Interactive GPU Active Contours for Segmenting Inhomogeneous Objects

Chris G. Willcocks, Philip T. G. Jackson, Carl J. Nelson, Amar V. Nasrulloh, Boguslaw Obara

Abstract—We present a segmentation software package targeting medical and biological applications, with a high-level of visual feedback and several usability enhancements over existing packages. In particular, we provide a new and fast GPU implementation of the local Gaussian distribution fitting (LGDF) energy model, which can be guided by human experts in a semi-automated framework. LGDF energy is capable of segmenting inhomogeneous objects with poorly defined boundaries, but existing implementations are prohibitively slow. While current segmentation methods can optimally minimize certain energy classes, the globally optimal solution is often not what is desired as image intensity information alone is often insufficient. Instead we provide real-time visual feedback through a built-in ray tracer of the active contour evolution, allowing users to halt evolution at any timestep to dynamically correct implausible evolution by painting new blocking regions or new seeds. Quantitive and qualititive validation is presented, demonstrating the practical efficacy of our interactive elements for a wide variety of real-world datasets.

Index Terms—Segmentation, Image Processing, Medicine, Interactive Systems, Real-time systems, Graphics Processors.

1 Introduction

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

- Biosciences:
  - Cellular, developmental & 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 & developmental neuroscience.
- Medical:
  - Automated differential diagnosis.
  - Diagnostic measurements, shape and volume, of:
    - Macular holes in retinal degeneration.
    - Aneurysms, clotting & infarction.
    - Tumors, neoplasia & dermatological moles.
    - MRI segmentation in dementia & Alzheimer's.
  - Computer Assisted Surgery:
    - Pre-surgical planning & surgery simulation.
    - Guided surgical navigation.

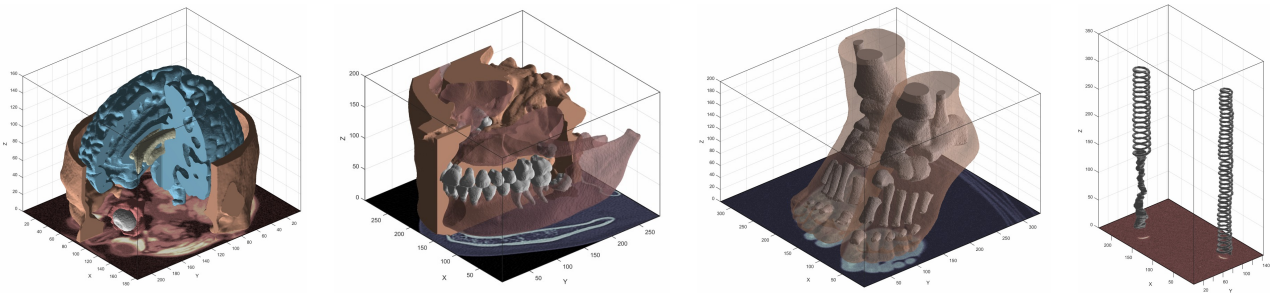
The primary problems with current segmentation approaches are that they are either: (1) too limited, e.g. only able to segment objects by simple criteria, such as objects with similar mean intensity [4], [5], (2) using too much memory or too slow, taking several hours to segment large 2D or 3D objects [6], (3) lacking in interactivity with the segmentation process in response to visual feedback [7], (4) requiring too much training data, or (5) difficult to use, requiring large interfaces and multiple algorithms [8].

- *The authors are with the School of Engineering and Computing Sciences, Durham University, South Road, DH1 3LE, Durham, United Kingdom.*
- *E-mail: {christopher.g.willcocks, p.t.g.jackson, carl.nelson, amar.v.nasrulloh, boguslaw.obara} @durham.ac.uk*

Manuscript received April 19, 2005; revised August 26, 2015.

Deep convolutional neural networks are the state-of-the-art in image segmentation, where millions of parameters of deeply layered convolutions are learnt using backpropagation [9]. 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. Similarly there are many algorithms and publicly available datasets concerned with bottom-up segmentation and semantic segmentation [10] however these mainly target 2D images or videos with color information which are significantly different to the 3D medical and biological datasets. While globally optimal solutions for certain energy classes can be found, such as with a primal dual approach [11], current implementations offer little-to-no visual feedback, which is especially problematic in large 3D images where local edits need to be made frequently.

Amongst the oldest and most widely cited segmentation approaches are active contours [12]; these are variational frameworks which allow users to define an initial open or closed curve that deforms so as to minimize an energy functional, outlining or surrounding the object of interest. While active contours have been applied to fully automatic approaches without initial contours [13], 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 an 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 [14], [15] where existing GPU segmentation methods currently rely on simple segmentation criteria restricting their usage and applications. The popular local Gaussian distribution fitting (LGDF) energy model [6] is much more powerful and able to segment a wider variety of general objects. However, it requires several intermediate processing steps that



(a) Brain & ventricles (b) Dental scan (c) Foot bones & tissues (d) Malaria sporozoite

Fig. 1: A selection of 3D objects segmented by our tool. Our interactive method allows users to efficiently capture specific objects within the data, such as the teeth in (b), which we have colored separately. Image (a) is a simulated brain MRI [1], images (b) & (c) are CT scans [2] and (d) shows a malaria sporozoite [3].

must be implemented sequentially, making it challenging to efficiently implement on graphics hardware. The current implementations of the LGDF energy model can segment small 2D images, but require several hours of processing for larger 2D or 3D images [6], preventing usage in many practical applications.

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 for the segmentation results at each update iteration, 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 active contour method of particular relevance to the image processing communities. Our software is shown to be stable to its input parameters and robust to noise through a large synthetic experiment, and it is evaluated through segmenting a wide variety of real-world images, such as in Figure 1.

2 Related Work

The field of active contours first gained mainstream adoption with the 'active snakes' model published by [12]. 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. The functional includes 'external energy' terms which are lower when the curve coincides with salient image features such as edges, along with 'internal energy' terms which penalize lack of smoothness in the contour. The result is a smooth curve that accurately deforms and locks on to object boundaries. This approach became hugely popular to retrieve precise measurements from objects, producing many novel energy functionals [24] and applications to specific domains, such as blood vessel segmentation [25].

Level set methods (the core theory in the book [26]) model contours implicitly as the zero-crossing of a scalar field. Originally they were proposed in [27] to model the evolution of inter-region boundaries in physical simulations. [28] 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 image edge. In [29], 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 [30] is minimized by an optimal partition of an image into piecewise smooth segments, and high-quality implementations exist on the GPU [16]. The global optimum can be found using a primal-dual algorithm [11] resulting in a cartoon-like rendering of the original image. Local solutions, such as with a trust-region approach [31], have applications in interactive segmentation.

2.1 GPU Segmentation

Accelerating image segmentation with GPUs is a large research field with several comprehensive surveys [14], [15], [32], [33]. The survey by [14] covers a broad range of algorithms and different imaging modalities, whereas [15] focuses more on GPU segmentation with a detailed discussion on the current GPU architecture. We discuss the literature with a focus on the quality of the GPU segmentation methods in terms of their ability to precisely segment a wide variety of complex images, and we qualitatively summarize each of the main categories in Table 1.

In Table 1, methods are awarded up to 5 stars in 'Image Complexity' if they can handle multiple imaging scenarios such as uneven lighting, severe noise, multiple objects, intensity inhomogeneity, blurred and/or broken object boundaries. Methods score 5 stars in 'Interactivity' if they provide an interface or functionality to control the segmentation process. 'Speed' measures the overall segmentation time where less than 3 stars implies that the method can only segment 2D objects at interactive rates. Methods score highly for 'Memory Efficiency' if they can process large datasets without storing the full dataset in main memory or GPU memory.

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

GPU Method	Representative Paper	Image Complexity	Interactivity	Speed	Memory Efficiency
Narrow Band	[5]	★☆☆☆☆	★★★★☆	★★★★★	★★★★★
Mumford-Shah	[16]	★★★☆☆	★☆☆☆☆	★★☆☆☆	★★☆☆☆
Seed Sketching	[17]	★★☆☆☆	★★★★☆	★★★★☆	★★★★☆
Clustering (superpixel)	[18]	★★☆☆☆	★☆☆☆☆	★★★★★	★★☆☆☆
Active Contours (GVF)	[19]	★☆☆☆☆	★★★☆☆	★★☆☆☆	★★☆☆☆
Active Contour (interactive)	[8]	★★★☆☆	★★★★★	★★★☆☆	★★★☆☆
Region Growing	[20]	★☆☆☆☆	★★★☆☆	★★★☆☆	★★★☆☆
Watershed	[21]	★★☆☆☆	★☆☆☆☆	★★★★☆	★★☆☆☆
Graph Cuts	[22]	★★★★★	★★★☆☆	★★★★☆	★☆☆☆☆
Active Shape Model	[23]	★☆☆☆☆	★★☆☆☆	★★☆☆☆	★★★☆☆
IGAC (single seed)	(ours)	★★★★☆	★★★☆☆	★★★☆☆	★★☆☆☆
IGAC (with brushes)	(ours)	★★★★★	★★★★★	★★★☆☆	★★☆☆☆

TABLE 1: Compact literature review of current GPU segmention methods. The scores are derived subjectively by testing available implementations where possible on a variety of 2D and 3D datasets.

traditional narrow band algorithm [34]. More recent extensions classify the active region using simple operations on the spatial and temporal derivatives of the level set function [5], 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 [4], [5], [35]. In contrast, the model proposed by [6] 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, their model is derived from [29] who introduce C^∞ regularization of the Heaviside and Dirac functions which are non-zero everywhere, unlike the C^2 regularized Heaviside (proposed in [36]) which is non-zero 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 [12]. Traditional methods rely on simple gradient energy, which converges to local minima, however [37] introduced a diffusion of the gradient vectors called gradient vector flow (GVF) to address this problem. [19] 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 [38]. 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 [29].

Ever since the original snakes paper, active contours have gained popularity through being able to interactively edit the contour, or setup constraints to guide its motion [12]. Region-based active contour methods provide the option to initialize with a simple primitive shape, or sketch a starting region [17]. The more advanced approach by [40] 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 [8] provides an interactive interface with geodesic active contours [41] and region competition [42]. Region competition favors a well-defined intensity range, whereas the geodesic approach is better suited for images with clear edges; by combining both approaches [8] can segment a broad range of images. However, it requires significant tuning and can still fail in complex images with neither a well-defined intensity range nor clear edges.

The influential public datasets with groundtruth segmentations (such as BSDS, MSRC, iCoseg, FlickrMFC, SegTrack) include videos or 2D images with color information such as cars, chairs, and people. Of these, the interactive approaches take as input a set of scribbles where objects follow similar color distributions [10]. Graph cut segmentation is popular in this field, where [22], [43] 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 [14]. Unfortunately, there is still a need for benchmark medical datasets with well-defined interactive performance evaluation [44].

There are several GPU approaches that produce a segmentation without relying on initialization of a seed region [13]. Clustering methods join regions of a high-dimensional feature space [45] and superpixel approaches [18] form clusters that are deliberately over-segmented into more manageable regions. These approaches are good at simplifying complex images, however they are poor at segmenting a specific object of interest, or sets of objects, from the background. 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 [6], 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 [29].

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

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

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 [46], 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 $\vec{x}$:

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

where $\omega(\vec{y} - \vec{x})$ is a Gaussian weighting function centered on $\vec{x}$, $p_{1,\vec{x}}$ is a Gaussian approximation of the intensity distribution for the part of the neighborhood of $\vec{x}$ lying outside the contour (and inside for $p_{2,\vec{x}}$), 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 $\vec{x}$ 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 $u_i(\vec{x})$, $\sigma_i(\vec{x})$ where $i \in \{1, 2\}$ for regions outside and inside the contour, respectively:

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

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

Specifically, they express for each pixel the mean and variance of neighboring grey 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 [46] to ensure numerical stability [47]:

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

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

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

where H is the C^∞ regularized Heaviside function, discretized to operate on a regular grid, first proposed by [29]:

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

The total energy functional (Equation 1) can be minimized by applying the calculus of variations [6] 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 $\delta(x) = H'(x)$ [29], λ_1, λ_2, ν and μ are parameters controlling the weight of the terms, and κ is the contour's local curvature [27]:

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

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

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

The data fitting term $e_1(\vec{x})$ quantifies how badly the pixel $\vec{x}$ would fit with the outside-contour parts of its neighbors' neighborhoods. When e_1 is high and $\vec{x}$ 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 $\vec{x}$. The same applies in reverse for e_2.

Due to the smooth form of the C^∞ regularized Heaviside (Equation 7), $\delta(\phi) = H'(\phi)$ is non-zero 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 [29].

3.1 GPU Implementation

The goal of the implementation is to iteratively solve Equation 8 for $\phi(\vec{x}, t)$, visualizing the results at each iteration. This is done by discretizing ϕ with respect to time and applying numerical integration: starting with $\phi(\vec{x}, t = 0)$ (which is specified by the user), an update loop computes $\phi(\vec{x}, t + \Delta t)$ by computing $\frac{\partial\phi}{\partial t}$ according to Equation 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 to port 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 Equation 8 requires convolutions (Equation 10) of intermediate variables that themselves rely on other convolutions (Equations 3-4). The relationships of these variables are shown in Figure 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 (Equation 7). All variables of the form GX represent the n-dimensional Gaussian convolution of X, where X must be computed and stored as a texture before GX can be computed. This is because we wish to use GPU texture memory, which has spatial locality in multiple dimensions, however texture memory must either be read-only or write-only within a given kernel function and therefore results computed from data in a texture buffer must be written to a different buffer.

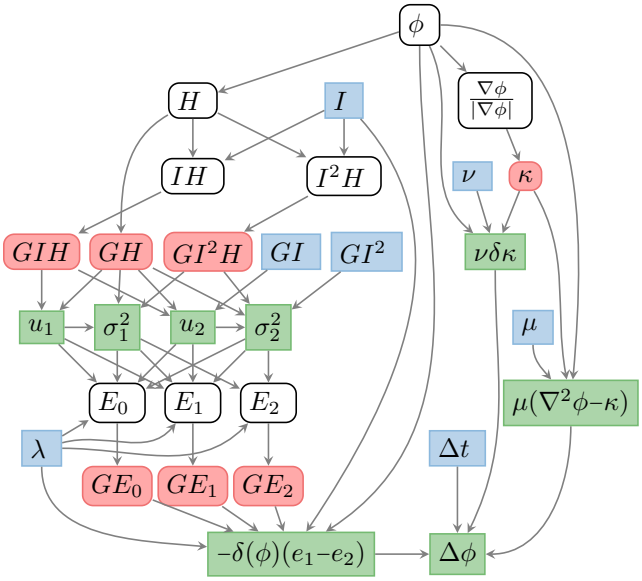


Fig. 2: Dependency graph between variables in the update process. The red variables require neighborhood computations whereas the blue variables represent constants. All variables except for the parameters ν, μ, λ and Δt are spatially varying fields. The green variables are quantities that are computed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

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

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

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

For σ_i^2 we have used the alternative variance formula $\mathrm{Var}[X] = E[X^2] - E[X]^2$, 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 the above 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 Equation 10 to get:

$$\begin{aligned} e_i(\vec{x}) = &\int_\Omega \omega(\vec{y} - \vec{x}) \left[\log(\sigma_i(\vec{y})) + \frac{u_i(\vec{y})^2}{2\sigma_i(\vec{y})^2}\right] \mathrm{d}\vec{y} \\ &- I(\vec{x}) \int_\Omega \omega(\vec{y} - \vec{x}) \frac{u_i(\vec{y})}{\sigma_i(\vec{y})^2} \,\mathrm{d}\vec{y} \\ &+ I(\vec{x})^2 \int_\Omega \omega(\vec{y} - \vec{x}) \frac{1}{2\sigma_i(\vec{y})^2} \,\mathrm{d}\vec{y} \qquad (13) \\ = &\, G_\sigma * \left[\log(\sigma_i(\vec{y})) + \frac{u_i(\vec{y})^2}{2\sigma_i(\vec{y})^2}\right] \\ &- I(\vec{x}) \left[G_\sigma * \frac{u_i(\vec{y})}{\sigma_i(\vec{y})^2}\right] + I(\vec{x})^2 \left[G_\sigma * \frac{1}{2\sigma_i(\vec{y})^2}\right] \qquad (14) \end{aligned}$$

To compute the three terms in Equation 14, we first precompute the operands of the Gaussian convolutions (E_0, E_1 and E_2 in Figure 2), storing them as textures, then convolve them (GE_0, GE_1 and GE_2 in Figure 2), then weight them by 1, I and I^2 and sum them. 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 $e_1 - e_2$. The memory layout of our kernels is shown in Figure 3, which lists our kernels in the order they are called and shows their inputs and outputs (corresponding to the nodes in Figure 2) within the available 4×32-bit channels per GPU texture buffer.

Gaussian convolutions require a large number of samples from texture memory, however an n-dimensional Gaussian filter can be separated into the matrix product of n vectors allowing us to convolve with n 1-dimensional 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. This is why

Preprocess 3D

	A				B				C			
	x	y	z	w	x	y	z	w	x	y	z	w
CPU			ϕ	I								
Prepare			ϕ	I					I	I^2		
X Gaussian			ϕ	I	GI	GI^2			I	I^2		
Y Gaussian			ϕ	I	GI	GI^2			GI	GI^2		
Z Gaussian			ϕ	I	GI	GI^2			GI	GI^2		
Compose			ϕ	I	GI	GI^2			GI	GI^2	ϕ	I
Neumann/Copy	GI	GI^2	ϕ	I	GI	GI^2			GI	GI^2	ϕ	I

Update 3D

	A				B				C			
	x	y	z	w	x	y	z	w	x	y	z	w
Prev Iteration	GI	GI^2	ϕ	I								
Normalised Grad	GI	GI^2	ϕ	I					$\nabla(\phi)_x$	$\nabla(\phi)_y$	$\nabla(\phi)_z$	ϕ
Prep Conv 1	GI	GI^2	ϕ	I	IH	H	I^2H	κ	$\nabla(\phi)_x$	$\nabla(\phi)_y$	$\nabla(\phi)_z$	ϕ
X Gaussian	GI	GI^2	ϕ	I	IH	H	I^2H	κ	GIH	GH	GI^2H	κ
Y Gaussian	GI	GI^2	ϕ	I	GIH	GH	GI^2H	κ	GIH	GH	GI^2H	κ
Z Gaussian	GI	GI^2	ϕ	I	GIH	GH	GI^2H	κ	GIH	GH	GI^2H	κ
Prep Conv 2	GI	GI^2	ϕ	I	E_0	E_1	E_2	κ	GIH	GH	GI^2H	κ
X Gaussian	GI	GI^2	ϕ	I	E_0	E_1	E_2	κ	GE_0	GE_1	GE_2	κ
Y Gaussian	GI	GI^2	ϕ	I	GE_0	GE_1	GE_2	κ	GE_0	GE_1	GE_2	κ
Z Gaussian	GI	GI^2	ϕ	I	GE_0	GE_1	GE_2	κ	GE_0	GE_1	GE_2	κ
Update ϕ	GI	GI^2	ϕ	I	GI	GI^2	ϕ'	I	GE_0	GE_1	GE_2	κ
Neumann/Copy	GI	GI^2	ϕ'	I	GI	GI^2	ϕ'	I	GE_0	GE_1	GE_2	κ

Read Only (input) Write Only (output)

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).

we have separate kernels for the X Gaussian, Y Gaussian, and Z Gaussian convolutions in Figure 3 accordingly. The three Gaussian convolutions of the image and Heaviside (GIH, GH, GI^2H, Figure 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 Figure 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 (Equation 9) requires all two (three in 3D) 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.

3.2 Interactive Brushes

There are many applications in the biosciences, computer vision, medical, and pattern recognition communities where guidance by human experts is required [8], [12], [17], [40], [48]. The current interactive GPU level set methods, such as [5], 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 [40], [43] 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 Figure 4 top-left and in the accompanying video).

To address these issues, we follow the strategies outlined in the survey [49] with similar functions to the modeling/graphics literature [50], 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 (Equation 8), and visualize the results after each iteration. Note: 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 [50].

All brush functions are centered at the mouse position $\vec{p}$ with radius r, and are implemented in the 'Compose' kernel (Figure 3). We have deliberately arranged the read buffer B to link to ϕ from the previous update iteration. To complete a brush action, we reluanch 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):

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

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

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

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

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

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:

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

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

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.

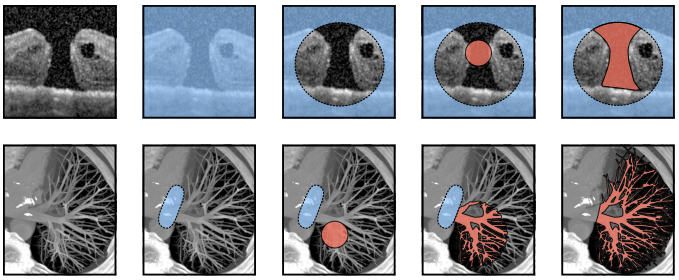


Fig. 4: Figure illustrating interactive use of our brush functions. The blue region represents the barrier brush $\phi = \infty$ and red regions are where $\phi < 0$ and otherwise $\phi > 0$.

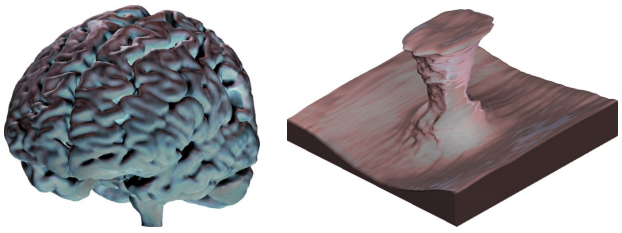
In Figure 4 we illustrate two simple use-cases of our interactive brushes. In the top row, the user paints using the 'barrier' brush to cover the full image region, shown in blue. This is followed by the 'erase' brush (Equation 17), to cut a permissible region in which a new seed region is placed (Equation 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 results in Table 6 2b-c).

3.3 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 (Figure 3). We send a camera matrix to initialize each pixel with a ray origin $\vec{o}$ and direction unit vector $\hat{d}$. We parameterize the ray's position by $\vec{r} = \vec{o} + \hat{d}s$ and, assuming ϕ to be the signed distance to the zero-crossing, advance the ray in steps by $s_{i+1} = s_i + \phi(\vec{r})$. 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 practice we find we can obtain sufficiently small visual artifacts at good performance by choosing a constant step size $\Delta s = 0.3\phi(\vec{r})$. 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 [51]). 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 [52].

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



(a) 3D segmented Brain (b) 3D segmented Macular Hole

Fig. 5: 3D views during segmentation rendered in real-time.

4 Results & 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 segmentation of the original CPU implementation and the result from our GPU kernels in 6 image types, and show the results in Table 2.

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

Image	Jaccard index
Synthetic Objects 2D	1
Tumour (small) 2D	1
Tumour (large) 2D	0.981
Macular Hole 3D	0.990
Brain 3D	0.984
Tumour 3D	0.993

TABLE 2: Comparing the Jaccard index of our GPU algorithm with the original LGDF energy model.

4.1 Noise & 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 Figure 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 3 from the same synthetic 2D object, and for a 3D macular hole [53].

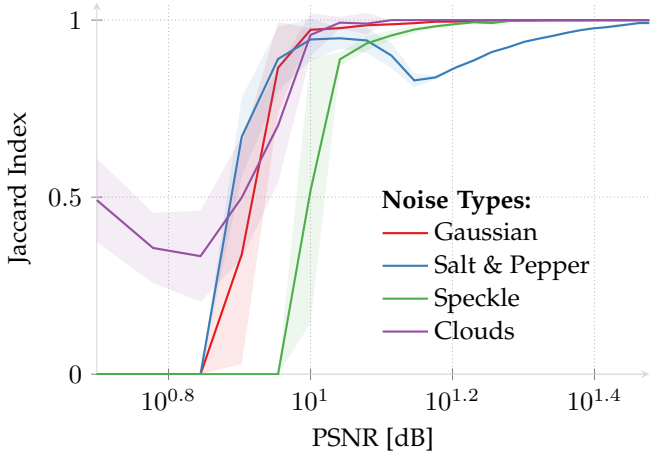


Fig. 6: The Jaccard index of a synthetic ground-truth segmentation and our segmentation result using the same parameters on 4 different types of noise. The standard deviation is shown by the error envelopes (transparent shaded regions); our method is robust to several noise types heavily corrupting the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

The results in Figure 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 Figure 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 & pepper, and speckle noise, we implemented a multi-frequency 'cloud' noise at a target PSNR, which simulates intensity inhomogeneity. In Figure 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

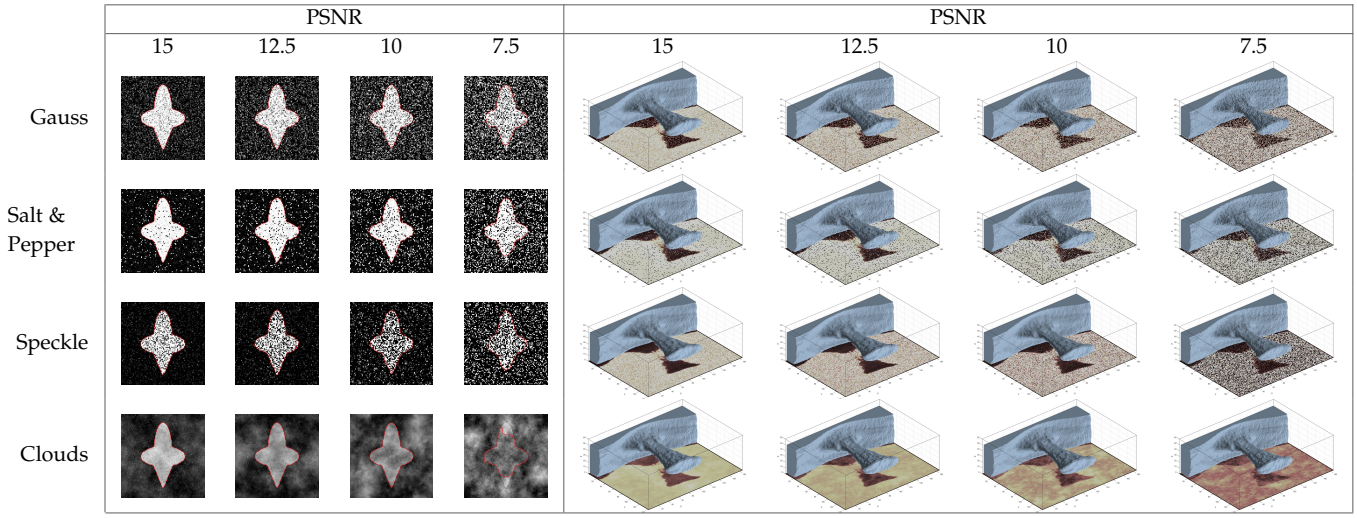


TABLE 3: Segmentation without interactive brushes attained from a single circular seed region inside the object.

following defaults: $\sigma = 3$, $\nu = 50$, $\lambda_1 = 1$, $\lambda_2 = 1.05$, $\Delta t = 0.1$, $\mu = 1$ (these are the parameters used in Figure 6 and Table 3). 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, ν, 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 4:

Description	Symbol	Suggested Range	Default
Capture Range	σ	$[1.01,\ 10]$	3
Smoothing Weight	ν	$[10,\ 90]$	50
Shrink or Grow	λ	$[-0.1,\ 0.1]$	0.05

TABLE 4: 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 $\mu = 1.0$.

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 Equations 2-4), and therefore determines the capture range. The parameter ν penalizes the length of the contour (Equation 6 and 8); a larger ν 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 behaviour by weighting the image terms e_1 and e_2 by λ_1 and λ_2 respectively (Equation 8), biasing the contour towards 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 ν.

To further justify the importance of our interactive brushes, we construct 6 extreme synthetic scenarios in Table 5. Images 1-3 show Gaussian, salt & pepper, and cloud noise corrupted to a severe PSNR of 5 (fail cases in Figure 6). By adjusting the parameters and constraining the contour with our brushes, we can easily (3-5 seconds 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.

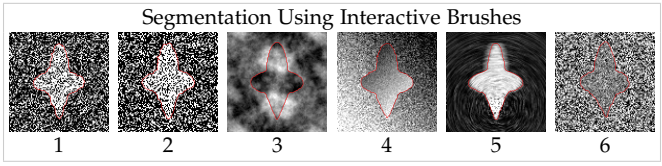


TABLE 5: The following challenging scenarios are quickly & easily segmented with our interactive brushes. Images 1-5 have a PSNR of 5 for Gaussian, salt & pepper, and multi-frequency noise accordingly, and images 4-6 show extreme scenarios of poorly defined and/or blurred boundaries.

4.2 Segmenting Real-world Images

We evaluate our software against several different imaging modalities on real-world data and show the results in Table 6. In all our results, we only adjust the parameters σ, ν, and λ as described in Table 4. By initializing $\phi(\vec{x}) = 2$ uniformly, we are able to automatically segment some objects without an initial seed region, such as the cells in Table 6 1c, and some of the small objects in 1a. 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. Similarly, we segmented the bronchioles inside the lungs in 2b without a seed region, however this also captured some other small objects in the image which we erased using the interactive brushes. The remaining images were segmented by painting a simple region inside the object of interest, and using the default grow parameter $\lambda = 0.05$. We found, as with our synthetic experiment, that the proposed parameters are insensitive to the different levels and types of noise encountered in the different imaging modalities. In general the default parameters suggested in Table 4 work well for most object segmentations, however in challenging cases

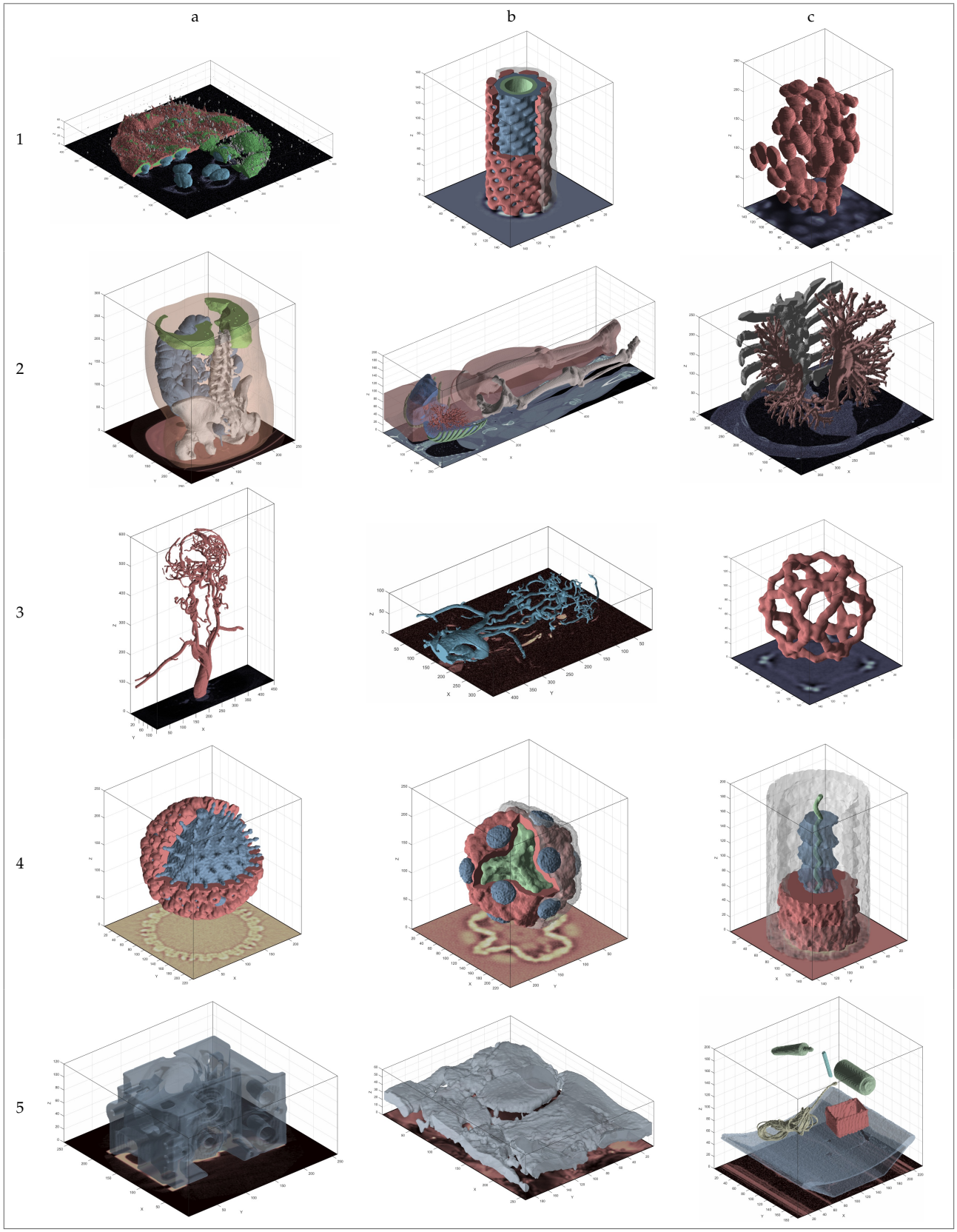


TABLE 6: Segmentation results of multiple objects displayed in different colors. 1a shows a segmented image of HaCaT human cell culture cells using confocal microscopy, 1b shows the interdigitation of segmented layers of eisosome proteins from cryo-EM tomography data [54], 1c shows selective plane illumination microscopy (SPIM) of zebrafish eye lens cells [55]. Row 2 shows medical CT scans of the abdomen, body, and thorax [2]. 3a shows an MRI of a cerebral aneurysm, and 3b an XA angiogram [2]. 3c shows the structure of the Sec13/31 COPII coat cage from cryo-EM data [56]. Row 4 shows the herpes simplex virus capsid [57], phi procapsid [58], and the mumps virus [59], all from cryo-EM data. 5a shows a CT scan of an engine block [60], 5b sintered alumina [3], and 5c shows a selection of objects from a CT scan of a backpack [60].

(such as multiple objects or thin objects) the parameters σ and ν 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. In the baggage data in 5c there are many other objects with the same intensity touching the rope and inside the box. For the purpose of demonstration, we used the barrier brush to prevent segmenting these other objects despite them strongly touching or overlapping the segmented rope and box with the same intensity.

Many of these segmentations, such as in Table 6 1a, 1c, 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 [4], [5], [35]. For example, when painting an initial seed region at the base of the aneurysm image in 3b, the active contour will not grow beyond approximately 100 voxels in the y-axis due to intensity inhomogeneity along the vessel. In contrast, the adopted LGDF energy model proposed by [6] 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 restricting to a local solution with a high-level of visual feedback, we can spot such issues and make ammendments 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 Figure 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 non-zero everywhere.

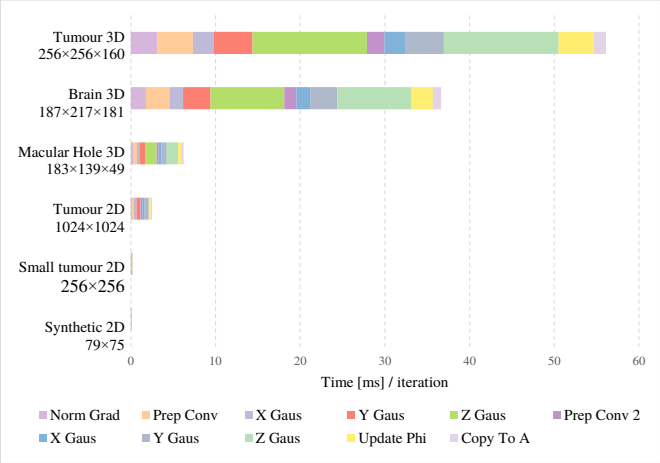


Fig. 7: Mean kernel timings over 100 frames for different images of different sizes. $\sigma = 3$ in all cases. Despite using texture memory, which is cached and has spatial locality in multiple dimensions [15], and fast constant memory to store the 1D separable Gaussian coefficients, convolution in the z-axis is significantly slower than the y and x axes.

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 4), 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.

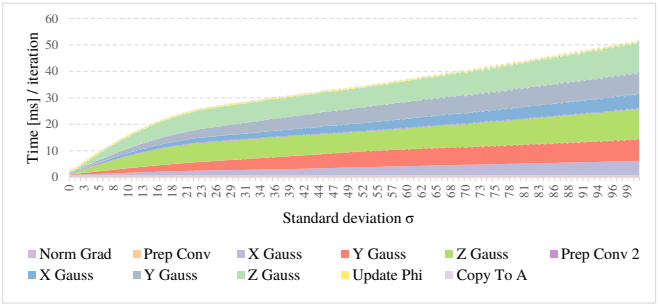


Fig. 8: Mean kernel timings over 100 frames with increasing σ for 3D macular hole. In practice we rarely require $\sigma > 10$.

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, 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 Figure 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 CPU MATLAB version (which is vectorized and calls code written in C for the Gaussian convolution). These results are shown in Figure 9.

In Figure 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.

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 Figure 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 non-zero everywhere to reduce convergence on local minima [29]. 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^∞

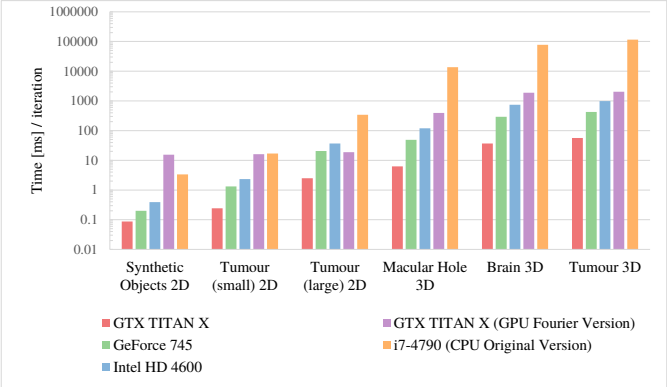


Fig. 9: Mean time [ms] over 100 iterations on different GPU hardware, compared to the original MATLAB implementation and our implementation using fast Fourier convolution on the GPU. Our OpenCL implementation with separable convolutions achieves over a $\times$1,000 speedup over the original vectorized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

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 [10], the problem of segmenting a human or plant, 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 inhomogeneity, or poorly defined edges rather than texture information or differing backgrounds. As with, [44] we would like to see benchmark 3D biological and medical datasets for evaluating interactive performance.

In the future, we hope to use our software in the creation of such datasets with groundtruth segmentations, multiple initializations, and a collection of interactive metrics for comparative studies.

6 Conclusion

In conclusion, we have shown that sophisticated level set segmentation energy models, with sequential dependencies amongst 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 [6] 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 biological, medical, and engineering scenarios. 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 (Github link will be included), alongside an optional MATLAB wrapper.

Acknowledgments

We are grateful to NVIDIA for providing a GTX TITAN X for this research. Table 7 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, School of Biological and Biomedical sciences, Durham University.

References

[1] C. A. Cocosco, V. Kollokian, R. K.-S. Kwan, G. B. Pike, and A. C. Evans, "BrainWeb: Online interface to a 3D MRI simulated brain database," *NeuroImage*, vol. 5, p. 425, 1997.

[2] A. Rosset, L. Spadola, and O. Ratib, "Osirix: An open-source software for navigating in multidimensional dicom images," *Journal of Digital Imaging*, vol. 17, no. 3, pp. 205–216, 2004.

[3] J. Schindelin, I. Arganda-Carreras, E. Frise, V. Kaynig, M. Longair, T. Pietzsch, S. Preibisch, C. Rueden, S. Saalfeld, B. Schmid, J.-Y. Tinevez, D. J. White, V. Hartenstein, K. Eliceiri, P. Tomancak, and A. Cardona, "Fiji: an open-source platform for biological-image analysis," *Nature Methods*, vol. 9, no. 7, pp. 676–682, 2012.

[4] W. K. Jeong, J. Beyer, M. Hadwiger, A. Vazquez, H. Pfister, and R. T. Whitaker, "Scalable and interactive segmentation and visualization of neural processes in em datasets," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 1505–1514, 2009.

[5] M. Roberts, J. Packer, M. C. Sousa, and J. R. Mitchell, "A work-efficient GPU algorithm for level set segmentation," in *Proceedings of the Conference on High Performance Graphics*. Eurographics Association, 2010, pp. 123–132.

[6] L. Wang, L. He, A. Mishra, and C. Li, "Active contours driven by local Gaussian distribution fitting energy," *Signal Processing*, vol. 89, no. 12, pp. 2435–2447, 2009.

[7] L. Zhu, P. Karasev, I. Kolesov, R. Sandhu, and A. Tannenbaum, "Interactive Image Segmentation From A Feedback Control Perspective," *ArXiv e-prints*, Jun. 2016.

[8] P. A. Yushkevich, J. Piven, H. C. Hazlett, R. G. Smith, S. Ho, J. C. Gee, and G. Gerig, "User-guided 3D active contour segmentation of anatomical structures: significantly improved efficiency and reliability," *Neuroimage*, vol. 31, no. 3, pp. 1116–1128, 2006.

[9] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14539

[10] H. Zhu, F. Meng, J. Cai, and S. Lu, "Beyond pixels: A comprehensive survey from bottom-up to semantic image segmentation and cosegmentation," *Journal of Visual Communication and Image Representation*, vol. 34, pp. 12 – 27, 2016.

[11] A. Chambolle and T. Pock, "A first-order primal-dual algorithm for convex problems withapplications to imaging," *Journal of Mathematical Imaging and Vision*, vol. 40, no. 1, pp. 120–145, 2011.

[12] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: active contour models," *International Journal of Computer Vision*, vol. 1, no. 4, pp. 321–331, 1988.

[13] M. Li, C. He, and Y. Zhan, "Adaptive level-set evolution without initial contours for image segmentation," *Journal of Electronic Imaging*, vol. 20, no. 2, pp. 023 004–023 004, 2011.

[14] A. Eklund, P. Dufort, D. Forsberg, and S. M. LaConte, "Medical image processing on the GPU past, present and future," *Medical Image Analysis*, vol. 17, no. 8, pp. 1073–1094, 2013.

[15] E. Smistad, T. L. Falch, M. Bozorgi, A. C. Elster, and F. Lindseth, "Medical image segmentation on GPUs a comprehensive review," *Medical Image Analysis*, vol. 20, no. 1, pp. 1–18, 2015.

[16] T. Pock, D. Cremers, H. Bischof, and A. Chambolle, "An algorithm for minimizing the Mumford-Shah functional," in *IEEE International Conference on Computer Vision*, 2009, pp. 1133–1140.

[17] H. L. J. Chen, F. F. Samavati, M. C. Sousa, and J. R. Mitchell, "Sketch-based volumetric seeded region growing," in *Proceedings of the Third Eurographics Conference on Sketch-Based Interfaces and Modeling*, 2006, pp. 123–130.

[18] C. Y. Ren and I. Reid, "gSLIC: a real-time implementation of SLIC superpixel segmentation," University of Oxford, Department of Engineering, Technical Report, Tech. Rep., 2011.

[19] Z. He and F. Kuester, "GPU-based active contour segmentation using gradient vector flow," in *International Conference on Advances in Visual Computing*, 2006, pp. 191–201.

[20] C. Kauffmann and N. Piche, "Cellular automaton for ultra-fast watershed transform on GPU," in *International Conference on Pattern Recognition*, 2008, pp. 1–4.

[21] L. Pan, L. Gu, and J. Xu, "Implementation of medical image segmentation in cuda," in *International Conference on Information Technology and Applications in Biomedicine*, 2008, pp. 82–85.

[22] V. Vineet and P. Narayanan, "CUDA cuts: Fast graph cuts on the GPU," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2008, pp. 1–8.

[23] S. A. Mahmoudi, F. Lecron, P. Manneback, M. Benjelloun, and S. Mahmoudi, "GPU-based segmentation of cervical vertebra in X-ray images," in *IEEE International Conference on Cluster Computing Workshops and Posters*, 2010, pp. 1–8.

[24] Y. Tian, F. Duan, M. Zhou, and Z. Wu, "Active contour model combining region and edge information," *Machine Vision and Applications*, vol. 24, no. 1, pp. 47–61, 2013.

[25] K. Sun, Z. Chen, and S. Jiang, "Local morphology fitting active contour for automatic vascular segmentation," *IEEE Transactions on Biomedical Engineering*, vol. 59, no. 2, pp. 464–473, 2012.

[26] S. Osher and R. Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces*, ser. Applied Mathematical Sciences. Springer New York, 2002.

[27] S. Osher and J. A. Sethian, "Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations," *Journal of Computational Physics*, vol. 79, no. 1, pp. 12–49, 1988.

[28] R. Malladi, J. A. Sethian, and B. C. Vemuri, "Shape modeling with front propagation: A level set approach," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 17, no. 2, pp. 158–175, 1995.

[29] T. F. Chan, L. Vese *et al.*, "Active contours without edges," *Image processing, IEEE transactions on*, vol. 10, no. 2, pp. 266–277, 2001.

[30] D. Mumford and J. Shah, "Optimal approximations by piecewise smooth functions and associated variational problems," *Communications on Pure and Applied Mathematics*, vol. 42, no. 5, pp. 577–685, 1989.

[31] L. Gorelick, F. R. Schmidt, and Y. Boykov, "Fast trust region for segmentation," in *2013 IEEE Conference on Computer Vision and Pattern Recognition*, June 2013, pp. 1714–1721.

[32] G. Pratx and L. Xing, "GPU computing in medical physics: A review," *Medical Physics*, vol. 38, p. 2685, 2011.

[33] L. Shi, W. Liu, H. Zhang, Y. Xie, and D. Wang, "A survey of GPU-based medical image computing techniques," *Quantitative Imaging in Medicine and Surgery*, vol. 2, no. 3, pp. 2223–2292, 2012.

[34] D. Adalsteinsson and J. A. Sethian, "A fast level set method for propagating interfaces," *Journal of Computational Physics*, vol. 118, no. 2, pp. 269–277, 1995.

[35] A. E. Lefohn, J. M. Kniss, C. D. Hansen, and R. T. Whitaker, "A streaming narrow-band algorithm: interactive computation and visualization of level sets," *IEEE Transactions on Visualization and Computer Graphics*, vol. 10, no. 4, pp. 422–433, 2004.

[36] H.-K. Zhao, T. Chan, B. Merriman, and S. Osher, "A variational level set approach to multiphase motion," *Journal of computational physics*, vol. 127, no. 1, pp. 179–195, 1996.

[37] C. Xu and J. L. Prince, "Snakes, shapes, and gradient vector flow," *IEEE Trans. on Image Processing*, vol. 7, no. 3, pp. 359–369, 1998.

[38] E. Smistad, A. C. Elster, and F. Lindseth, "Real-time gradient vector flow on GPUs using OpenCL," *Journal of Real-Time Image Processing*, vol. 10, no. 1, pp. 67–74, 2012.

[39] J. Schmid, J. A. Iglesias-Guitián, E. Gobbetti, and N. Magnenat-Thalmann, "A GPU framework for parallel segmentation of volumetric images using discrete deformable models," *The Visual Computer*, vol. 27, no. 2, pp. 85–95, 2010.

[40] B. Mory, "Interactive Segmentation of 3D Medical Images with Implicit Surfaces," Ph.D. dissertation, STI, Lausanne, 2011.

[41] V. Caselles, F. Catté, T. Coll, and F. Dibos, "A geometric model for active contours in image processing," *Numerische Mathematik*, vol. 66, no. 1, pp. 1–31, 1993.

[42] S. C. Zhu and A. Yuille, "Region competition: Unifying snakes, region growing, and Bayes/MDL for multiband image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, no. 9, pp. 884–900, 1996.

[43] L. Grady, "Random walks for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 11, pp. 1768–1783, Nov 2006.

[44] F. Zhao and X. Xie, "An overview of interactive medical image segmentation," *Annals of the BMVA*, vol. 2013, no. 7, pp. 1–22, Apr 2013.

[45] B. Fulkerson and S. Soatto, "Really quick shift: Image segmentation on a GPU," in *Trends and Topics in Computer Vision*, 2010, pp. 350–358.

[46] C. Li, C. Xu, C. Gui, and M. D. Fox, "Level set evolution without re-initialization: a new variational formulation," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, 2005, pp. 430–436.

[47] D. Peng, B. Merriman, S. Osher, H. Zhao, and M. Kang, "A PDE-based fast local level set method," *Journal of Computational Physics*, vol. 155, no. 2, pp. 410–438, 1999.

[48] R. Whitaker, D. Breen, K. Museth, and N. Soni, *Segmentation of Biological Volume Datasets Using a Level-Set Framework*. Vienna: Springer Vienna, 2001, pp. 249–263.

[49] S. Olabarriaga and A. Smeulders, "Interaction in the segmentation of medical images: A survey," *Medical Image Analysis*, vol. 5, no. 2, pp. 127–142, 2001.

[50] M. Eyiyurekli and D. Breen, "Interactive free-form level-set surface-editing operators," *Computers & Graphics*, vol. 34, no. 5, pp. 621–638, 2010.

[51] T. L. Kay and J. T. Kajiya, "Ray tracing complex scenes," in *Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH. ACM, 1986, pp. 269–278.

[52] A. Evans, "Fast approximations for global illumination on dynamic scenes," in *ACM SIGGRAPH 2006 Courses*. ACM, 2006, pp. 153–171.

[53] D. H. W. Steel and A. J. Lotery, "Idiopathic vitreomacular traction and macular hole: a comprehensive review of pathophysiology, diagnosis, and treatment," *Eye*, vol. 27, no. 1, pp. 1–21, 2013.

[54] L. Karotki, J. T. Huiskonen, C. J. Stefan, N. E. Zikowska, R. Roth, M. A. Surma, N. J. Krogan, S. D. Emr, J. Heuser, K. Grnewald, and T. C. Walther, "Eisosome proteins assemble into a membrane scaffold," *The Journal of cell biology*, vol. 195, no. 5, pp. 889–902, 2011.

[55] M. Jarrin, L. Young, W. Wu, J. M. Girkin, and R. A. Quinlan, "Chapter twenty-one - in vivo, ex vivo, and in vitro approaches to study intermediate filaments in the eye lens," in *Intermediate Filament Proteins*, ser. Methods in Enzymology, M. B. Omary and R. K. Liem, Eds. Academic Press, 2016, vol. 568, pp. 581 – 611.

[56] S. M. Stagg, C. Grkan, D. M. Fowler, P. LaPointe, T. R. Foss, C. S. Potter, B. Carragher, and W. E. Balch, "Structure of the Sec13/31 COPII coat cage," *Nature*, vol. 439, no. 7073, p. 234238, 2006.

[57] J. T. Chang, M. F. Schmid, F. J. Rixon, and W. Chiu, "Electron cryotomography reveals the portal in the herpesvirus capsid," *Journal of virology*, vol. 81, no. 4, pp. 2065–2068, 2007.

[58] A. Sen, J. B. Heymann, N. Cheng, J. Qiao, L. Mindich, and A. C. Steven, "Initial location of the RNA-dependent RNA polymerase in the bacteriophage Phi6 procapsid determined by cryo-electron microscopy," *The Journal of biological chemistry*, vol. 283, no. 18, pp. 12 227–12 231, 2008.

[59] R. Cox, A. Pickar, S. Qiu, J. Tsao, C. Rodenburg, T. Dokland, A. Elson, B. He, and M. Luo, "Structural studies on the authentic mumps virus nucleocapsid showing uncoiling by the phosphoprotein," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 111, no. 42, pp. 15 208–15 213, 2014.

[60] D. Bartz, "Volvis datasets," http://www.volvis.org, 2005, accessed: 2016-03-30.

Chris G. Willcocks received a PhD in Computer Science at Durham University in 2013 where he specialized in real-time GPGPU rendering and deformation of large volumetric datasets. He researched object deformation at Newcastle University Game Lab before accepting a post-doctoral 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 computer graphics and bioimage informatics.

Philip T. G. Jackson received a BSc degree in Computer Science & 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 investigates the use of recurrent neural networks for multi-object localisation.

Carl J. Nelson received an MSci degree in Biology & Physics within the Natural Sciences Programme from Durham University and is currently reading for a PhD degree in Computing Science, also from Durham University, UK.

His interdisciplinary research focuses on developing advanced and novel quantification techniques for bioimaging. The tools are used to further the understanding of biological processes that can be gleaned through bioimaging techniques.

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 a senior lecturer in Computer Science at the University of Durham, UK.

His interdisciplinary research focuses on advancing the state of the art in bioimage informatics technologies aimed at a better understanding of the complex biological processes, from nano to macro.