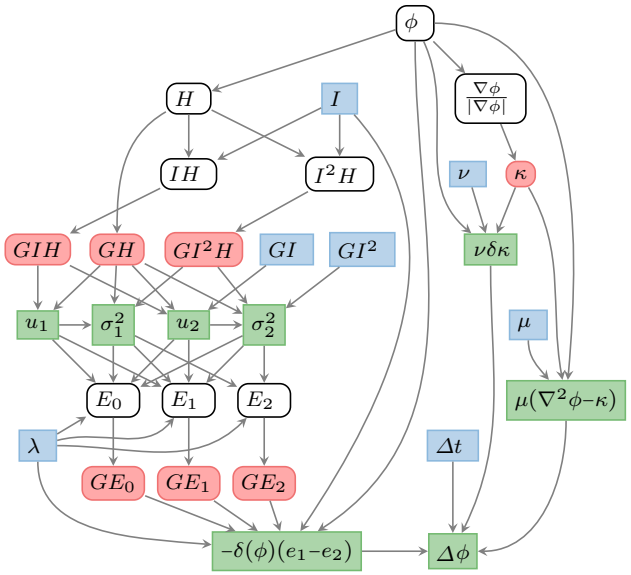


Fig. 2 Dependency graph between variables in the update process. The red variables require neighborhood computations, whereas the blue variables represent constants. All variables except for the parameters ν, μ, λ and Δt are spatially varying fields. The green variables are quantities that are computed 'on the fly' and never stored in a texture

For σ_i^2 we have used the alternative variance formula $\mathrm{Var}[\mathrm{X}] = \mathrm{E}[\mathrm{X}^2] - \mathrm{E}[\mathrm{X}]^2$, and for u_2 and σ_2 we have used $G_\sigma * (1 - H) = 1 - G_\sigma * H$ in the denominators, where $G_\sigma *$ denotes convolution with a Gaussian kernel of standard deviation σ. This is not to be confused with σ_1 and σ_2, the local intensity standard deviations outside and inside the contour. By exploiting these tricks, we are able to compute Eqs. 11–12 using only three convolutions per update cycle (since GI and GI^2 are constant). To compute the image force term $e_1 - e_2$, we expand the brackets in Eq. 10 to get:

$$\begin{aligned} e_i(\mathbf{x}) = &\int_\Omega \omega(\mathbf{y}-\mathbf{x})\left[\log(\sigma_i(\mathbf{y})) + \frac{u_i(\mathbf{y})^2}{2\sigma_i(\mathbf{y})^2}\right]\mathrm{d}\mathbf{y} \\ &- I(\mathbf{x})\int_\Omega \omega(\mathbf{y}-\mathbf{x})\frac{u_i(\mathbf{y})}{\sigma_i(\mathbf{y})^2}\,\mathrm{d}\mathbf{y} \\ &+ I(\mathbf{x})^2\int_\Omega \omega(\mathbf{y}-\mathbf{x})\frac{1}{2\sigma_i(\mathbf{y})^2}\,\mathrm{d}\mathbf{y} \end{aligned} \tag{13}$$

$$\begin{aligned} = &G_\sigma * \left[\log(\sigma_i(\mathbf{y})) + \frac{u_i(\mathbf{y})^2}{2\sigma_i(\mathbf{y})^2}\right] \\ &- I(\mathbf{x})\left[G_\sigma * \frac{u_i(\mathbf{y})}{\sigma_i(\mathbf{y})^2}\right] + I(\mathbf{x})^2\left[G_\sigma * \frac{1}{2\sigma_i(\mathbf{y})^2}\right] \end{aligned} \tag{14}$$

To compute the three terms in Eq. 14, we first pre-compute the operands of the Gaussian convolutions (E_0, E_1 and E_2 in Fig. 2), then convolve them (GE_0, GE_1 and GE_2 in

Fig. 2), then weight them by 1, I and I^2 and sum them. This results in just six convolutions altogether. Note that e_1 and e_2 are not computed separately; the variables E_0, E_1 and E_2 are the three corresponding parts of $e_1 - e_2$.

3.2 GPU architecture

The six required Gaussian convolutions require a large number of buffer reads. However, an n-dimensional Gaussian filter can be separated into the matrix product of n vectors allowing us to convolve with n 1D filters instead of one very large n-dimensional filter. This reduces l^2 texture samples to $2l$ in 2D or l^3 texture samples to $3l$ in 3D, for a truncated Gaussian kernel of length l. Therefore, our overall algorithmic complexity is $O(n \cdot l)$ for an input of size n.

The buffer reads for the horizontal Gaussian pass are coalesced, but for the vertical and depth passes the reads are not coalesced and therefore very slow. This could be alleviated by transposing the image between convolutions, making the buffer reads coalesced for vertical and depth passes. However, transposing the image three times per convolution is slow, even when this is optimized by using local/shared memory. In our architecture, we instead make use of texture memory, which preserves spatial locality among neighboring pixels in all three dimensions, making access time for all three passes comparable to coalesced buffer reads. This allows us to skip the transpositions altogether and convolve up to four images at once in the available texture memory channels, yielding faster overall performance than local/shared memory approaches.

Texture memory buffers must either be read-only or write-only within a given kernel function; therefore, results computed from data in a texture buffer must be written to a different buffer. The memory layout for our architecture includes kernels for the separable X, Y, and Z Gaussian passes accordingly, which we show in Fig. 3. This figure lists our kernels in the order they are called and shows their inputs and outputs (corresponding to the nodes in Fig. 2) within the available 4×32-bit channels per GPU texture buffer. Besides the convolutions, the rest of our implementation is straightforward; we store the 1D convolution filter weights in constant memory and all intermediate values reside in registers.

The three Gaussian convolutions of the image and Heaviside (GIH, GH, GI^2H, Fig. 2) are the result of neighborhood operations, but are not dependent on each other. This is also the case with the three Gaussian convolutions GE_0, GE_1, GE_2. We therefore create kernels shown in Fig. 3 to perform each set of three Gaussian convolutions simultaneously, and two more kernels to prepare for them (called 'Prep Conv 1' to compute H, IH, I^2H, and 'Prep Conv 2' to compute E_0, E_1, E_2). The curvature field κ (Eq. 9) requires all three (two in 2D) gradient components to be first stored in texture memory in order to avoid race conditions, since all differential operations are computed by central finite differences, a neighborhood operation. This is why we compute κ early on and pass it through the Gaussian convolution kernels in the conveniently available w channel of the texture buffer; computing κ immediately before 'Update ϕ' would require an extra texture buffer since there is only one unused channel at that point. After updating, we force the partial derivatives of ϕ to be zero at their corresponding image boundaries (in the 'Neumann/Copy' kernel) to prevent numerical instability and copy the result back into buffer A for the next iteration.

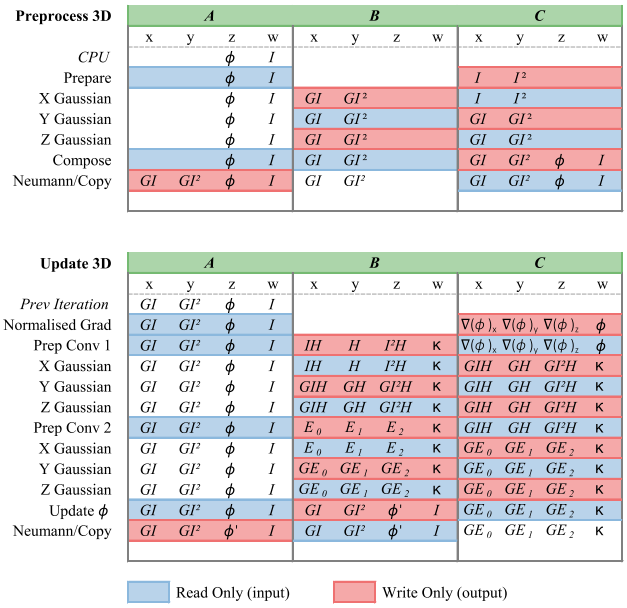


Fig. 3 Memory layout of our GPU kernels for the 3D case. Each row represents a kernel operating on 4-channel texture objects A, B, C. The kernels read variables from one or two of the textures (blue) and write into a single texture (red)

3.3 Interactive brushes

There are many applications in the biosciences, computer vision, medical, and pattern recognition communities where guidance by human experts is required [7, 20, 27, 48, 50]. The current interactive GPU level set methods, such as [36], provide interfaces to (1) initialize ϕ inside/outside the object, (2) dynamically adjust parameters, and in some cases (3) allow ϕ to be edited (a union operator on new objects/regions, followed by rerunning of the algorithm); however, it is difficult to refine evolution such as to prevent contour leaking or constrain the evolution. The graph-cuts and radial-basis function approaches

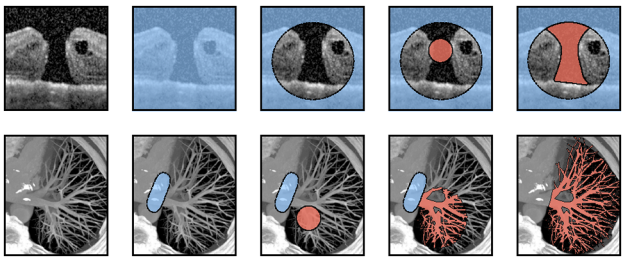


Fig. 4 Figure illustrating interactive use of our brush functions. The blue region represents the barrier brush $\phi = \infty$ and red regions are where $\phi < 0$ and otherwise $\phi > 0$

[15, 27] allow users to sketch lines or define control points which are tagged to both the desired object and the undesired regions, but we find the process difficult to refine where the segmented boundary lies somewhere between the input locations, where there may not be discernible image intensity features (see Fig. 4 top-left and in the accompanying video).

To address these issues, we follow the strategies outlined in the survey [29] with similar functions to the modeling/graphics literature [12]; however, we closely integrate brush functions with our segmentation kernels with the goal of editing and constraining ϕ during the iterative evolution process itself. Specifically, we provide functions to initialize, append, erase, and constrain (locally stop evolution of ϕ) after each iteration of the update step (Eq. 8), and visualize the results after each iteration. Note that for simplicity we define our functions with circular (2D) or spherical (3D) regions, but there is nothing to prevent implementing more bespoke functions, such as surface pulling [12].

All brush functions are centered at the mouse position $\mathbf{p}$ with radius r and are implemented in the 'Compose' kernel (Fig. 3). We have deliberately arranged the read buffer B to link to ϕ from the previous update iteration. To complete a brush action, we relaunch the 'Compose' kernel with the brush parameters followed by the 'Neumann/Copy' kernel between each update iteration. The initialization brush sets ϕ to a binary step function with a small positive constant (we choose 2 empirically):

$$\phi(\mathbf{x}) := 2 \cdot \mathrm{sgn}(\|\mathbf{x} - \mathbf{p}\| - \mathrm{r}) \tag{15}$$

where $:=$ denotes assignment. The user can continue to 'paint' new foreground regions using the additive brush:

$$\phi(\mathbf{x}) := \begin{cases} \phi(\mathbf{x}) & \text{if } \|\mathbf{x} - \mathbf{p}\| - r > 0 \\ \min(\|\mathbf{x} - \mathbf{p}\| - \mathrm{r}, \phi(\mathbf{x})) & \text{otherwise} \end{cases} \tag{16}$$

To erase a foreground region, we simply reassign any values inside the brush region with a small positive constant:

$$\phi(\mathbf{x}) := \begin{cases} \phi(\mathbf{x}) & \text{if } \|\mathbf{x} - \mathbf{p}\| - r > 0 \\ 2 & \text{otherwise} \end{cases} \tag{17}$$

However, while the erase brush is useful for undoing undesired strokes, it will not stop the contour from leaking into undesired regions, as ϕ will continually update and burst through the previously erased region again. Therefore, we introduce a 'barrier' brush to persistently block the level set from growing into a fixed region. Rather than define this region in another buffer, we set ϕ to ∞ and check for ∞ values when computing $\Delta\phi$ in the 'Update ϕ' kernel:

$$\phi(\mathbf{x}) := \begin{cases} \phi(\mathbf{x}) & \text{if } \|\mathbf{x} - \mathbf{p}\| - r > 0 \\ \infty & \text{otherwise} \end{cases} \text{(compose kernel)} \tag{18}$$

$$\Delta\phi(\mathbf{x}) := \begin{cases} 0 & \text{if } \phi(\mathbf{x}) = \infty \\ \Delta\phi(\mathbf{x}) & \text{otherwise} \end{cases} \quad \text{(update } \phi \text{ kernel)} \tag{19}$$

In our implementation, we found it useful to allow users to pause and unpause evolution with $\Delta t = 0$ and $\Delta t = 0.1$, while still allowing users to commit brush strokes. This makes it easier to guide the contour without having to compete against its growth. Furthermore, by using the previous value of ϕ stored in the B buffer z-channel in combination with the rendered value of ϕ stored in the A buffer z-channel, we can display the currently brush size and position without committing the stroke.

In Fig. 4, we illustrate two simple use-cases of our interactive brushes. In the top row, the user paints using the 'barrier' brush to cover the full image region, shown in blue. This is followed by the 'erase' brush (Eq. 17), to cut a permissible region in which a new seed region is placed (Eq. 16), which evolves to segment the macular hole without leaking into the opening. (We show this in 3D in the accompanying video.) Similarly, in the lower row, the vessels are segmented without leaking into the heart (see also Table 5 2b–c).

3.4 Real-time rendering

To render the zero-crossing of the level set function ϕ in 3D, we launch a render kernel after the Neumann/Copy step in the update loop (Fig. 3). We send a camera matrix to initialize each pixel with a ray origin $\mathbf{o}$ and direction unit vector $\hat{d}$. We parameterize the ray's position by $\mathbf{r} = \mathbf{o} + \hat{d}s$ and, assuming ϕ to be the signed distance to the zero-crossing, advance the ray in steps by $s_{i+1} = s_i + \phi(\mathbf{r})$. However, ϕ is not a perfect signed distance function; therefore, we must divide our step size by the maximum derivative of ϕ; this value is not known precisely, but in

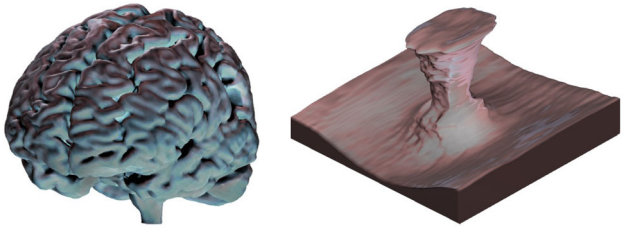


Fig. 5 3D views during segmentation rendered in real time. **a** 3D segmented brain. **b** 3D segmented macular hole

practice we find we can obtain sufficiently small visual artifacts at good performance by choosing a constant step size $\Delta s = 0.3\phi(\mathbf{r})$. Further, given that ϕ is not defined outside of the image boundaries, we initially advance s_0 to the start of the image axis-aligned bounding box (where the s_0 is calculated using an analytical ray-box intersection function [21]). To increase visual quality, we implement 3D ambient occlusion and soft-shadows by marching the ray in the directional of the normal and light source once it has hit a surface [11].

The output of our real-time rendering implementation, using hardware trilinear interpolation to sample ϕ and with $\Delta s = 0.3\phi(\mathbf{r})$, is shown in Fig. 5. (The render kernel has negligible impact on performance.)

4 Results and validation

In this section, we provide quantitative results validating our algorithm's performance, parameter insensitivity, and robustness to noise. We also provide qualitative results to justify the utility of our interactive brushes and assess the segmentation of real-world images from various domains.

To confirm that our algorithm implements the LGDF energy model correctly, we measure the Jaccard index between the resulting segmentations from the original sequential CPU implementation and our GPU implementation, and show the results in Table 1.

These results show the GPU to be near-identical to the CPU implementation; we find small discrepancies at the boundary at sub-voxel precision caused by different implementations of low-level math library functions and different (mathematically equivalent) algebra in the intermediate steps (Eqs. 11 and 12).

4.1 Noise and parameter insensitivity

We conducted a large number of noise experiments on a synthetic 2D object, which has sharp and smooth features, and plot the mean and standard deviation of the results in Fig. 6. These experiments all use the same parameters and initialize ϕ to a small circle inside the synthetic object. We also qualitatively show a subset of the experiments in Table 2 from the same synthetic 2D object, and for a 3D macular hole [45].

Table 1 Comparing the Jaccard index for our GPU implementation with the CPU implementation

Image	Jaccard index
Synthetic objects 2D	1
Tumor (small) 2D	1
Tumor (large) 2D	0.981
Macular hole 3D	0.990
Brain 3D	0.984
Tumor 3D	0.993

The results in Fig. 6 show that the method can segment severely noisy images, corrupted with a PSNR of about $10^{1.05}$, under a constant parameter assignment. While the results in Fig. 6 show the method is more robust to Gaussian noise than speckle noise, it is important to understand that this is only within the parameters chosen; improvements can generally be made by adjusting the parameters for individual scenarios. In addition to Gaussian, salt and pepper, and speckle noise, we implemented a multi-frequency 'cloud' noise at a target PSNR, which simulates intensity inhomogeneity. In Fig. 6, it appears that the cloud noise improves under a PSNR of $10^{0.81}$; however, this is caused by the cloud-like objects inside the synthetic object being captured. In such cases, we can still segment the underlying object, but only through decreasing σ or using the interactive brushes.

By systematically adjusting the parameters to maximize the mean Jaccard index over all noise types, we found the

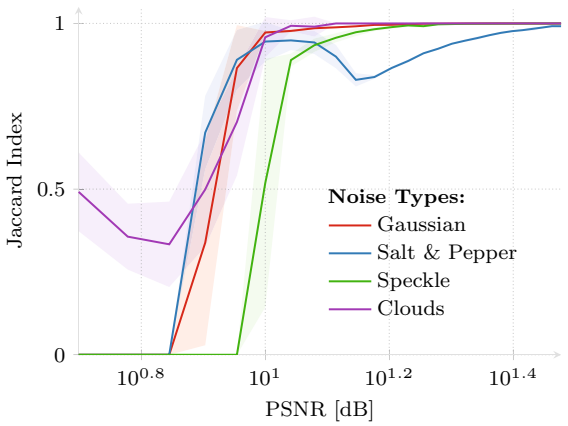
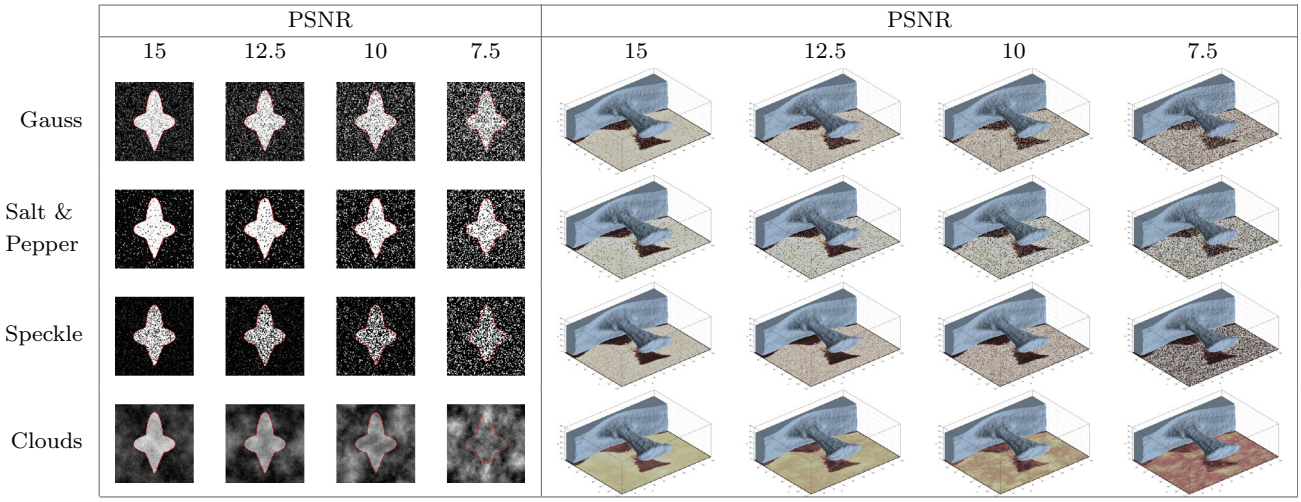


Fig. 6 Jaccard index of a synthetic ground-truth segmentation and our segmentation result using the same parameters on 4 different types of noise. The standard deviation is shown by the error envelopes (transparent shaded regions); our method is robust to several noise types heavily corrupting the object to a PSNR of about $10^{1.05}$

Table 2 Segmentation without interactive brushes attained from a single circular seed region inside the object



following defaults: $\sigma = 3$, $v = 50$, $\lambda_1 = 1$, $\lambda_2 = 1.05$, $\Delta t = 0.1$, $\mu = 1$. (These are the parameters used in Fig. 6 and Table 2.) We also found, through our synthetic experiments and in segmenting real-world images, that across all of the encountered images we only need to adjust σ, v, and λ, where $\lambda_1 = 1 + \max(0, -\lambda)$ and $\lambda_2 = 1 + \max(0, \lambda)$. To make these parameters more intuitive, we assign more meaningful descriptions to them in Table 3:

We call σ a 'capture range' parameter as it describes the range from which a pixel's energy may be affected by the contour (see Eqs. 2–4) and therefore determines the capture range. The parameter v penalizes the length of the contour (Eqs. 6 and 8); a larger v value results in a smoother contour which is less likely to burst through small gaps or capture small/sharp features. Traditionally many active contour methods have been designed to grow or shrink until they reach the object boundary and then stop; the parameter λ optionally enables this behavior by weighting the image terms e_1 and e_2 by λ_1 and λ_2, respectively (Eq. 8), biasing the contour toward shrinking or growing. By adjusting these parameters in real time, inexperienced users quickly learn to intuitively manipulate them in combination with our interactive brushes. In most cases, we set $\lambda = 0.05$ to prefer contour growth and adjust only σ and v.

To further justify the importance of our interactive brushes, we construct 6 extreme synthetic scenarios in Table 4. Images 1–3 show Gaussian, salt and pepper, and cloud noise corrupted to a severe PSNR of 5 (fail cases in Fig. 6). By adjusting the parameters and constraining the contour with our brushes, we can easily (3–5 s per image) segment the underlying object. Images 4–5 show that the LGDF energy can segment noisy objects with intensity inhomogeneity and weak/blurred edges. Image 6 shows an object whose intensity mean is the same as its background, with the only difference being in intensity variance.

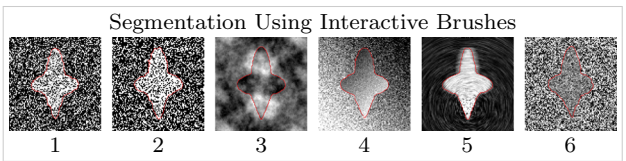
4.2 Segmenting real-world images

We evaluate our software against several different imaging modalities using real-world images and show the results in Table 5. In all our results, we only adjust the parameters σ, v, and λ as described in Table 3. By initializing $\phi(\mathbf{x}) = 2$ uniformly, we are able to automatically segment small objects without an initial seed region, such as for images of

Table 3 Our proposed parameters for controlling the method. All images in this paper are generated using these three parameters within their suggested range and constants $\Delta t = 0.1$ and $\mu = 1.0$

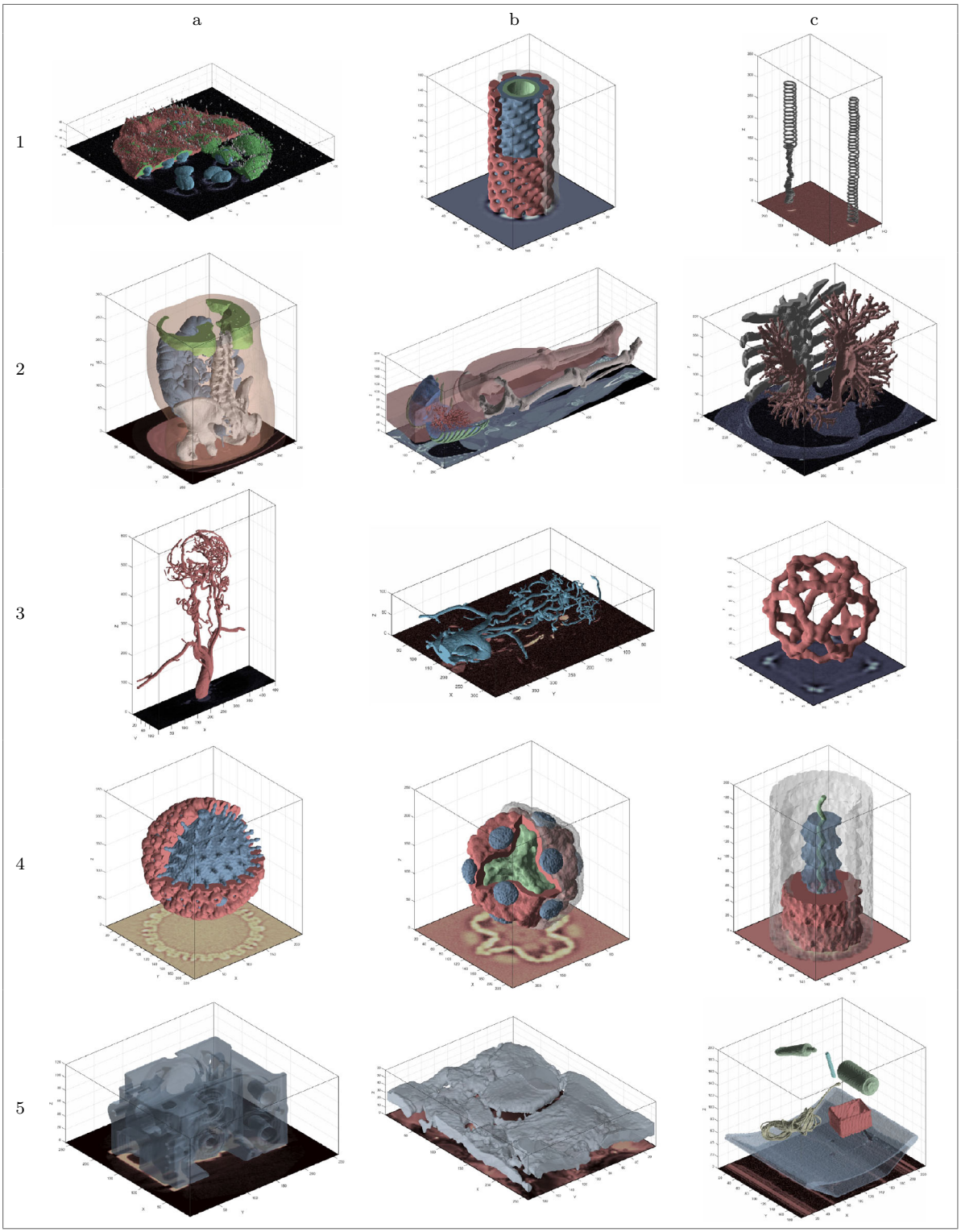
Description	Symbol	Suggested range	Default
Capture range	σ	$[1.01,\ 10]$	3
Smoothing weight	v	$[10,\ 90]$	50
Shrink or grow	λ	$[-0.1,\ 0.1]$	0.05

Table 4 Following challenging scenarios are quickly and easily segmented with our interactive brushes



Images 1–5 have a PSNR of 5 for Gaussian, salt and pepper, and multi-frequency noise accordingly, and images 4–6 show extreme scenarios of poorly defined and/or blurred boundaries

Table 5 Segmentation results of multiple objects displayed in different colors. 1a shows a segmented image of HaCaT human cell culture cells using confocal microscopy, 1b shows the interdigitation of segmented layers of eisosome proteins from cryo-EM tomography data [19], 1c shows a malaria sporozoite [38]. Row 2 shows medical CT scans of the abdomen, body, and thorax [37]. 3a shows an MRI of a cerebral aneurysm and 3b an XA angiogram [37]. 3c shows the structure of the Sec13/31 COPII coat cage from cryo-EM data [44]. Row 4 shows the herpes simplex virus capsid [6], phi procapsid [40], and the mumps virus [9], all from cryo-EM data. Row 5 shows applications outside of biology and medicine: 5a is a CT scan of an engine block [2], 5b sintered alumina [38], and 5c shows a selection of objects from a CT scan of a backpack [2]



cells. This works because $\delta(2)$ is large enough that ϕ can still be deformed by image forces, allowing new segments to appear anywhere in the image; this is not possible with a narrow band approach. In general, the default parameters suggested in Table 3 work well for most object segmentations; however, in challenging cases (such as multiple objects or thin objects) the parameters σ and v can be dynamically adjusted in real time where the user can 'slide' the parameter within the suggested range until the motion of the contour is satisfactory to achieve the desired result.

Many of the segmentations (Table 5 1a, 3a-b, and 5b-c) are not possible with the current GPU level set segmentation approaches, which use simple speed functions to attract and/or shrink the contour within a fixed intensity range [18, 23, 36]. For example, when painting an initial seed region inside a vessel network with intensity inhomogeneity, the active contour will not grow along the vessel. In contrast, the adopted LGDF energy model allows us to paint a simple initial sphere anywhere on the object which then spreads through the network of vessels. In cases where the contour evolution misses a vessel or oversegments part of the object, evolution is temporarily halted ($\Delta t = 0$), local amendments are made, and then evolution is resumed ($\Delta t = 0.1$). By making local adjustments with a high level of visual feedback, we can spot such issues and make amendments immediately.

4.3 Performance and memory usage

In our cross-platform C++/OpenCL application, we measure the mean kernel timings over 100 frames for different sized images on a GTX TITAN X and show the results in Fig. 7. We can see that the overall algorithm performance is approximately linear in the number of pixels/voxels, since we process the full dataset as the C^∞ Heaviside and Dirac functions are nonzero everywhere. This agrees with our expected complexity of $O(n \cdot l)$ for an input of n voxels and a truncated 1D Gaussian kernel of length l.

Figure 8 shows how the overall running time increases with larger σ and that the performance in the z-axis becomes more similar to the y- and x-axes with larger σ. In the practical and suggested range of σ [1.01, 10] (Table 3), it can be seen that the running time increases in small steps (zoom to the lower-left of the graph). This is because running time is primarily influenced by the size of the 1D Gaussian filter buffer, whose size is $\lfloor 4\sigma + 1 \rfloor$ to approximate the Gaussian function with reasonable support.

We also investigated other optimizations given that the Gaussian convolution is the primary bottleneck of our approach. We implemented Gaussian convolution in the Fourier domain using MATLAB GPU arrays. While Fourier convolution allows for a lower order of growth,

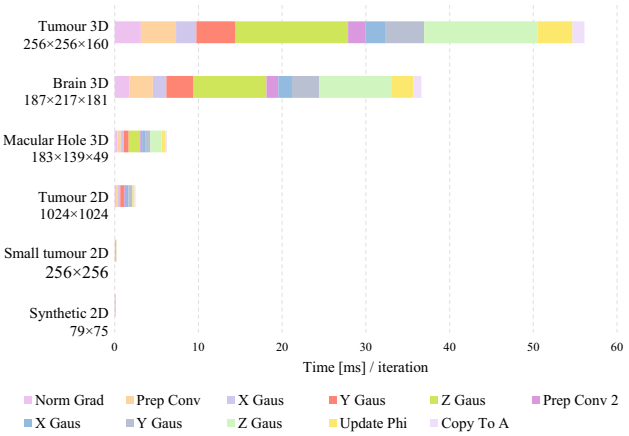


Fig. 7 Mean kernel timings over 100 frames for different images of different sizes. $\sigma = 3$ in all cases. Despite using texture memory, which is cached and has spatial locality in multiple dimensions [43], and fast constant memory to store the 1D separable Gaussian coefficients, convolution in the z-axis is significantly slower than the y- and x-axes

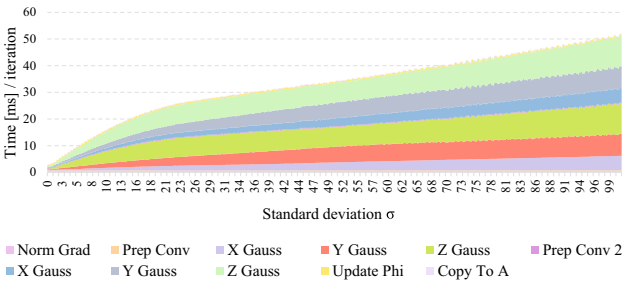


Fig. 8 Mean kernel timings over 100 frames with increasing σ for 3D macular hole. In practice, we rarely require $\sigma > 10$

the benefits are outweighed by the large constant factor due to the algorithm complexity; this takes 400ms per frame using a GTX TITAN X, which is off the scale in Fig. 8.

The mean time of 100 iterations with our C++ OpenCL implementation is evaluated across different hardware and compared to our GPU Fourier implementation and the original MATLAB version on the CPU (which is vectorized and calls code written in C for the Gaussian convolution). These results are shown in Fig. 9.

In Fig. 9, our algorithm substantially outperforms the original implementation in all images. Given that we process the entire dataset with compact kernels and separable convolutions, we can fully utilize high-end GPU hardware to obtain a substantial speedup of up to three orders of magnitude from the original version, and 1–2 orders of magnitude from our GPU Fourier convolution version. This means that segmentations which previously took over an hour can now be achieved in a few seconds, without any trade in quality.

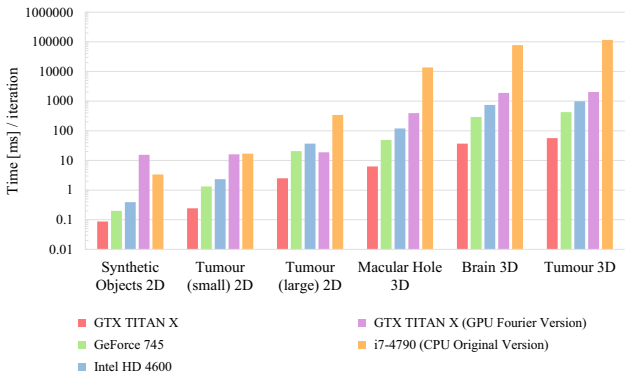


Fig. 9 Mean time [ms] over 100 iterations on different GPU hardware, compared to the original MATLAB implementation and our implementation using fast Fourier convolution on the GPU. Our OpenCL implementation achieves over a $\times$ 1,000 speedup over the original vectorized MATLAB version in larger images, scaling up with the parallel hardware accordingly

With high-end GPU hardware, our algorithm is limited by memory consumption. We require 48 bytes of texture memory per pixel or voxel for the entire image (4 bytes per channel in Fig. 3). In cases where the image does not fit into the available GPU memory, we must either downsample or crop the region of interest before segmentation.

5 Discussion

The primary limitation of our implementation is that we require storing the full dataset at the original resolution in GPU texture memory, as the C^∞ Heaviside and Dirac functions are nonzero everywhere to reduce convergence on local minima [5]. This also limits the algorithm's speed. In future work, we will investigate dynamically adjusting the resolution away from the zero-crossing of the C^∞ Heaviside, to reduce the memory requirements and improve performance, and evaluate the impact of this approach on segmentation quality.

While there are some excellent publicly available datasets for interactive segmentation of real-world 2D color images and videos [53], the problem of segmenting everyday objects in color photographs, e.g., with a graph cut approach on distributions of color information, is fundamentally different to segmenting a tissue or organ. In the latter case, the challenge is more often due to intensity inhomogeneity or poorly defined edges, rather than complex backgrounds or discontinuities within the object. As with [51], we would like to see benchmark 3D biological and medical datasets for evaluating interactive performance.

6 Conclusion

In conclusion, we have shown that sophisticated level set segmentation energy models, with sequential dependencies among intermediate processing steps, can be implemented efficiently on the GPU through careful structuring of the GPU kernels within the constraints of the GPU memory architecture. While active contours are used in unsupervised algorithms, they continue to benefit from interactive approaches that enable users to guide and constrain the contour to capture specific parts of more challenging objects. We have shown that the LGDF energy model proposed by [47] requires little parameter tuning, is robust against different types of noise, and can be generalized to a broad range of real-world 3D images from biology and medicine. Segmenting many of these images was not possible with existing GPU level set algorithms due to their simple energy functionals. We have greatly enhanced the LGDF model's performance, making it practical in many more use-cases than before (including 3D images). We also extended its functionality through interactive brush functions that give direct influence over the dynamic contour evolution. In the future, we believe GPU adaptations of advanced segmentation algorithms will continue to proliferate, using similar design processes to ours.

7 Availability

We release our C++/OpenCL software and source code under the GNU General Public License Version 3, alongside an optional MATLAB wrapper. The implementation is cross-platform using GLFW with few dependencies, where binaries for Linux and Windows are also available: https://github.com/cwkx/IGAC.

Acknowledgements We are grateful to NVIDIA for providing a GTX TITAN X for this research. Table 5 1a shows fixed HaCaT human cell culture cells stained with SiR-Actin (Spirochrome) RED, rat anti-tubulin antibody/secondary anti-rat Alexa488 antibody GREEN and DNA DAPI BLUE. The cells are imaged with a Zeiss 880 Airyscan LSM confocal microscope, prepared, and imaged by Miss Bethany Cole, Miss Joanne Robson & Dr Tim Hawkins. Durham Centre for Bioimaging Technology, Department of Biosciences, Durham University.

References

1. Adalsteinsson, D., Sethian, J.A.: A fast level set method for propagating interfaces. J. Comput. Phys. **118**(2), 269–277 (1995)
2. Bartz, D.: Volvis datasets. http://www.volvis.org (2005). Accessed 2016 Mar 30
3. Caselles, V., Catté, F., Coll, T., Dibos, F.: A geometric model for active contours in image processing. Numer. Math. **66**(1), 1–31 (1993)
4. Chambolle, A., Pock, T.: A first-order primal-dual algorithm for convex problems with applications to imaging. J. Math. Imaging Vis. **40**(1), 120–145 (2011)
5. Chan, T.F., Vese, L., et al.: Active contours without edges. IEEE Trans. Image Process. **10**(2), 266–277 (2001)
6. Chang, J.T., Schmid, M.F., Rixon, F.J., Chiu, W.: Electron cryotomography reveals the portal in the herpesvirus capsid. J. Virol. **81**(4), 2065–2068 (2007)
7. Chen, H.L.J., Samavati, F.F., Sousa, M.C., Mitchell, J.R.: Sketch-based volumetric seeded region growing. In: Proceedings of the Third Eurographics Conference on Sketch-Based Interfaces and Modeling, pp. 123–130 (2006)
8. Cocosco, C.A., Kollokian, V., Kwan, R.K.-S., Pike, G.B., Evans, A.C.: BrainWeb: Online interface to a 3D MRI simulated brain database. NeuroImage **5**, 425 (1997)
9. Cox, R., Pickar, A., Qiu, S., Tsao, J., Rodenburg, Cynthia, Dokland, T., Elson, A., He, B., Luo, M.: Structural studies on the authentic mumps virus nucleocapsid showing uncoiling by the phosphoprotein. Proc. Natl. Acad. Sci. U.S.A. **111**(42), 15208–15213 (2014)
10. Eklund, A., Dufort, P., Forsberg, D., LaConte, S.M.: Medical image processing on the GPU past, present and future. Med. Image Anal. **17**(8), 1073–1094 (2013)
11. Evans, A.: Fast approximations for global illumination on dynamic scenes. In: *ACM SIGGRAPH 2006 Courses*, pp. 153–171. ACM (2006)
12. Eyiyurekli, M., Breen, D.: Interactive free-form level-set surface-editing operators. Comput. Graph. **34**(5), 621–638 (2010)
13. Fulkerson, B., Soatto, S.: Really quick shift: image segmentation on a GPU. In: Trends and Topics in Computer Vision, pp. 350–358 (2010)
14. Gorelick, L., Schmidt, F.R., Boykov, Y.: Fast trust region for segmentation. In: 2013 IEEE Conference on Computer Vision and Pattern Recognition, pp. 1714–1721 (2013)
15. Grady, L.: Random walks for image segmentation. IEEE Trans. Pattern Anal. Mach. Intell. **28**(11), 1768–1783 (2006)
16. He, Z., Kuester, F.: GPU-based active contour segmentation using gradient vector flow. In: International Conference on Advances in Visual Computing, pp. 191–201 (2006)
17. Jarrin, M., Young, L., Wu, W., Girkin, J.M., Quinlan, R.A.: Chapter twenty-one—in vivo, ex vivo, and in vitro approaches to study intermediate filaments in the eye lens. In: Omary, M.B., Liem, R.K.H. (eds.) Intermediate Filament Proteins, volume 568 of Methods in Enzymology, pp. 581 – 611. Academic Press (2016)
18. Jeong, W.K., Beyer, J., Hadwiger, M., Vazquez, A., Pfister, H., Whitaker, R.T.: Scalable and interactive segmentation and visualization of neural processes in em datasets. IEEE Trans. Vis. Comput. Graph. **15**(6), 1505–1514 (2009)
19. Karotki, L., Huiskonen, J.T., Stefan, C.J., Ziółkowska, N.E., Roth, R., Surma, M.A., Krogan, N.J., Emr, S.D., Heuser, J., Grünewald, K., Walther, T.C.: Eisosome proteins assemble into a membrane scaffold. J. Cell Biol. **195**(5), 889–902 (2011)
20. Kass, M., Witkin, A., Terzopoulos, D.: Snakes: active contour models. Int. J. Comput. Vis. **1**(4), 321–331 (1988)
21. Kay, T.L., Kajiya, J.T.: Ray tracing complex scenes. In: Conference on Computer Graphics and Interactive Techniques, SIGGRAPH, pp. 269–278. ACM (1986)
22. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015)
23. Lefohn, A.E., Kniss, J.M., Hansen, C.D., Whitaker, R.T.: A streaming narrow-band algorithm: interactive computation and visualization of level sets. IEEE Trans. Vis. Comput. Graph. **10**(4), 422–433 (2004)
24. Li, C., Xu, C., Gui, C., Fox, M.D.: Level set evolution without re-initialization: a new variational formulation. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 1, pp. 430–436 (2005)
25. Li, M., He, C., Zhan, Y.: Adaptive level-set evolution without initial contours for image segmentation. J. Electron. Imaging **20**(2), 023004 (2011)
26. Malladi, R., Sethian, J.A., Vemuri, B.C.: Shape modeling with front propagation: a level set approach. IEEE Trans. Pattern Anal. Mach. Intell. **17**(2), 158–175 (1995)
27. Mory, B.: Interactive Segmentation of 3D Medical Images with Implicit Surfaces. Ph.D. thesis, STI, Lausanne (2011)
28. Mumford, D., Shah, J.: Optimal approximations by piecewise smooth functions and associated variational problems. Commun. Pure Appl. Math. **42**(5), 577–685 (1989)
29. Olabarriaga, S.D., Smeulders, A.W.M.: Interaction in the segmentation of medical images: a survey. Med. Image Anal. **5**(2), 127–142 (2001)
30. Osher, S., Fedkiw, R.: Level Set Methods and Dynamic Implicit Surfaces. Applied Mathematical Sciences. Springer, New York (2002)
31. Osher, S., Sethian, J.A.: Fronts propagating with curvature-dependent speed: algorithms based on Hamilton–Jacobi formulations. J. Comput. Phys. **79**(1), 12–49 (1988)
32. Peng, D., Merriman, B., Osher, S., Zhao, H., Kang, M.: A PDE-based fast local level set method. J. Comput. Phys. **155**(2), 410–438 (1999)
33. Pock, T., Cremers, D., Bischof, H., Chambolle, A.: An algorithm for minimizing the Mumford–Shah functional. In: IEEE International Conference on Computer Vision, pp. 1133–1140 (2009)
34. Pratx, G., Xing, L.: GPU computing in medical physics: a review. Med. Phys. **38**, 2685 (2011)
35. Ren, C.Y., Reid, I.: gSLIC: a real-time implementation of SLIC superpixel segmentation. Technical report, University of Oxford, Department of Engineering, Technical Report (2011)
36. Roberts, M., Packer, J., Sousa, M.C., Mitchell, J.R.: A work-efficient GPU algorithm for level set segmentation. In: Proceedings of the Conference on High Performance Graphics, pp. 123–132. Eurographics Association (2010)
37. Rosset, A., Spadola, L., Ratib, O.: Osirix: an open-source software for navigating in multidimensional dicom images. J. Digit. Imaging **17**(3), 205–216 (2004)
38. Schindelin, J., Arganda-Carreras, I., Frise, E., Kaynig, V., Longair, M., Pietzsch, T., Preibisch, S., Rueden, C., Saalfeld, S., Schmid, B., Tinevez, J.-Y., White, D.J., Hartenstein, V., Eliceiri, K., Tomancak, P., Cardona, A.: Fiji: an open-source platform for biological-image analysis. Nat. Methods **9**(7), 676–682 (2012)
39. Schmid, J., Iglesias-Guitián, J., Gobbetti, E., Magnenat-Thalmann, N.: A GPU framework for parallel segmentation of volumetric images using discrete deformable models. Vis. Comput. **27**(2), 85–95 (2010)
40. Sen, A., Heymann, J.B., Cheng, N., Qiao, J., Mindich, L., Steven, A.C.: Initial location of the RNA-dependent RNA polymerase in the bacteriophage Phi6 procapsid determined by cryo-electron microscopy. J. Biol. Chem. **283**(18), 12227–12231 (2008)

41. Shi, L., Liu, W., Zhang, H., Xie, Y., Wang, D.: A survey of GPU-based medical image computing techniques. Quant. Imaging Med. Surg. **2**(3), 2223–2292 (2012)
42. Smistad, E., Elster, A.C., Lindseth, F.: Real-time gradient vector flow on GPUs using OpenCL. J. Real Time Image Process. **10**(1), 67–74 (2012)
43. Smistad, E., Falch, T.L., Bozorgi, M., Elster, A.C., Lindseth, F.: Medical image segmentation on GPUs a comprehensive review. Med. Image Anal. **20**(1), 1–18 (2015)
44. Stagg, S.M., Gürkan, C., Fowler, D.M., LaPointe, P., Foss, T.R., Potter, C.S., Carragher, B., Balch, W.E.: Structure of the Sec13/31 COPII coat cage. Nature **439**(7073), 234–238 (2006)
45. Steel, D.H.W., Lotery, A.J.: Idiopathic vitreomacular traction and macular hole: a comprehensive review of pathophysiology, diagnosis, and treatment. Eye **27**(1), 1–21 (2013)
46. Vineet, V., Narayanan, P.J.: CUDA cuts: fast graph cuts on the GPU. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 1–8 (2008)
47. Wang, L., He, L., Mishra, A., Li, C.: Active contours driven by local Gaussian distribution fitting energy. Signal Process. **89**(12), 2435–2447 (2009)
48. Whitaker, R., Breen, D., Museth, K., Soni, N.: Segmentation of Biological Volume Datasets Using a Level-Set Framework, pp. 249–263. Springer, Vienna (2001)
49. Xu, C., Prince, J.L.: Snakes, shapes, and gradient vector flow. IEEE Trans. Image Process. **7**(3), 359–369 (1998)
50. Yushkevich, P.A., Piven, J., Hazlett, H.C., Smith, R.G., Ho, S., Gee, J.C., Gerig, G.: User-guided 3D active contour segmentation of anatomical structures: significantly improved efficiency and reliability. Neuroimage **31**(3), 1116–1128 (2006)
51. Zhao, F., Xie, X.: An overview of interactive medical image segmentation. Ann. BMVA **2013**(7), 1–22 (2013)
52. Zhao, H.-K., Chan, T., Merriman, B., Osher, S.: A variational level set approach to multiphase motion. J. Comput. Phys. **127**(1), 179–195 (1996)
53. Zhu, H., Meng, F., Cai, J., Shijian, L.: Beyond pixels: a comprehensive survey from bottom-up to semantic image segmentation and cosegmentation. J. Vis. Commun. Image Represent. **34**, 12–27 (2016)
54. Zhu, L., Karasev, P., Kolesov, I., Sandhu, R., Tannenbaum, A.: Interactive Image Segmentation From A Feedback Control Perspective. *ArXiv e-prints* (2016)
55. Zhu, S.C., Yuille, A.: Region competition: Unifying snakes, region growing, and Bayes/MDL for multiband image segmentation. IEEE Trans. Pattern Anal. Mach. Intell. **18**(9), 884–900 (1996)

Chris G. Willcocks received a PhD in Computer Science at Durham University in 2013 where he specialized in real-time GPU rendering and deformation of large volumetric datasets. He researched object deformation at Newcastle University before accepting a postdoctoral research position at Durham University in 2015. His interdisciplinary research aims to enable elegant solutions to otherwise challenging or computationally expensive problems in the fields of machine learning, high-performance computing, image processing, and computer graphics.

Philip T. G. Jackson received a BSc degree in Computer Science and Physics within the Natural Sciences Programme from Durham University followed by an MSc in Computer Science and is currently reading for a PhD degree in Computer Science, also from Durham University, UK. His research has investigated recurrent and generative adversarial neural networks for multi-object localization and counting, unsupervised learning of image embeddings, and recurrent neural networks for sequence learning.

Carl J. Nelson received an MSc degree in Biology and Physics within the Natural Sciences Programme from Durham University and a PhD degree in Bioimage Informatics through Computing Science, also from Durham University, UK. He is currently a postdoctoral researcher at the University of Glasgow where his interdisciplinary research focuses on developing advanced and novel microscopy systems for imaging the developing zebrafish. These microscope and analysis tools are used to further the understanding of biological processes that can be gleaned through bioimaging and microscopy.

Amar V. Nasrulloh received an undergraduate degree in Physics (2004) from Institut Teknologi Sepuluh Nopember (ITS) and a masters degree in Electrical Engineering (2010) specializing in Biomedical Engineering from Institut Teknologi Bandung (ITB). He is currently reading for a PhD degree at Durham University, UK, funded by LPDP Indonesia. His interdisciplinary research interests focus on applying physics and image processing methods to biology and medicine.

Boguslaw Obara received an MSc in Physics from the Jagiellonian University and PhD in Computer Science from the AGH University of Science and Technology, Krakow, Poland. He has been a researcher at the Polish Academy of Sciences (2001–2007), a Fulbright fellow (2006–2007) and a postdoctoral researcher at the University California, USA (2007–2009), and the University of Oxford, UK (2007–2009). He is currently an associate professor in Computer Science at Durham University, UK. His research focuses on image processing techniques.