Slovak University of Technology in Bratislava

Faculty of Civil Engineering

Department of Mathematics and Descriptive Geometry

Analysis of 3D and 4D Images of Organisms in Embryogenesis

PhD. student: **Ing. Michal Smíšek**

Supervisor: **Prof. RNDr. Karol Mikula, DrSc.**

Bratislava, May 2015

Acknowledgment

Scientific work at the size of a dissertation thesis is never a one-man show. This section serves as an acknowledgment to those, who stood beside me and help me to accomplish this work to the point where it is today.

First and foremost, I would like to express my gratidude towards the supervisor of my thesis, Prof. Karol Mikula. Not long after I first met him, he restored my faith in the possibility of performing a world-class scientific research in an often not-too-ideal Slovak environment. He taught me how to use both mathematics and programming as engineering and scientific tools, and for the last 7 years, he has been continuously providing a generous share of his skill and expertise to me.

Secondly, I would like to thank the staff and collaborators of the Institute of Mathematics and Descriptive Geometry at the Faculty of Civil Engineering, Slovak Technical University, for their open-mindedness and help. Thanks to Assoc. Prof. Peter Frolkovič, for helping me to achieve meaningful results in just about all the major topics presented in this work. Thanks to Assoc. Prof. Mariana Remešíková for being with me at the first days of my scientific career. Thanks to Assoc. Prof. Angela Handlovičová for teaching me how to add zeroes and multiply by ones in mathematical proofs (and other stuff). Thanks to Prof. Gabriel Okša for serving as a constructive opponent in my dissertation exam and thanks to Prof. Miloš Šrámek for providing me with his experience especially in the field of distance function - with help of these gentlemen, I was able to turn an idea into a solution. Thanks to Prof. Nadine Peyriéras and Dr. Francois Graner for their inputs from the field of their expertise - developmental biology.

Then, I would like to thank my classmates and fellow PhD. students: Jozef Urbán, for providing creativity and help in the process of design of the novel distance function algorithms and Róbert Špir, for designing and implementing a sound application to justify the viability of these algorithms.

I would also like to thank my collaborator from the German Aerospace Institute, Dr. Klaus Strobl, for teaching me how to think like an engineer and how to

get the job done.

Globalization and Internet were other vital helps. I would like to specifically thank the Massachusetts Institute of Technology for providing top-quality courses and sharing them publicly, free of charge, via their OpenCourseWare program. Namely, thanks to Prof. Gilbert Strang and Prof. Arthur Mattuck for their applied mathematics courses, Prof. Walter Lewin for his courses on physics, and thanks to Prof. Charles Leiserson, Prof. Erik Demaine, Jonathan Blow and Casey Muratori for teaching me how to think in code.

All these achievements wouldn't be possible without constant love, care and support of my family: dad Miroslav, mum Eva, sister Michaela and the love of my life, Paulína Ondrášiková.

1 Abstract

In this work, we present a few modifications to the state-of-the-art algorithms, as well as several novel approaches, related to the detection of cells in biological image processing.

We start by explanation of a PDE-based image processing evolution called FBLSCD and study its properties. We then define a fully automatic way of finding the stop time for this evolution. Afterwards, we try to see the FBLSCD as a morphological grayscale erosion, and we formulate a novel cell detection algorithm, called LSOpen, as an intersection of PDE-based and morphological image processing schools.

Then, we discuss the best ways of inspecting cell detection results, i.e. cell identifiers. We try to quantitatively benchmark various cell detection methods by the relative amount of false positives, false negatives and multiply-detected centers yielded. We will observe that comparing cell detection results in a binary fashion is insufficient, therefore we are going to utilize the concept of distance function.

Motivated by this need for robust cell detection result comparison, we analyze commonly-used methods for computing the distance function and afterwards we formulate a novel algorithm. This one has complexity $O(n \log_2 n)$ and it yields Euclidean distance. In addition to that, we introduce a modification to this algorithm, enabling it to work also in maze-like, wall- and corner-containing, environments. This modification relies on the line rasterization algorithm. We perform various experiments to study and compare distance function methods. Results illustrate the viability of newly-proposed method.

Further, a software for the comparing and inspecting cell detection results, SliceViewer, is specified, designed, implemented and tested.

In the end, quantitative experiments are discussed, validating the above-mentioned novelties.

Abstrakt

V tejto práci prezentujeme viacero modifikácií zvyčajne používaných algoritmov, a taktiež zopár nových prístupov, v kontexte detekcie buniek v spracovaní biologických dát.

Na začiatku popíšeme algoritmus FBLSCD a analyzujeme jeho vlastnosti. Potom zadefinujeme plne automatickú metódu na nájdenie zastavovacieho času tohto procesu. Následne sa na FBLSCD pozrieme cez optiku morfologického spracovania obrazu, uchopíme ho ako morfologickú eróziu v škále šedi, a pomocou znalostí morfologických princípov spracovania obrazu sformulujeme alternatívny algoritmus, LSOpen.

V nasledujúcej sekcii diskutujeme o inšpekcii výsledkov detekcie centier, čiže množine bunkových identifikátorov. Pozorujeme, že porovnávanie výsledkov binárne je nedostatočné, preto využijeme koncept funkcie vzdialenosti.

Motivovaní požiadavkou robustného porovnávania množín bunkových identifikátorov, analyzujeme dostupné metódy počítania funkcie vzdialenosti a následne formulujeme nový algoritmus. Tento je rádu $O(n \log_2 n)$ a je definovaný pomocou Euklidovskej vzdialenosti. Navyše uvedieme modifikáciu tohto algoritmu, umožňujúcu počítanie vzdialenosti aj v prostrediach obsahujúcich steny a rohy, podobných bludiskám. Taktiež uvedieme viaceré experimenty, ktoré validujú novo navrhnuté algoritmy.

Následne je zdokumentovaný softvér SliceViewer.

Na záver uvedieme experimenty validujúce vyššie spomínané modifikácie a postupy.

Contents

List of Figures

List of Tables

List of Abbreviations

(In order in which they appear in text)

PDE - Partial Differential Equation

FBLSCD - Flux-Based Level Set Center Detection

LMCE - Local Maxima Count Evolution

LMCED - Local Maxima Count Evolution Decline

LSErode - Level-Set Erosion

LSDilate - Level-Set Dilation

LSOpen - Level-Set Opening

DF - Distance Function

UDF - Unconstrained Distance Function

CDF - Constrained Distance Function

BF - Brute-Force

TRRT - Time-Relaxed Rouy Tourin

FMM - Fast Marching Method

FSM - Fast Sweeping Method

VDT - Vector Distance Transform

DP - Dijkstra-Pythagoras

BDP - Bresenham-Dijkstra-Pythagoras

FIFO - First In, First Out

RAM - Random Access Memory

HDD - Hard Disk Drive

RGB - Red-Green-Blue

2 Introduction

Two-photon confocal laser microscopy technology enables us to view a *zebrafish* embryo few hours after fertilization, *in vivo*, and observe its behavior in the upcoming hours at the cellular level. Data provided by the microscope are sequences of 3D frames in time, containing an intensity value for each voxel.

Zebrafish is an organism of choice for many current *in vivo* developmental biology research projects - it is so due to its translucency, ability to accept contrast dyes, resistance to potential damage caused by mechanical handling and the heat of microscope light [1]. Images are captured in two channels: in the first, we can see the cell membranes, in the second, intensity blobs representing cell nuclei. We are working with 3D+time datasets, where we refer to a given pixel by its coordinates, namely (x, y, z, θ). The confocal laser microscope produces images as 2D slices (with a position of a pixel in a slice given by x and y coordinates) of 512x512 pixels, and the depth of the imaging is controlled by coordinate z, which varies from dataset to dataset. Its maximum is $z_{max} = 104$ in the dataset studied. Number of time steps, θ_{max}, is also a varying parameter, in our dataset $\theta_{max} = 320$. The dataset we are using is 070418a.

There are usually two types of *image understanding* problems solved, using this type of biological data. At first, for a given dataset, the task is to quantitatively describe various biological properties of the organism[2][3][4][5]. Secondly, the goal is to perform the tracking of cells over time, i.e. to reconstruct the cell lineage tree[6][7][8].

In either of the two mentioned frameworks, an automatic identification of cells is part of the image processing pipeline. For this task, we use the state-of-the-art algorithm, called FBLSCD [9][10], and an alternative method, called LSOpen[11]. There exist also other methods for object detection - see for example [12], [13] or [14] for reference. In this work, we discuss possibilities of benchmarking these methods, and describe the design and implementation of a software enabling us to do so.

We will next list the content of following chapters and highlight novelties in

them in **bold**.

Chapter 3 covers the image processing problem called *object detection*. Basic image features are introduced, together with the outlines of their standard detection methods.

Subchapter 3.1 describes the state-of-the-art algorithm, Flux-based level set center detection (FBLSCD). This algorithm is presented as PDE-based scale space evolution, which forces intensity contours of an image to shrink in inward-normal direction. It is demonstrated, that it serves well in the role of blob detection method, and since cell detection is a specific case of blob detection problem, we will see that FBLSCD is capable of performing this task with given data.

In subchapter 3.2, we discuss a challenging problem of setting the parameters of FBLSCD. There are four parameters to be set: δ as advection term coefficient, μ as curvature term coefficient, R as minimal intensity for the local maximum to be considered as a cell identifier and T as the duration of scale space evolution. All of these terms are data-dependent and quality of their setting directly influences the quality of results the algorithm yields. We try to tackle the challenge of formulating a **fully-automatic way to set the value of parameter** T, considering that δ, μ and R are given. In [10], authors suggested to consider observing local maxima count evolution (LMCE) and define T as plateau point, but then they were dissatisfied with the results it yielded while working with real data. We built upon their experience by first defining T **as a minimum of** quantity we call **local maxima count evolution decline (LMCED)**, and then we suggested to **apply least-squares polynomial fit to this evolution and define** T **as its first local minimum**. This novelty yields results for T which have minimal variance in the frame-to-frame sense.

Subchapter 3.3 describes a combination of two image processing viewpoints, namely PDE-based and morphological approaches, for the purpose of object detection. At first, we reduce FBLSCD equation by removing the curvature term by setting $\mu = 0$ and normalize the advection term by setting $\delta = 1$. This operation can be seen as *morphological level set grayscale erosion* [9] [2] and we call it *LSErode*.

It can be argued that this operation can differentiate between large blobs of cell nuclei and high-intensity peaks of noise set in the low-intensity backgrounds. However, in case where there are low-intensity valleys of noise in the cell nuclei objects present, this method would enhance them, thus dramatically reducing detectability of a given nucleus. Therefore, we expand upon the concept of LSErode and by simply switching the sign at the advection term, **we formulate a *morphological level set grayscale dilation* and call it *LSDilate***. This method does the exact opposite to LSErode, namely it shrinks low-intensity noise artifacts in high-intensity blobs and expands high-intensity noise artifacts in low-intensity background. Morphological image processing school provides a standard concept of combining erosion and dilation by simply performing these operations in succession: erosion and then dilation is called *opening*, while dilation followed by erosion is called *closing*. **By performing a step of LSErode and then a step of LSDilate, we obtain *morphological level set grayscale opening* and call this operation *LSOpen***. We will show that this novel method is robust to both types of noise mentioned above.

In subchapter 3.4, methods of cell detection algorithms evaluation are discussed. At first, we argue that the very large amount of data provided by cell detection algorithm makes it impractical for human inspector to evaluate the quality of this result. However, by letting two distinct cell detection algorithms yield their results, out of which one may be the ground truth data, and **comparing these only in identifiers in which they differ**, we reduce this workload significantly. Furthermore, if we manage to not only tell the correspondence of centers in a one-to-one manner, but also to **measure the *distance* from a center yielded by the first algorithm to the set of centers yielded by the second algorithm and vice versa**, this workload can be reduced even further.

Chapter 4 discusses methods for computing the *distance function*. Last subchapter of previous chapter can be seen as a motivation for this direction of research. In the beginning of this chapter, we define the problem and specify requirements for the solution.

Subchapter 4.1 studies *unconstrained distance function* methods - those are methods, which compute distance d from the set of source points Ω_0 in the whole domain Ω, s.t. $\Omega_0 \subset \Omega$. Here, we analyze 5 commonly-used DF methods: Brute-force (BF), Time relaxed Rouy Tourin (TRRT), Fast marching method (FMM), Fast sweeping method (FSM) and Vector distance transform (VDT). In addition to describing properties of these methods, we also present pseudocodes for each of them. We try to classify these methods based upon 1) distance definition - eikonal or Euclidean and 2) pixel visit order strategy - multi-visit, wavefront or sweeping. **We formulate our own novel method, which fills up an empty gap in this classification, namely a Euclidean wavefront-type method. We call it *Dijkstra-Pythagoras*.** This method relies on first computing a distance estimation, upon first few visits of a given pixel, similarly to Dijkstra-like graph search algorithms, and then solidifies a correct Euclidean distance value on the last visit of a given pixel, using Pythagoras' theorem.

The next subchapter, 4.2, formulates a more complicated problem: the task is to compute distance from the set of source points Ω_0 in the domain Ω, s.t. $\Omega_0 \subset \Omega$, with addition of respecting set of wall points $\Omega_\infty \subset \Omega$. We call this problem type *constrained distance function* computation. It is in this environment, where the wavefront-type methods behave more reliably than sweeping-type methods. **We have to introduce some modifications for the Dijkstra-Pythagoras to work, namely to implement a visibility test from a given pixel to the source pixel. We use line rasterization algorithm to perform this task, and we call this novel method *Bresenham-Dijkstra-Pythagoras*.**

To wrap up the discussion about distance functions, we perform a wide variety of experiments and present them in the subchapter 4.3. We measure precision, speed and quality of results for unconstrained and constrained cases and also qualitative properties of these results to solve maze navigation problem. **Here, we can see that the novel BDP method has a few quantitative and qualitative advantages** over the state-of-the-art method, FMM.

Chapter 5 is a more technical one, where the SliceViewer, a software enabling

the user to semi-automatically evaluate cell detection results, is introduced and presented. Software enables user to view 4D data on a 2D computer screen by introducing two sliders, is capable of visualizing motion in time by making three distinct channels (red, green and blue) view three consequent frames (previous, current and next, resp.), and quality of cell detection results can be measured by classification of correspondencies and differences.

And finally, chapter 6 is a discussion about various experiments performed. In the first experiment of this chapter, we will see a comparison of behavior of FBLSCD and LSOpen cell detection methods. This experiment shows that while majority of results yielded by FBLSCD and LSOpen are in accordance, there is also a **small percentage of differences, namely LSOpen result sets are missing some correct cell nuclei which got detected by FBLSCD, but LSOpen results shows that FBLSCD introduced a few false positives to the result set**. In the second experiment, we also compare the results against a ground truth data and observe, that both methods perform reasonably similar and well. In the third experiment, we try to **empirically analyze correctness of LSOpen**'s numerical discretization and implementation.

3 Detection of objects in images

In order to solve the vaguely defined image processing problem called *image understanding*, image processing softwares/pipelines often choose to extract a system of significant entities called *features*, from the image. Using their hierarchy, topology and spatial relationships, they try to obtain understanding of image content.

Among the features usually considered, are edges, boundaries, corners, blobs and ridges.

Edges are defined as points where there is a boundary between two logical objects. Quality of edge detection results usually depends on strong gradient magnitude. More elaborate techniques are able to fill in the missing edges, by extrapolating from the present ones, using knowledge of object shape and topology [15]. An edge is a locally one-dimensional structure.

Boundary of an object is a locally one-dimensional structure for 2D image, similar to an edge. The difference is that it is a closed curve and is encompasses a simply connected region.

Blobs in image processing are understood as clusters of pixels having roughly the same properties - color, intensity or texture.

Many classes of algorithms aim to detect these features. Some analyze image using the mathematical operators, such as Laplacian of the image, gradient vector field, hessian matrices, eigenvalues and eigenvectors [13]. Also, the morphological image processing offers a handful of techniques: hit and miss transform, skeletonization, pruning [12]. Some special shapes can be found in the image using the Hough transform. The class of algorithms we are dealing with are PDE-based models.

3.1 Flux-based level set center detection

Flux-Based Level Set Center Detection (FBLSCD) is, from the image processing point of view, a blob-detection method. It was first introduced in [10]. It is based upon the difference in size of nuclei and noise artifacts - the smallest cellular structures are still larger than the largest noise structures. In FBLSCD, all contours of the level set of the image are forced to shrink in the inward normal direction (according to the advection term). Furthermore, the speed of this shrinking is enhanced by the curvature of the contour (according to the curvature diffusion term). Thus, in theory, the noise artifacts should disappear sooner than the cell blobs do. From the mathematical point of view, it is formulated as an advection-diffusion-type partial differential equation. Its equation reads as follows:

$$u_t + \delta \frac{\nabla u}{|\nabla u|} \nabla u - \mu |\nabla u| \nabla \cdot \left(\frac{\nabla u}{|\nabla u|} \right) = 0. \tag{3.1}$$

Initial condition is the input image. Zero Neumann boundary condition is introduced at the image boundary. First term is the time derivative, $\delta \geq 0$ in the second term is the coefficient of the advection, and $\mu \geq 0$ in the third term represents the coefficient of the curvature diffusion. Scale evolution parameter is t, $t \in [0, T]$, where T will be called *stop time*, and we will discuss it in the following chapters. At time T, local maxima of u larger than some threshold value R are considered. The set of these points represent the blob center identifiers, and is considered to be the output of the FBLSCD algorithm.

We solve the FBLSCD equation numerically. We discretize the equation in space, using the finite volume method. Image voxel serves as a natural choice for the control volume. We use upwind principle for advection term discretization, and diamond cell method to discretize the curvature diffusion term. For the time discretization in time, we use semi-implicit principle. The fully discretized scheme of eq. (3.1) is following:

$$
\begin{aligned}
m(V_{ijk})\frac{u_{ijk}^{n}-u_{ijk}^{n-1}}{\tau_C} &= \sum_{N_{ijk}^{in}}(u_{i+p,j+q,k+r}^{n-1}-u_{ijk}^{n-1})v_{ijk}^{pqr} \\
&+ \mu\bar{Q}_{ijk}^{n-1}\sum_{N_{ijk}} m(e_{ijk}^{pqr})\frac{u_{i+p,j+q,k+r}^{n-1}-u_{ijk}^{n-1}}{Q_{ijk}^{pqr;n-1}m(\sigma_{ijk}^{pqr})}\,, \qquad (3.2)
\end{aligned}
$$

where n is the present time step, $n-1$ is the past time step, u_{ijk} is the actual voxel, $u_{i+p,j+q,k+r}$ is a neighboring voxel (with the property $|p|+|q|+|r|=1$), the set N_{ijk}^{in} is the set of inflow voxels, $m(V_{ijk})$ is voxel size, τ_C is time step size, v_{ijk}^{pqr} is the upwind scheme approximation of the advection defined as $v_{ijk}^{pqr} = \delta\frac{\nabla u^{n-1}}{|\nabla u^{n-1}|}$, $\bar{Q}_{ijk}$ is approximated gradient modulus in the voxel center, Q_{ijk}^{pqr} is approximated gradient modulus on a given voxel face, $m(e_{ijk}^{pqr})$ is the distance between neighboring voxel centers, $m(\sigma_{ijk}^{pqr})$ is the measure of voxel face. A more detailed derivation and discussion of the eq. (3.2) can be found in [2] [9].

The FBLSCD behavior is illustrated in the figure 3.1.

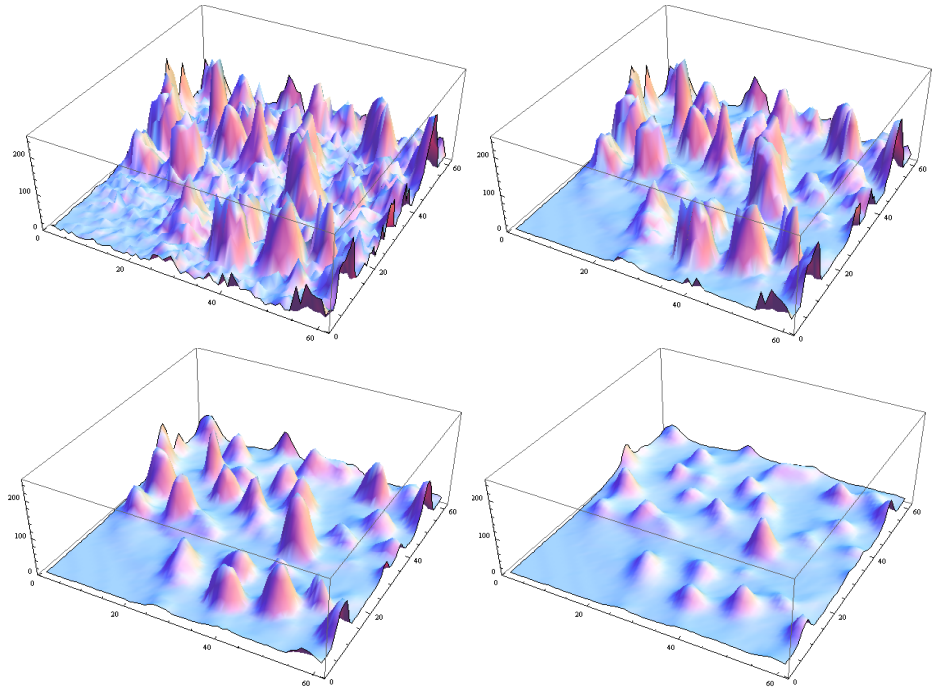


Figure 3.1: The center detection process, shown in 60x60 pixel part of slice $z = 60$ of frame $\theta = 100$. Top left, the intensity function of the original image. Top right, the intensity function of the filtered image. The filtering was performed by the GMCF algorithm [16]. Bottom left, the intensity after 10 steps of center detection process. Bottom right, the intensity after 30 steps.

3.2 Finding the optimal stop time

In [10], the authors consider the following method of finding the optimal FBLSCD stop time: observe the local maxima count evolution (LMCE) over the scale parameter t, and stop it the first time it happens to be non-descending. This method yielded reasonable results for the input artificial images, but didn't work that well with real data. We observed [11] that this event is not guaranteed to occur at the representative time, and in some cases it does not happen at all - see fig. 3.2 for LMCE of frame $\theta = 100$.

A more refined way is to first compute the LMCE decline sequence (LMCED) from LMCE, and then consider the first time it happens to be below a certain threshold. We construct the LMCED by taking first difference of LMCE and multiply its values by -1, so that the largest values mean the largest decline of local maxima count in subsequent frames. Note that in the language of the LMCED,

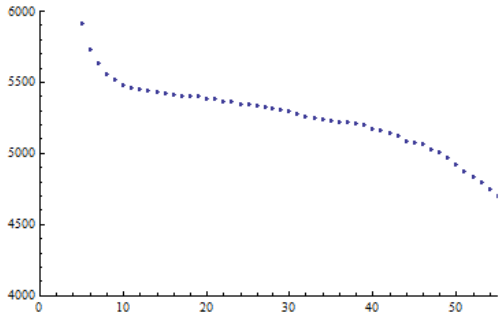


Figure 3.2: Local maxima count evolution (LMCE) at time $\theta = 100$. While after a while the decline stops, a proper plateau is never truly formed and it is difficult to define the stop time.

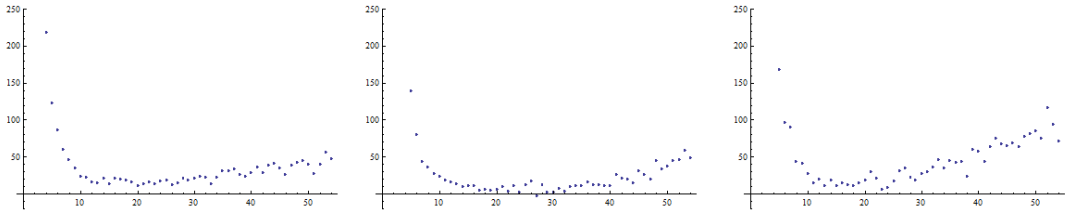


Figure 3.3: Local maxima count evolution decline (LMCED) at time $\theta = 0$, 100 and 200. If we were to define stop time as global minimum of LMCED, stop times for given frames would be 19, 26 and 22, respectively.

the original approach was to see it become less than or equal to zero, and call time of this occurence the stop time. Now this threshold can be larger than zero. This approach was proposed in [2].

We want to propose a novel approach, namely to consider minimum of LMCED. It can still be greater than zero. This strategy is a certain weakening of the requirement of LMCE's non-descendence, and, formulated as such, it is an automatic method for finding the optimal stop time, as no input from the user is required. The results can be seen in fig. 3.3.

Whatever strategy we choose, a natural requirement for the results it yields is to not to vary too much from frame to frame along the dataset. This requirement arises naturally from the fact that the sequence of 3D frames, if viewed as a video, flows rather smoothly, from frame to frame, without flickering or exhibiting any

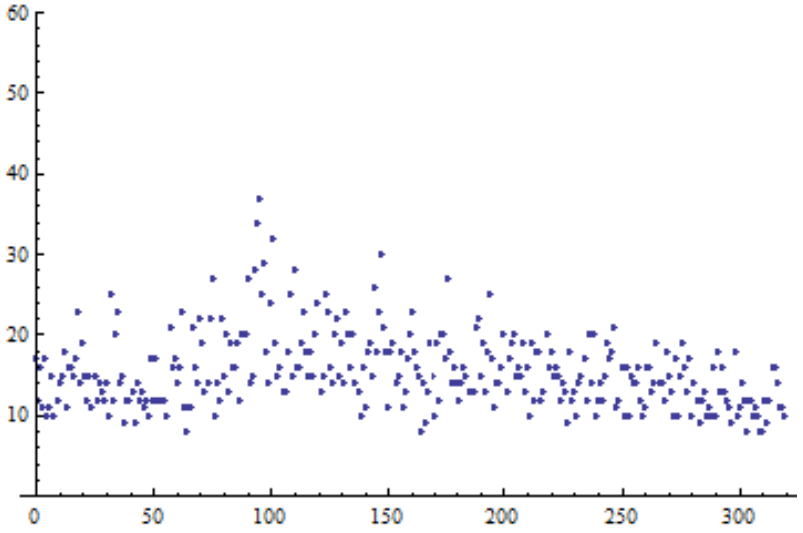


Figure 3.4: Global minima of LMCED for the whole dataset $\theta \in [0...320]$. Standard deviation is 4.62377.

abrupt intensity jumps. So the transitions of all the parameters between the frames should be smooth as well.

We found out, that taking the newly proposed minimum of LMCED did not have this property - it seems to vary from frame to frame with a rather large variance. See fig. 3.4 for reference. In order to tackle this, we found a least-squares fit of the LMCED and took the first local minimum of this fit. We computed fitting of real LMCED data by polynomials of degree $n \in [2...7]$. First local minima of the fittings, which are computed from each frame's LMCED and are independent of the results in neighboring frames, happen to have much lower variance from frame to frame than taking raw LMCED minima. Observe this information for the chosen time in the fig. 3.5, and see the analysis for the whole dataset in the fig. 3.6.

The open question at this point is, what is the right degree of a fitting polynomial, n. One of ways to choose n is to pick the one which minimizes variance of optimal time steps in a sequence of frames of the dataset. For FBLSCD, this happens to be $n = 5$. Another way is to consider visual inspection of the LMCED in the sequence of frames, for each n, and let the user pick the correct n.

Thus, we propose a novel way to define the fully-automatic optimal stop time: by considering the first local minimum of the variance-minimizing polynomial fit

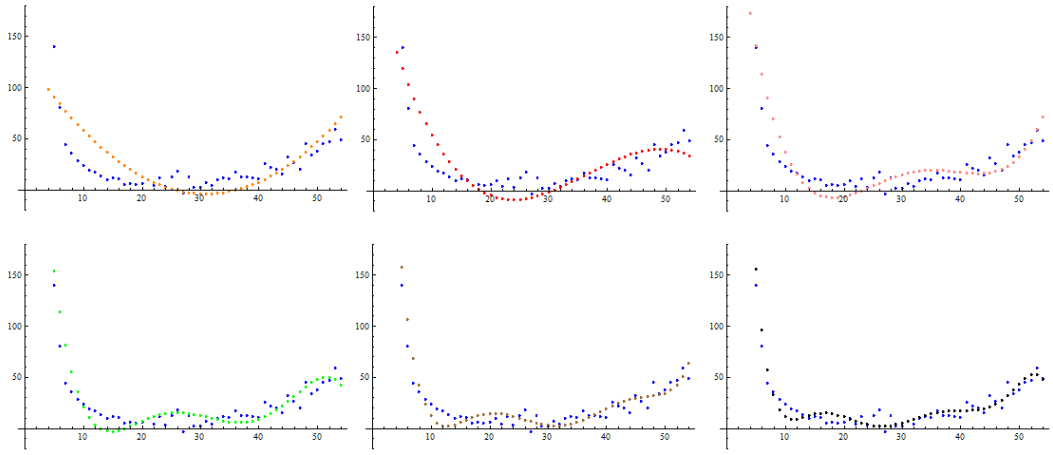


Figure 3.5: Polynomial fittings of LMCED at $\theta = 100$. Upper row, degrees of fitting polynomial 2, 3 and 4, lower row, degrees of fitting polynomial 5,6 and 7.

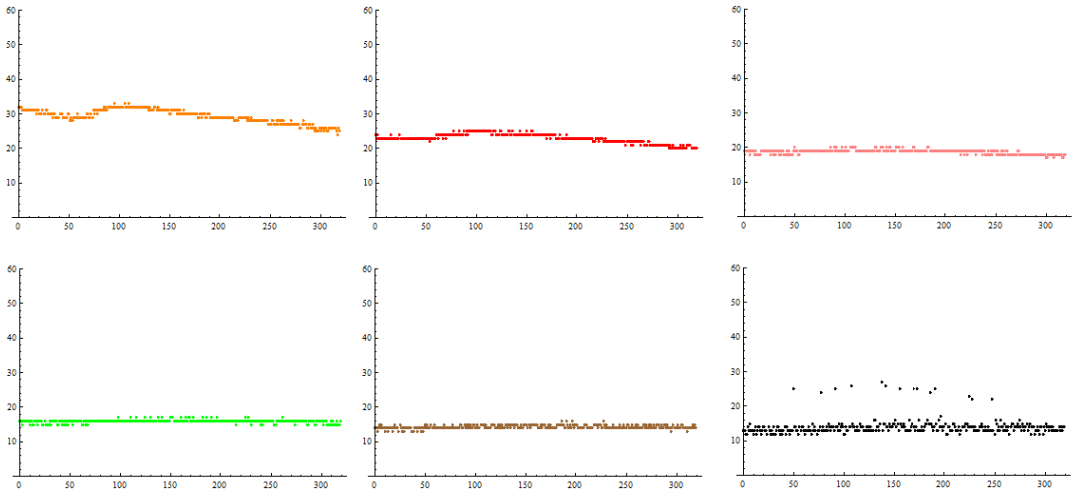


Figure 3.6: First local minima of polynomial fittings of LMCED, for the whole dataset, $\theta \in [0...320]$. Upper row, degrees of fitting polynomial 2, 3 and 4, lower row, degrees of fitting polynomial 5, 6 and 7. The corresponding standard deviations are 1.9597, 1.3698, 0.6128, 0.4555, 0.5795 and 2.4477. Thus, 5th degree polynomial fit yields minimal standard deviation.

of the LMCED data. It is in accordance with the requirement of smoothness of parameter variation along the inspected sequence and also with visual inspection.

In order to validate this approach, we performed a similar analysis of the optimal stop time on one more dataset, 070420a. The results yielded by this analysis can be seen in the fig. 3.7.

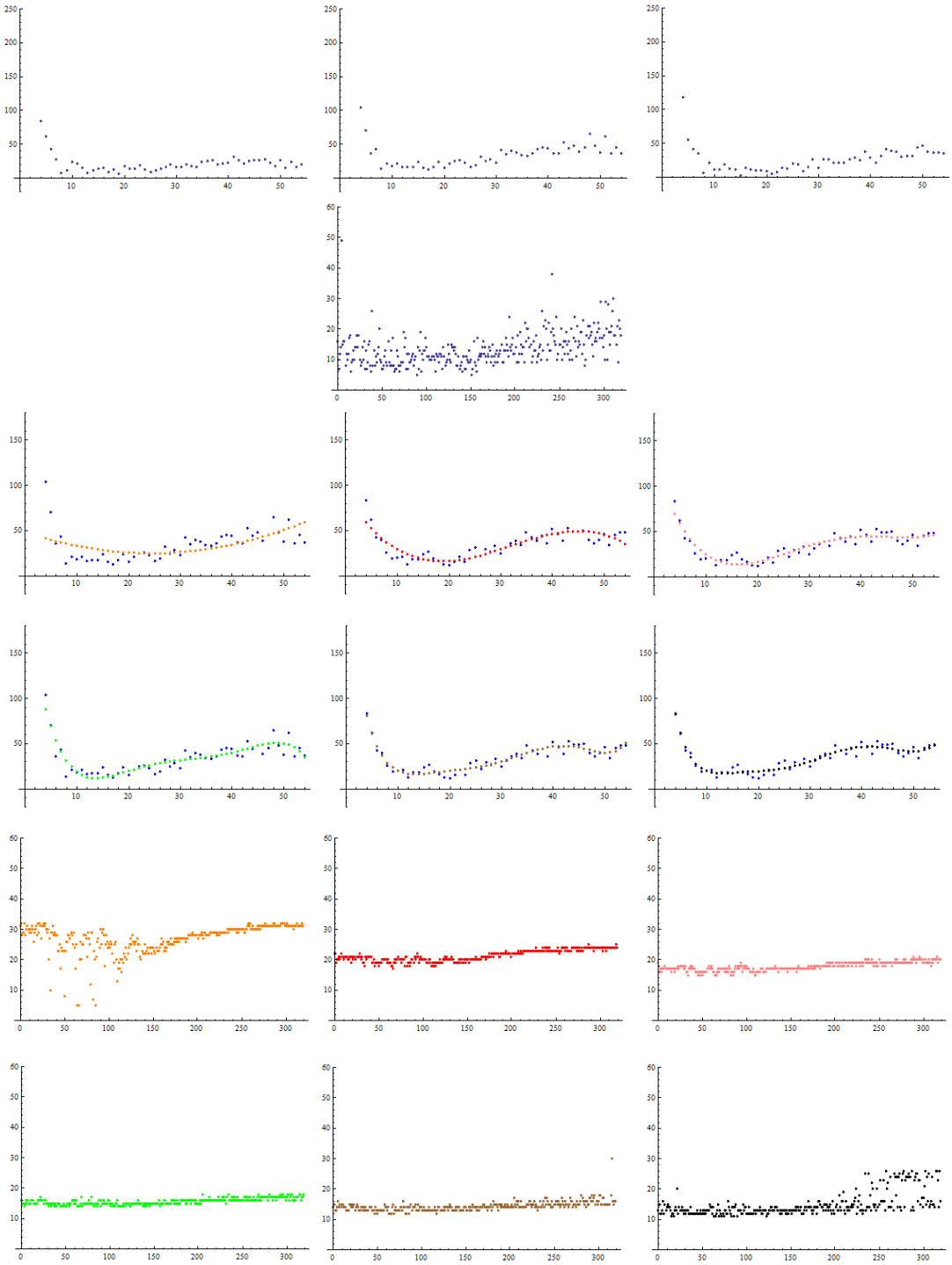


Figure 3.7: The complete optimal stop time analysis for a different dataset, 070420a. First row - LMCED at time $\theta = 0$, 100 and 200. Second row - Global minima of LMCED for the dataset. Standard deviation is 5.27447. Third and fourth row - Polynomial fittings of LMCED at $\theta = 100$, for $n \in (2..7)$. Fifth and sixth row - First local minima of polynomial fittings of LMCED, for the whole dataset. The corresponding standard deviations are 4.7886, 1.8622, 1.3320, 1.0179, 1.5174, 3.8109. It is again the 5th degree polynomial fit, which minimizes the standard deviation.

3.3 Morphological cell detection

FBLSCD can be interpreted as a morphological grayscale erosion with curvature regularization [14]. Let us observe its behavior with this property in mind, and try to construct and consider the usage of similar, more general morphological PDE-based operators.

We have tested FBLSCD with phantom data, where we introduced salt and pepper noise. We set diffusion coefficient μ to zero (in [9], $\delta = 800\mu$ in eq. (3.1)), and it was observed that while it is able to handle pepper-type noise, salt noise remains to be present. Reason for this is that advection in FBLSCD is in the direction of the inward normal, only if we consider level sets of high-intensity blobs surrounded by low-intensity environment. As soon as we have low-intensity blobs surrounded by high-intensity environment, behavior of FBLSCD is different - now the advection happens in the outward normal. This is when salt-type noise is located in the cell interior - this noise artifact not only remains, but also grows in size [11].

In general, it is a difficult task to analyze the noise contained in the real data and find a well-suited mathematical/stochastic model to describe it. It is an open question, if a) this type of artifacts happens to be present in the real data input, and even if it is, if b) it remains present to certain extent also after the image filtering. Nevertheless, we felt motivated to design an algorithm, which can tackle both types of noisy structures, since a) we want our image processing workflow to be able to process a variety of input data with different properties and b) we often want to skip the noise filtering step in the workflows, in order to save computer resources or preserve data fidelity.

We have seen that FBLSCD (namely the implementation of its discretization, see eq. (3.2)) with $\mu = 0$ and $\delta = 1$ is a level-set based pepper-shrinking and salt-expanding evolution. This behavior is in fact similar to the grayscale level set erosion, a well-known morphological image processing operation. Thus, we called this special case of FBLSCD *LSErode*. PDE of LSErode looks as follows:

$$u_t + \frac{\nabla u}{|\nabla u|} \cdot \nabla u = 0. \tag{3.3}$$

Initial condition is the input image and boundary condition is zero Neumann.

We will now illustrate the process of discretization of LSErode's equation to obtain a formulation solvable numerically by computer. The discretization will be performed regardless of problem dimensionality. We will use finite volume method, with pixel/voxel/doxel serving as a control volume. We will call this volume uniformly to be "pixel", even if we may mean voxel (in 3D case) or doxel (in 4D case). We will further assume that all sizes are 1: pixel's volume is 1, distance between two pixel centers is 1, each face size is 1 and each edge size is 1 (imagine a grid of unit cubes here, where all these properties hold true, in 3D case). We will use two identities of vector analysis:

$$\int_{\Omega} \bar{f} \cdot \nabla g dx = \int_{\Omega} \nabla \cdot \left(\bar{f} g\right) dx - \int_{\Omega} g \nabla \cdot \bar{f} dx \tag{3.4}$$

and

$$\int_{\Omega} \nabla \cdot \bar{f} dx = \int_{\Gamma} \bar{f} \cdot \bar{\nu} d\gamma \qquad , \tag{3.5}$$

where g is a scalar function, $\bar{f}$ is a vector function, Ω is integration domain, Γ is domain boundary and $\bar{\nu}$ is its a unit outward normal vector.

At first, we will discretize the equation (3.3) in time, using explicit method:

$$\frac{u^n - u^{n-1}}{\tau} + \frac{\nabla u^{n-1}}{|\nabla u^{n-1}|} \cdot \nabla u^{n-1} = 0, \tag{3.6}$$

where n is the new time step, $n-1$ is the previous time step and τ is time step size. Then, we will integrate over volume of pixel p:

$$\int_{V_p} \frac{u^n - u^{n-1}}{\tau} dx + \int_{V_p} \frac{\nabla u^{n-1}}{|\nabla u^{n-1}|} \cdot \nabla u^{n-1} dx = 0, \tag{3.7}$$

where p is the actual pixel and V_p is its volume. Since the volume of the pixel is 1, the first term of (3.7) can be approximated as

$$\int_{V_p} \frac{u^n - u^{n-1}}{\tau} dx \approx \frac{u_p^n - u_p^{n-1}}{\tau}, \tag{3.8}$$

where u_p is the value of u in pixel p. To approximate the second term, first observe that it has the form similar to (3.4), where $\bar{f} = \frac{\nabla u^{n-1}}{|\nabla u^{n-1}|}$ and $g = u^{n-1}$. If we apply

this identity, we obtain

$$\int_{V_p} \frac{\nabla u^{n-1}}{|\nabla u^{n-1}|} \cdot \nabla u^{n-1} dx = \int_{V_p} \nabla \cdot \left(\frac{\nabla u^{n-1}}{|\nabla u^{n-1}|} u^{n-1} \right) dx - \int_{V_p} u^{n-1} \nabla \cdot \left(\frac{\nabla u^{n-1}}{|\nabla u^{n-1}|} \right) dx. \tag{3.9}$$

Now we see that both right-hand-side terms of (3.9) have the form similar to (3.5). If we apply this identity and assume that u^{n-1} is constant in V_p, we end up with

$$\int_{V_p} \frac{\nabla u^{n-1}}{|\nabla u^{n-1}|} \cdot \nabla u^{n-1} dx \approx$$

$$\approx \sum_{q \in N_p} \int_{e_{pq}} u^{n-1} \frac{\nabla u^{n-1}}{|\nabla u^{n-1}|} \cdot \nu_{pq} d\gamma - u_p^{n-1} \sum_{q \in N_p} \int_{e_{pq}} \frac{\nabla u^{n-1}}{|\nabla u^{n-1}|} \nu_{pq} d\gamma, \tag{3.10}$$

where N_p is the neighborhood of those pixels of p, which share a common face with it - this is a 4-neighborhood in 2D case, 6-neighborhood in 3D and 8-neighborhood in 4D, e_{pq} is the face area between neighboring pixels p and q and ν_{pq} is the normal vector of this face, pointing outward from pixel p. Further, let us define

$$v_{pq} = \frac{u_q^{n-1} - u_p^{n-1}}{|\nabla u^{n-1}|_{pq}}, \tag{3.11}$$

where $|\nabla u^{n-1}|_{pq}$ is a numerical approximation of gradient magnitude on the face between pixels p and q - here, we use the so called *diamond cell* estimation - see [2] for implementation details. Using this definition, we can partition N_p into N_p^{out} and N_p^{in}, s.t. $N_p = N_p^{out} \cup N_p^{in}$:

$$N_p^{out} = \{q \in N_p, v_{pq} > 0\} \tag{3.12}$$

and

$$N_p^{in} = \{q \in N_p, v_{pq} \leq 0\}. \tag{3.13}$$

We will approximate terms in (3.10) using the upwind principle:

$$\int_{e_{pq}} u^{n-1} \frac{\nabla u^{n-1}}{|\nabla u^{n-1}|} \cdot \nu_{pq} d\gamma \approx u_p^{n-1} v_{pq} \qquad \text{if} \qquad q \in N_p^{out} \tag{3.14}$$

and

$$\int_{e_{pq}} u^{n-1} \frac{\nabla u^{n-1}}{|\nabla u^{n-1}|} \cdot \nu_{pq} d\gamma \approx u_q^{n-1} v_{pq} \qquad \text{if} \qquad q \in N_p^{in}. \tag{3.15}$$

Thus, we can approximate the first right-hand-side term of (3.10) using the upwind principle and the second right-hand-side term by just applying the fact that $N_p = N_p^{out} \cup N_p^{in}$:

$$\int_{V_p} \frac{\nabla u^{n-1}}{|\nabla u^{n-1}|} \cdot \nabla u^{n-1} dx \approx \sum_{q \in N_p^{out}} u_p^{n-1} v_{pq} + \sum_{q \in N_p^{in}} u_q^{n-1} v_{pq} - \sum_{q \in N_p^{out}} u_p^{n-1} v_{pq} - \sum_{q \in N_p^{in}} u_p^{n-1} v_{pq}, \tag{3.16}$$

which can be further simplified to

$$\int_{V_p} \frac{\nabla u^{n-1}}{|\nabla u^{n-1}|} \cdot \nabla u^{n-1} dx \approx \sum_{q \in N_p^{in}} (u_q^{n-1} - u_p^{n-1}) v_{pq}. \tag{3.17}$$

Final fully discrete form of (3.3) then reads as follows:

$$\frac{u_p^n - u_p^{n-1}}{\tau} + \sum_{q \in N_p^{in}} (u_q^{n-1} - u_p^{n-1}) v_{pq} = 0. \tag{3.18}$$

This linear system is a directly solvable explicit scheme.

We know that there exists a morphological operation called *dilation*. Erosion and dilation the basic, atomic operations in morphological image processing theory. We observed that we can formulate the level-set version of dilation by simply switching the advection sign in the LSErode.

Its PDE reads as

$$u_t - \frac{\nabla u}{|\nabla u|} \cdot \nabla u = 0 \tag{3.19}$$

and its fully discrete form, which can be obtained following the similar derivation principle as in (3.3) - (3.18), but now with v_{pq} containing minus sign in its definition:

$$v_{pq} = -\frac{u_q^{n-1} - u_p^{n-1}}{|\nabla u^{n-1}|_{pq}}. \tag{3.20}$$

Compare this with definition of v_{pq} in (3.11) to see the difference. The final fully discretized form of (3.19) is then the same equation (3.18).

Having tested this operation, we have seen, it works as expected - it is a salt-shrinking and pepper-expanding operation. We called this equation *LSDilate* [11].

Morphological operator theory gives us a way of combining these equations together. Similar to morphological opening, which is an erosion step, followed by

dilation step, we performed a step of LSErode, followed by a step of LSDilate, and called this non-atomic operation *LSOpen*. We have computed several LSOpen steps (an alternating sequence of LSErode and LSDilate steps) and we observed, that with phantom data, this operation makes both salt- and pepper-type artifacts disappear, and the local maxima larger than a certain threshold, similar to FBLSCD, are viable cell center identifiers [11].

Behavior of LSErode, LSDilate and LSOpen is illustrated in the fig. 3.8.

Notice that LSOpen is not formulated as a standard advection-diffusion equation, so the way to analyze it theoretically is an open question from mathematical point of view. We will, however, try to reason about its properties experimentally: see the third experiment in chapter 6.

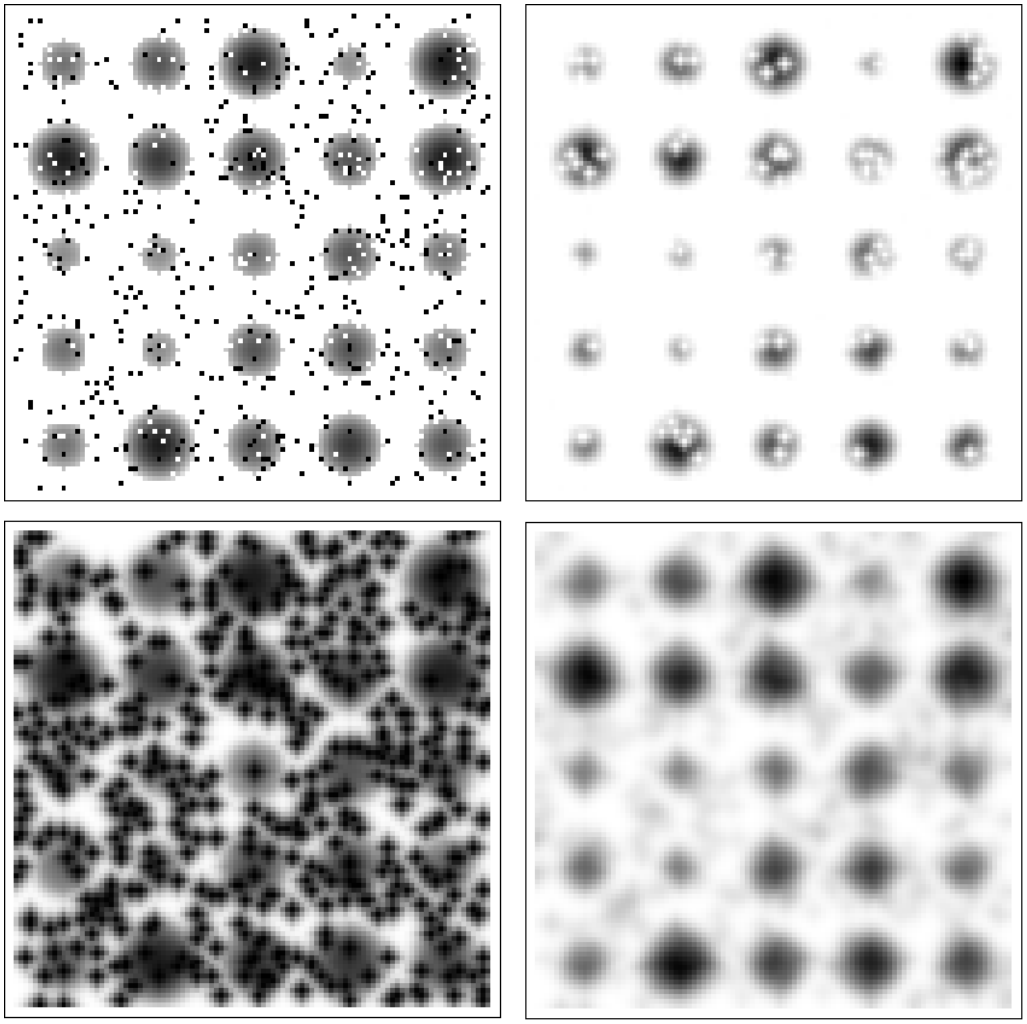


Figure 3.8: Illustration of cell detection algorithms behavior with noisy phantom images. Upper left - input phantom image, with cell intensities being quadratic functions of distance to its centers of varying size, with salt and pepper noise added. Upper right - 10 steps of LSErode. Notice black spots disappear and white enhance. Lower left - 10 steps of LSDilate. Notice white spots disappear here, and black spots enhance. Lower right - 10 steps of LSOpen. Both noise types disappear, and local maxima larger than threshold R tend to denote true cell center positions with increasing scale parameter t.

3.4 How to compare various cell detection results?

In order to reason about quality of available cell detection algorithms, one has to have a way of measuring the quality of results they yield.

At first, it is reasonable to inspect the results at first visually, e.g. by visualizing the sets of points representing cell identifiers, superimposed over the original 3D data volume.

If the results are visually pleasing, the next step would be to estimate the number of false positives, false negatives, and the number of multiple identifiers yielded for one cell. However, FBLSCD yields approximately 3500 cell identifiers at the beginning of the dataset and about 6500 in the end. Dataset is 320 frames long, so a rough estimation of total cell images present in the video is about 1.6 million. It is therefore difficult to check the correctness of each detected identifier manually.

However, it is worth noting that comparing two different cell identifier sets is not that much work, provided they don't contradict each other in too many points. Then, checking only their differences suffices.

Thus, the first method we propose, is not to check each identifier yielded by a given method by itself, but instead, compare always two different results of two different cell detection approaches and only check visually those points in which they differ. As we will see in the chapter about the software, we implemented a software solution enabling us to perform this task efficiently. As we will see in the experiment discussion chapter, this method of mutual result comparison reduces the amount of work to be done by the human inspector significantly.

We have further observed, that the vast majority of identifier differences happen to be only small shifts, within the same cell nucleus. Therefore, we felt motivated to not only compare the identifiers in a binary fashion (i.e. labelling each identifier as 'corresponding' and 'not corresponding' to an identifier in the alternative method), but also to introduce some kind of metric, that would measure the distance to the closest cell identifier yielded by the alternative method, for each 'not corresponding' identifier.

Therefore, we propose a second reduction of the amount of work to be done

by the visual inspector: Computing the distance function for each cell identifier in one method, finding the corresponding closest identifier from this method to each cell identifier in the second method via steepest descent, and vice versa, and only checking differences over certain threshold of distance.

We will see that this approach worked well with our data and methods, in the experiment discussion chapter.

An open question for now is how to efficiently compute cell identifier correspondencies and their distances in order to find the closest pairs. We will use distance function to help us. This will motivate us to discuss distance function (DF) computation in the next chapter.

4 Methods for computing distance function on a pixel-based grid

At first, let us define the vocabulary, since these definitions these terms vary across scientific texts. The mathematical community calls the result of distance computation to be the *distance function* (DF) [4][21]. The signal processing community uses the name *distance transform* or *distance field* [17][18][25]. To our understanding, there are no practical differences between the meanings of these two terms. Since this work is classified as an applied mathematics text, we will stick to the former of the names across the text. A *DF method* will then be an algorithm, which yields DF as a result.

We will analyze commonly-used DF methods. Let us classify these based on

A) distance definition – distance function found as a solution of the so called *eikonal equation*, or *Euclidean distance* and

B) pixel visit order strategy – multi-visit, wavefront, or sweeping.

Then a new method will be introduced and we will see it filling an empty gap in the logic of existing methods. Further, we will use resulting DFs to find the shortest path solution of maze navigation, and demonstrate the usefulness of the newly proposed method in this environment.

This work limits its scope to analyzing these commonly used DF methods: Brute-force, Time-Relaxed Rouy-Tourin Scheme, Fast Marching Method, Fast Sweeping Method and Vector Distance Transform. Multiple surveys comparing a broader range of approaches have been published [17] [18]. The DF application studied in this article is navigation in a maze-like environment. There are many other applications of DFs, let us mention biological and medical data analysis [4] [19] [6] and level-set computational physical simulations [15], just to name a few.

4.1 Unconstrained distance function

Let us introduce *computational domain* Ω, $\Omega \subset R^n$, and $\Omega_0, \Omega_0 \subseteq \Omega$, called the *source set*. For means of digital image processing, we simplify our task to dimen-

sionality $n = 2$, domain Ω being a square grid of *pixels*, and subset Ω_0 being a set of *source pixels*. Define function $d, d : \Omega \to R$ called *distance*. Distance should be $d(a) = 0, \forall a \in \Omega_0$ (this can be regarded as special Dirichlet-type condition) and the task of a DF method is to find $d(b), \forall b \in \Omega \backslash \Omega_0$.

4.1.1 Distance definition

Eikonal equation is given by

$$|\nabla d| = 1 \text{ in } \Omega \backslash \Omega_0, \qquad d = 0 \text{ in } \Omega_0$$

It says, that the gradient magnitude of the distance should be $= 1$ everywhere, except for source pixels, where the value is fixed at 0. If we consider pixel a and a set of all its 4-neighbors being $N_a = \{n, e, s, w\}$, we can define $\forall b \in N_a$ quantities [20]

$$M_a^b = \left(min(d(b) - d(a), 0)\right)^2,$$

and use them to define approximation of gradient magnitude $\bar{g}(a)$ [20]

$$\bar{g}(a) = \sqrt{(max(M_a^n, M_a^s) + max(M_a^e, M_a^w))}. \tag{4.1}$$

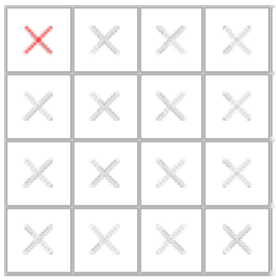
Eikonal-based methods solve a problem $\bar{g}(a) = 1$, $\forall a \in \Omega \backslash \Omega_0$.

In Euclidean space, we define *Euclidean distance* by Pythagoras' theorem. $\forall a = (a_x, a_y), b = (b_x, b_y) \in \Omega$:

$$d(a, b) = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2}.$$

In continuous formulation of the problem, results yielded by either eikonal equation or Euclidean distance definition are the same. However, on a discrete grid, they are different: a result obtained by eikonal equation condition fails to yield correct Euclidean distances, and vice versa: results obtained with Euclidean distance condition fails to have $\bar{g} = 1$, for $\bar{g}$ defined as in eq. (4.1). This is illustrated by a simple example in tab. 4.1.

Reason for this disparity is non-exactness of discrete gradient approximation. Eikonal equation only needs information about several closest neighbors of a given



(1)

0	1	2	3
1	1.707	2.545	3.442
2	2.545	3.252	4.048
3	3.442	4.048	4.755

(3)

0	1	2	3
1	1.414	2.236	3.162
2	2.236	2.828	3.606
3	3.162	3.606	4.243

(2)

0	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

(4)

0	1	1	1
1	0.586	0.855	0.94
1	0.855	0.838	0.895
1	0.94	0.895	0.901

Table 4.1: Disparity between eikonal and Euclidean distances in discrete case. Left illustration: compute DF in a small 4x4 pixel image, where top left pixel is the only source pixel. (1)(2): results for eikonal equation approximated as stated above, inexact d values (1) and correct $\bar{g}$ values (2). (3)(4): results for Euclidean distance definition, correct d values (3) and inexact $\bar{g}$ values (4), if considering $\bar{g}$ defined as in eq. (4.1)

pixel in order to tell this pixel's distance value. Euclidean distance, on the other hand, requires and uses information about closest source pixel for a given pixel, which can in fact be anywhere in the image. Thus, we can say that eikonal solvers in general behave locally, while Euclidean are global. We will further see that the classification of a method as local or global determines a lot of its properties and behavior.

4.1.2 Pixel visit order strategy

In this work, we adopt the term *multi-visit* approach: it is a solution, which doesn't try to optimize the number of times each pixel is visited. It aims to reduce implementation time, but usually at the price of computational redundancy. With this definition of multi-visit approach, a multi-visit method can be formulated for both distance definitions, Euclidean and eikonal equation-based.

Wavefront strategies ensure that each pixel is assigned final value already in

Algorithm 1 Brute-Force DF algorithm pseudocode

Require: allocate $\forall p \in \Omega : d(p) = +\infty$

```
for each pixel p ∈ Ω do
    for each pixel s ∈ Ω_0 do
        D = sqrt((p_x − s_x)^2 + (p_y − s_y)^2)
        if D < d(p) then
            d(p) = D
        end if
    end for
end for
```

the first pass. To make this happen, methods have to pay attention to visiting pixels in the correct order. Algorithms use min-priority heap data structure for managing pixel visit order.

Sweeping strategies allow multiple passes of each pixel, but the number of these sweeps is bound to the dimensionality of the problem - it is 4 sweeps for a 2D problem in total. By altering sweep directions DF is computed for the whole domain without the need for order-managing priority heap.

4.1.3 Brute-force (BF)

We can trivially formulate a multi-visit method yielding exact results in Euclidean sense: Set $d(a) = \infty, \forall a \in \Omega$. For $\forall a \in \Omega$, search through $\forall b \in \Omega_0$, and define D as Euclidean distance between a and b. If $D < d(a)$, set $d(a) := D$. Since this method visits each source pixel for each pixel, its complexity is $O(ns)$, where n is the number of pixels in Ω and s is the number of pixels in Ω_0. The worst-case scenario of brute-force method occurs when half of the pixels are source pixels: in that case, $s = n/2$, and complexity is thus $O(n^2/2) = O(n^2)$. Algorithm is also presented in the form of a pseudocode - see Alg. 1.

Algorithm 2 Time-Relaxed Rouy-Tourin DF scheme

Require: allocate $\forall p \in \Omega_0 : d(p) = 0, f(p) = 1$

Require: allocate $\forall p \in \Omega \backslash \Omega_0 : d(p) = 0, f(p) = 0$, define $D : \Omega \rightarrow R$

$\tau = 0.5$

$\epsilon = 0.0001$

while number of p s.t. $f(p) = 1 <$ number of $p \in \Omega$ **do**

  for each pixel $p \in \Omega$, s.t. $f(p) == 0$ **do**

    $(i, j) = p$ coordinates

    $m_{i+1,j} = [\min(d_{i+1,j} - d_{i,j}, 0)]^2$

    $m_{i-1,j} = [\min(d_{i-1,j} - d_{i,j}, 0)]^2$

    $m_{i,j+1} = [\min(d_{i,j+1} - d_{i,j}, 0)]^2$

    $m_{i,j-1} = [\min(d_{i,j-1} - d_{i,j}, 0)]^2$

    $D(p) = d(p) + \tau - \tau\sqrt{\max(m_{i+1,j}, m_{i-1,j}) + \max(m_{i,j+1}, m_{i,j-1})}$

    if $|D(p) - d(p)| < \epsilon$ **then**

      $f(p) = 1$

    end if

  end for

  for each pixel $p \in \Omega$ **do**

    $d(p) = D(p)$

  end for

end while

4.1.4 Time-Relaxed Rouy-Tourin (TRRT) scheme [4]

This multi-visit method yields discretized solution of eikonal equation, by formulating it as an evolution in time [20], with consecutively raised values in a sedimentation-like manner. Each pass, all pixels except for the source pixels and those with $\bar{g} = 1$ are incremented by a time-relaxed gradient estimation defined in eq. (4.1). This method is fairly easy to implement. Number of times each pixel is visited is proportional to its distance, which is of course not known beforehand. Worst-case complexity of this method, that is when image is extremely thin, is therefore $O(n^2)$. Algorithm is described in Alg. 2.

4.1.5 Fast Marching Method (FMM) [21]

This is a wavefront-type method yielding eikonal equation-based results. All pixels are fixed in only one pass. The visit order is defined by causality principle - pixels which are closer to Ω_0 will be visited sooner than those further away. The first ones visited are all $x \in \Omega_0$. This algorithm assigns labels to pixels, similar to graph algorithms like breadth-first-search or Dijkstra's. A pixel can have 3 types of labels, meaning 'unvisited', 'to be visited' and 'visited'. Labels can also be interpreted in terms of field fire: 'grass', 'fire front' and 'burnt ground' – let us stick with this intuitive naming convention. All the pixels labeled as 'fire front' are contained in a data structure called *min-priority-heap* [23]. As pixels change label from 'grass' to 'fire front', they are inserted to the bottom of min-heap, as their states are changed from 'grass' to 'fire front', while min-value pixels are popped from its top as their states are changed from 'fire front' to 'burnt ground'. The behavior of min-heap is similar to FIFO queue, but unlike in FIFO, ordering of elements in min-heap can be changed during the waiting process. Worst-case complexity of this method is $O(n \log_2 n)$ - each pixel is visited once, therefore n, but in order to do so, it has to make it all the way from bottom to top of min-priority-heap, which is implemented as a *binary tree*. Height of the binary tree containing n elements is $\log_2 n$.

FMM is considerably harder to implement than TRRT scheme, since one has to implement a min-priority-heap data structure and its operations. To see the pseudocode of this procedure, refer to Alg. 3.

4.1.6 Fast Sweeping Method (FSM) [24]

This is a sweeping method yielding eikonal equation-based results. This method operates with concept of inializing $d(a) = \infty, \forall a \in \Omega \backslash \Omega_0$, and then reducing values at given pixels until they stop at correct values. We address this updating of values in non-increasing manner by the term *relaxation*.

Initialize $d(a) = \infty, \forall a \in \Omega \backslash \Omega_0$ and $d(b) = 0, \forall b \in \Omega_0$. Start from top left corner and make a sweep through computational domain in right-down direction (implemented as double for loop with both iterators increasing from 0) and relax all

Algorithm 3 Fast Marching Method (FMM) DF algorithm pseudocode

Require: values for v are defined as: 'grass'=0, 'fire front'=1, 'burnt ground'=2

Require: allocate $\forall p \in \Omega \backslash \Omega_0 : d(p) = +\infty, v(p) =$ 'grass',$s(p) =?$, allocate *heap*

Require: allocate $\forall p \in \Omega_0 : d(p) = 0, v(p) =$ 'fire front',$s(p) = p$, *heap*.pushBack(p)

while *heap* is not empty **do**
  p = *heap*.pop ▷ pops a 'fire front' pixel, whose estimate value is minimal of all
  for each pixel $r \in$ Neighborhood4(p), s.t. $v(r) =$'grass' or 'fire front' **do**
    $(i, j) = r$ coordinates
    $x = \min(d_{i,j+1}, d_{i,j-1})$ ▷ use $+\infty$ whenever out of bounds
    $y = \min(d_{i+1,j}, d_{i-1,j})$
    $a = 2$
    $(x == +\infty \ ? \ (a--; x = 0;) : \text{nothing})$
    $(y == +\infty \ ? \ (a--; y = 0;) : \text{nothing})$
    $b = -2 * (x + y)$
    $c = x^2 + y^2 - 1$
    $D = b^2 - 4ac$
    $T = (-b + \sqrt{D})/(2a)$
    if $T < d(r)$ **then**
      $d(r) = T$
      if $v(r) =$ 'grass' **then**
        heap.insert(r)
        $v(r) =$ 'fire front'
      else
        heap.decreaseKey(r, T)
      end if
    end if
  end for
  $v(p) =$'burnt ground'
end while

pixels using upwind neighbors, if they carry relaxation information. Make a similar sweep in right-up direction starting from bottom left corner, in left-up direction starting from bottom right corner and in left-down direction starting from top right corner. (by switching respective iterators' behavior in for loops), using always previously visited neighbors to relax value of d. Since each pixel is passed exactly 4 times for a 2D problem, complexity of this method is $O(4n)$, which is in the $O(n)$ complexity class of algorithms. (see [24], page 607, remark (4)).

An algorithm pseudocode can be seen in Alg. 4.

4.1.7 Vector Distance Transform (VDT) algorithm [25]

This is a sweeping method yielding Euclidean distance results. Logic of the sweeps is the same as in FSM, but the globalness and source-awareness of Euclidean method changes relaxation rules. Now, instead of computing actual value of d directly from the values of d of the previously visited neighbors, we check their sources. If Euclidean distance to a source of any neighbor is less than current distance value, set current value to it, and call that neighbor's source to be also current pixel's source. Complexity of this method is also *O(n)*.

This Euclidean solver is *source-aware*, as it not only keeps a record of distance values for each pixel, but it also needs to address the source pixel guaranteeing this value for each pixel. Let us introduce *source function* s, $s : \Omega \rightarrow \Omega_0$. $\forall a \in \Omega$, we can get the source of a by referring to $s(a)$.

To study algorithm pseudocode, refer to Alg. 5.

4.1.8 Novel method: Dijkstra-Pythagoras (DP)

Let us now formulate a wavefront-type method - similar to FMM, guaranteeing the narrow-bandedness of this approach, but unlike FMM, yielding Euclidean distance results.

For initialization, set $d(a) = \infty, \forall a \in \Omega \backslash \Omega_0$ and $d(b) = 0, \forall b \in \Omega_0$. Set $s(a) = undefined, \forall a \in \Omega \backslash \Omega_0$ and $s(b) = b$, $\forall b \in \Omega_0$. Label all $a \in \Omega \backslash \Omega_0$ as 'grass' and all $b \in \Omega_0$ as 'fire front'. Allocate min-heap h and insert all $b \in \Omega_0$ into it.

Algorithm 4 Fast Sweeping Method (FSM) DF algorithm pseudocode

Require: allocate $\forall p \in \Omega_0 : d(p) = 0$, allocate $\forall p \in \Omega \backslash \Omega_0 : d(p) = +\infty$

```
1:  function UPDATE(i,j)
2:      x = min(d_{i,j+1}, d_{i,j-1})          ▷ use +∞ whenever out of bounds
3:      y = min(d_{i+1,j}, d_{i-1,j})
4:      D
5:      if |x − y| < 1 then
6:          D = (x + y + sqrt(2 − (x − y)^2)) /2
7:      else
8:          D = min(x, y) + 1
9:      end if
10:     if D < d(p) then
11:         d(p) = D
12:     end if
13: end function
14: for (i=0; i<idim; i++) do
15:     for (j=0; j<jdim; j++) do
16:         UPDATE(i,j)
17:     end for
18: end for
19: for (i=0; i<idim; i++) do
20:     for (j=jdim-1; j ≥ 0; j−−) do
21:         UPDATE(i,j)
22:     end for
23: end for
24: for (i=idim-1; i ≥ 0; i−−) do
25:     for (j=0; j<jdim; j++) do
26:         UPDATE(i,j)
27:     end for
28: end for
29: for (i=idim-1; i ≥ 0; i−−) do
30:     for (j=jdim-1; j ≥ 0; j−−) do
31:         UPDATE(i,j)
32:     end for
33: end for
```

Algorithm 5 Vector Distande Transform (VDT) algorithm pseudocode

Require: allocate $\forall p \in \Omega_0 : d(p) = 0, s(p) = p$

Require: allocate $\forall p \in \Omega \backslash \Omega_0 : d(p) = +\infty, s(p) =$ '?'

```
1:  function UPDATE(i,j)
2:      p = (i, j)
3:      for all r ∈Neighborhood4(p), s.t. s(r) is not '?' do
4:          D = sqrt((s(r)_i − i)^2 + (s(r)_j − j)^2)
5:          if D < d(p) then
6:              d(p) = D
7:              s(p) = s(r)
8:          end if
9:      end for
10: end function
11: for (i=0; i<idim; i++) do
12:     for (j=0; j<jdim; j++) do
13:         UPDATE(i,j)
14:     end for
15: end for
16: for (i=0; i<idim; i++) do
17:     for (j=jdim-1; j ≥ 0; j−−) do
18:         UPDATE(i,j)
19:     end for
20: end for
21: for (i=idim-1; i ≥ 0; i−−) do
22:     for (j=0; j<jdim; j++) do
23:         UPDATE(i,j)
24:     end for
25: end for
26: for (i=idim-1; i ≥ 0; i−−) do
27:     for (j=jdim-1; j ≥ 0; j−−) do
28:         UPDATE(i,j)
29:     end for
30: end for
```

Backbone of this approach is a two-fold way of d relaxation. First few times (at most 8, one for each neighbor), the pixel a can undergo a Dijkstra-like relaxation: $d(a) := d(b) + 1$, anytime one of a's direct 4-neighbors b is being fixed, or $d(a) := d(c) + \sqrt{2}$, anytime one of a's diagonal neighbors c is being fixed, provided that this new value is $< d(a)$. Sources are handled as $s(a) := b$ or $s(a) := c$, respectively. We will see later that this design enables the algorithm to work also in constrained DF computations. The first time the pixel a is touched, it changes label from 'grass' to 'fire front' and gets inserted into the min-heap h. Each time $d(a)$ is relaxed afterwards, its $s(a)$ is updated to current relaxation source, and heap h gets heapified at a. Eventually, pixel a manages to bubble to the top of the min-heap. It checks its distance values to sources of all its already fixed neighbors (possibly at most 8, again, one check for each fixed neighbor) – this time distance is Euclidean. Then, $d(a) = min$ of these values, $s(a) = s$ of this distance-minimizing-neighbor, a gets a chance to relax all its 8-neighbors in the Dijkstra way and is finalized by being assigned a label 'burnt ground'.

Algorithm ends as soon as h becomes empty. At this time, $\forall a \in \Omega, d(a) =$ correct Euclidean distance, $s(a)$ contains correct references to source pixels, and all $a \in \Omega$ carry the label 'burnt ground'.

Each pixel can be relaxed in a Dijkstra-way at most 8 times, which has complexity $O(\log_2 n)$ per pixel due to heap swaps. Afterwards, it has to check at most 8 neighbor source distances to be finalized in a Pythagoras-way (thus the name Dijkstra-Pythagoras) – which has constant complexity $O(8 + 8) = O(16) = O(1)$ per pixel. The final algorithm complexity is $O(n. \log_2 n)$ – same as FMM. To see the pseudocode for DP, refer to Alg. 6.

Tab. 4.2 presents an overview of combinations of distance definitions and pixel visiting strategies. We see that Dijkstra-Pythagoras fills up a logical empty spot in the chart.

Algorithm 6 Dijkstra-Pythagoras DF algorithm pseudocode

Require: $\forall p \in \Omega \backslash \Omega_0 : d(p) = \infty, v(p) = 0, s(p) = \text{'?'}$, allocate *heap*

Require: $\forall p \in \Omega_0 : d(p) = 0, v(p) = 1, s(p) = p$, *heap*.insert($p$)

while *heap* != empty **do**
  a = *heap*.pop
  for each pixel $b \in$ Neighborhood8(a) , which has $v(b) = 2$ **do**
    $r = s(b)$
    $D_p = \sqrt{(r_x - a_x)^2 + (r_y - a_y)^2}$
    if $D_p < d(a)$ **then**
      $d(a) = D_p$
      $s(a) = r$
    end if
  end for
  $v(a) = 2$ ▷ a set to 'burnt ground'
  for each pixel $c \in$ Neighborhood8(a), which has $v(c) = 1$ or $v(c) = 0$) **do**
    $D_d = d$
    ($c \in$ Neighborhood4(a) ? $D_d += 1$: $D_d += \sqrt{2}$)
    if $D_d < d(c)$ **then**
      $d(c) = D_d$
      $s(c) = a$
      ($v(c) = 0$? *heap*.insert(c) : *heap*.decreaseKey(c, D_d))
      $v(c) = 1$ ▷ c set to 'fire front' anyways
    end if
  end for
end while

	Multi-visit	Wavefront	Sweeping
Eikonal equation	TRRT	FMM	FSM
Euclidean distance	BF	Dijkstra-Pythagoras	VDT

Table 4.2: Unconstrained DF methods overview. Column captions - pixel visit order policies, row captions - distance definition. Dijkstra-Pythagoras is a novel method, introduced in this work, filling up the logical gap which arised from the classification of existing methods.

4.2 Constrained Distance Function

Until this point, the novel DP algorithm may have felt like an unnecessary addition to VDT – after all, they both provide the same numerical results, and VDT is faster. In this chapter we introduce maze-like concept, where some of the pixels are impassable. We will see that it is in this case, where wavefront concept is more suitable than the sweeping one.

4.2.1 Problem formulation

Define a set of pixels $\Omega_\infty, \Omega_\infty \subseteq \Omega$, called *wall pixels*. These are impassable for the distance-carrying signal, so the information has to find its way around them. In the *maze navigation* problem, *maze* can be thought of as Ω, while Ω_∞ denotes *maze walls* and Ω_0 denotes *maze exit(s)*.

Shortest path finding methods studied here rely on computing d from Ω_0 throughout Ω while respecting the constraints set by Ω_∞. Then, it obtains the path by steepest descent method. Maze entrance (or current standpoint of an agent of interest in the maze - from our viewpoint, these are interchangeable) can be interpreted as steepest descent's initial point. We will then see that the cost of computing optimal paths from other standpoints only requires computing the new steepest descent procedure - d is computed only once. We identify Ω_∞ as a set of *constraints*, and we call the DF respecting them to be the *constrained distance function* (CDF). A method that can compute CDF would then be called *constrained distance function method* (CDF method).

4.2.2 Usability of existing DF methods for finding CDF

Since number of passes needed in sweeping approaches scales with number of turns in a maze [24] and it is often hard to estimate this number beforehand, sweeping approach is not ideal for solving complex mazes. Nevertheless, we try to experiment with the applicability of FSM in CDF environment - the results will be presented and discussed later, in the experiments section.

We don't know if there exists any modification to the brute-force Euclidean algorithm, which would make it suitable to solve this type of problem.

In maze environments, the localness of approaches based on eikonal equation is an advantage. Wall pixels can be simply regarded as having $d(w) = \infty, \forall w \in \Omega_\infty$ and both TRRT and FMM would work without further modifications. How about the newly-introduced DP? Its globalness seems to be a problem. We will, however, use its source-awareness to overcome it.

4.2.3 Method enhancement: Bresenham-Dijkstra-Pythagoras

Two fundamental questions have to be answered in order for the DP concept to work. First, how do we check if a given $a, a \in \Omega$, can see the given source $b, b \in \Omega_0$, without having its view occluded by some pixel $w, w \in \Omega_\infty$? And second, how do we treat the case when b is not visible from a?

To answer the first question, consider using rasterized line algorithms, like DDA or Bresenham's (whichever is faster on current hardware). By doing so, we are taking an advantage of pixels forming a square grid similar to screen raster. If a pixelated line from a to b is drawn, and none of this line's pixels are wall pixels, we could say b *can see* a. However, if at least one pixel of this line is a wall pixel, we have to admit that w prevents b from seeing a. This is illustrated in fig. 4.1.

To answer the second question, consider introducing a concept of *quasi-sources*. Maybe a pixel c cannot see the pixel a directly, but what if c sees pixel b, which in turn sees the pixel a? If the broken line $c-b-a$ is the shortest of all possible broken lines (considering line break-points to be allowed at pixel centers only) connecting c to a without violating the visibility condition, we can say that b serves as a quasi-source for c. To distinguish quasi-sources from sources in Ω_0, let us call $\forall a \in \Omega_0$ the *true-sources*. For a given pixel p, there can be a long list of quasi-sources, until we reach the true-source a. This is illustrated in fig. 4.1, rightmost image.

From implementation point of view, modifications to DP algorithms are minor. Firstly, we have to include visibility check before finalizing $d(a)$ in the Pythagoras-way, once a was removed from top of min-heap. Note that we don't have to check

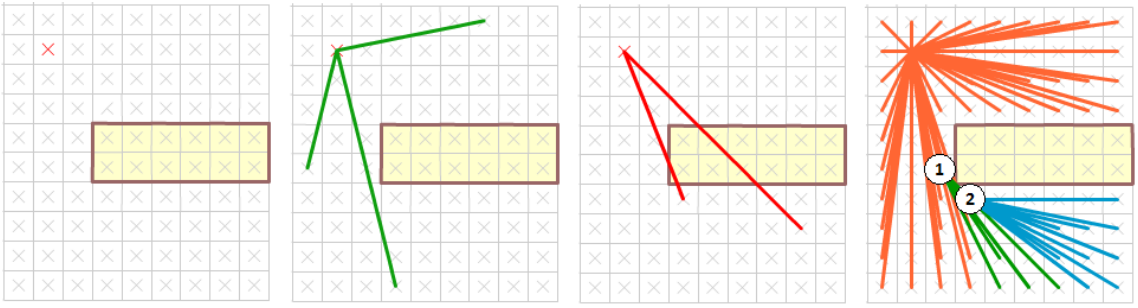


Figure 4.1: Illustration of CDF behavior. From left to right: (a) Problem formulation – from a red source in upper left part of the domain, compute d in whole gray domain, but respect the wall defined as 12 pixels in the frame. (b) Examples of pixels which can clearly see the true-source directly. (c) Examples of pixels where visibility of true-source is occluded by the wall. (d) Euclidean CDF solution: orange-connected pixels see the true-source directly, green-connected pixels can see the quasi-source 1 which sees true-source, and blue-connected pixels can see quasi-source 2 which sees quasi-source 1 which sees true-source. The distance of any pixel to true-source is computed as the length of the shortest broken line from it to the true-source, with line-breaks allowed only at the quasi-sources.

visibility when relaxing $d(a)$ in the Dijkstra-way, since sources are set to neighbors themselves, and it is clear that a pixel can see its 8-neighborhood neighbor, provided they both are non-wall pixels. For relaxations happening in the Pythagoras-way, we use the DDA algorithm to get rasterized line. Secondly, whenever a new neighbor is touched for the first time and it is a wall, we have to set it to 'burnt ground' immediately.

To see the pseudocode, refer to alg. 7. Observe that it contains only minor additions to unconstrained version - algorithm styling highlights these.

To distinguish this modified version from original Dijkstra-Pythagoras, we call this wall-aware algorithm *Bresenham-Dijkstra-Pythagoras* (BDP) due to its usage of line rasterization technique to tell source visibility.

This algorithm was successfully used to compute constrained distance function in spatio-temporal 4D tubular cell structures in order to reconstruct cell lineage trees. For details, see [29].

Algorithm 7 Bresenham-Dijkstra-Pythagoras DF algorithm pseudocode

Require: $\forall p \in \Omega \backslash \Omega_0 : d(p) = \infty, v(p) = 0, s(p) =$ '?', allocate *heap*

Require: $\forall p \in \Omega_0 : d(p) = 0, v(p) = 1, s(p) = p$, *heap*.insert($p$)

while *heap* != empty **do**
  a = *heap*.pop
  for each pixel $b \in$ Neighborhood8(a) , which has $v(b) = 2$ **do**
    $r = s(b)$
    $D_p = \sqrt{(r_x - a_x)^2 + (r_y - a_y)^2}$
    $L =$ set of pixels of rasterized straight line rendered from a to r
    if $D_p < d(a)$ **and** $\forall u \in L : u \notin \Omega_\infty$ **then**
      $d(a) = D_p$
      $s(a) = r$
    end if
  end for
  $v(a) = 2$ ▷ a set to 'burnt ground'
  for each pixel $c \in$ Neighborhood8(a), which has $v(c) = 1$ or $v(c) = 0$) **do**
    if $c \notin \Omega_\infty$ **then**
      $D_d = d$
      ($c \in$ Neighborhood4(a) ? $D_d += 1$: $D_d += \sqrt{2}$)
      if $D_d < d(c)$ **then**
        $d(c) = D_d$
        $s(c) = a$
        ($v(c) = 0$? *heap*.insert(c) : *heap*.decreaseKey(c, D_d))
        $v(c) = 1$ ▷ c set to 'fire front' anyways
      end if
    else
      $v(c) = 2$
    end if
  end for
end while

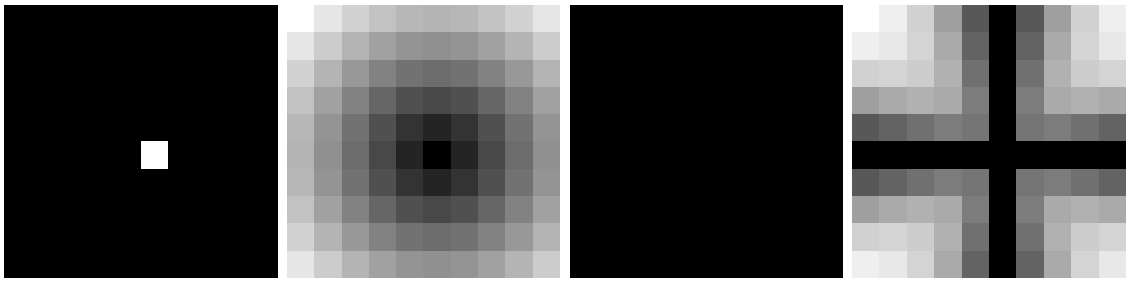


Figure 4.2: Results of one dot experiment. From left to right: input Ω_0, resulting d (result yielded by BF, but the result yielded by e.g. FSM is visually indistinguishable from this one), difference between results of BF and analytical, difference between results of FSM and analytical. There is no numerical difference between BF result and analytical solution. Range of difference between FSM result and analytical solution is [0, 0.6355]. Difference results are shown rescaled, white color meaning largest error.

4.3 Distance function experiments

4.3.1 One dot experiment

The first experiment was intended as an exercise to validate implementations of six studied algorithms. We computed DF on a 10x10 pixel image, where the pixel (5,5) was the only source pixel. We found it interesting that the numerical results of all Euclidean distance-based algorithms (BF, DP and VDT) were the same, and the results of all eikonal-based algorithms (TRRT, FMM and FSM) were also the same, although these were different than the results of Euclidean distance-based set of algorithms. For this simple example, we computed an exact analytical solution, by simply setting each pixel's d to Euclidean distance from pixel (5,5). Results are presented also visually in fig. 4.2.

4.3.2 Dots experiment

In the second experiment, we wanted to measure CPU time consumption and its dependence on the number of sources. We created three 1000x1000 pixel images, containing 1,000; 10,000 and 100,000 randomly-placed source pixels, respectively. The results are presented in tab. 4.3. It can be observed that Brute-Force method scales roughly linearly with the number of sources. TRRT exhibits negative scaling

Method	BF	TRRT	DP	FMM	VDTM	FSM
CPU time 1000 source px	10s	9s	2s	1s	1s	0s
CPU time 10000 source px	104s	5s	2s	2s	1s	0s
CPU time 100000 source px	1083s	4s	3s	2s	1s	0s

Table 4.3: Dots experiment shows computational time of studied methods and its dependence upon the number of source pixels.

w.r.t. the number of sources, and so does Dijkstra-Pythagoras. Other methods performed quickly and their performance was only slightly affected by the number of sources.

4.3.3 Diagonal line experiment

In the third experiment, we wanted to exploit the qualitative difference between eikonal-based and Euclidean distance approaches as clearly as possible. We created a 20x20 pixel image, placed a rasterized straight line to be passing from its top-left corner to the bottom-right one, and we wanted to compute the DF from this line. Eikonal-based methods solved this task with zero error w.r.t. analytical solution (to our surprise - notice that these same algorithms are incapable of solving a much simpler problem, to compute the distance from a single point, exactly). Euclidean distance methods introduced the largest error in the first diagonal below and first diagonal above the main diagonal - using the matrix algebra vocabulary - with error magnitude descending further from diagonal. The reason for this difference lies in the fact that eikonal solvers tend to interpolate interconnected neighboring source pixels and see them as forming an interface, while Euclidean distance solvers see these points as separated and treat them as such.

To diminish this phenomenon, it is suggested to use the so called *shell boundary condition* [27]. That means, if we want to compute the distance to an interface with a very good precision, we should consider including not only the pixels with 0 distance value into Ω_0, but also some narrow band of near-zero-distance pixels, to formulate an interconnected interface. Notice that in this case, d at these pixels should not be

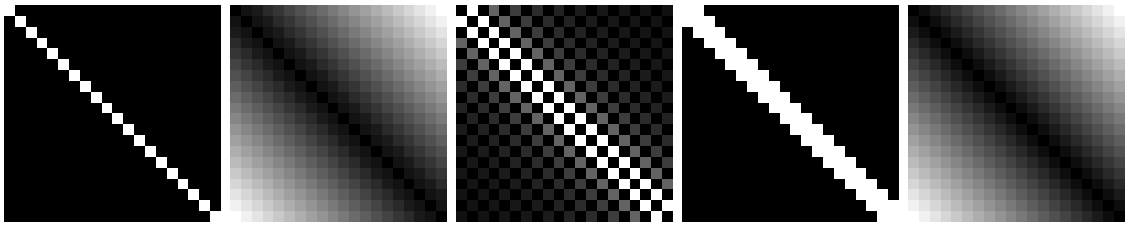


Figure 4.3: Results of diagonal line experiment. From left to right: (a) input Ω_0 without considering shell boundary condition - white color of pixel p means that $p \in \Omega_0$, (b) resulting d (result yielded by BF) prior to considering shell boundary condition, (c) difference between results of BF and analytical - largest error value is in white and has value 0.293 and black is 0 error, (d) input Ω_0 considering a shell boundary condition, (e) resulting d - result yielded by BF - after including shell boundary condition - this results contain no error w.r.t. exact analytical solution

set to 0, but to their exact, analytically computed value, to achieve the best precision of DF result - this is an exception to the rule that $d(a) = 0, \forall a \in \Omega_0$. All algorithms have to be slightly modified then, at least the initial setting of $d(a) = 0, \forall a \in \Omega_0$, has to be removed.

We introduce the shell boundary condition by considering main diagonal pixels to carry value $d(a) = 0$ and first diagonal below and above pixels to carry the exact value $d(a) = \sqrt{2}/2$ - this is the distance between center of a given pixel and a point which is a projection of this point on the line passing through the main diagonal. After introducing this modification, results of all algorithms contained zero error w.r.t. the exact solution. Results are visualized in fig. 4.3.

In general, formulating a shell boundary condition for arbitrary shape is a non-trivial task.

4.3.4 Circle experiment

In this experiment, we wanted to measure precision of the results yielded by DF methods. For a 100x100 pixel image, we calculated an exact analytical euclidean distance to a circle of radius 25 pixels, centered at [49.5,49.5] - center of this circle happens to be in the middle of the picture if array indices are zero-based (common way of indexing arrays in programming languages). We used shell boundary condition - all pixels, for which the values of analytical solution were < 1, were

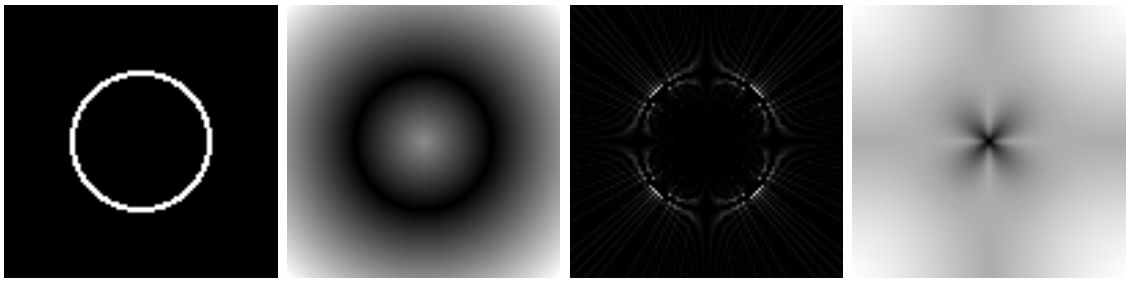


Figure 4.4: Results of circle experiment. From left to right: (a) input Ω_0, (b) resulting d - shown result yielded by BF, but again, results yielded by all other methods are visually indistinguishable from this one, (c) difference between results of BF and analytical, (d) difference between results of FSM and analytical. Range of difference between BF result and analytical solution is [0, 0.303]. Range of difference between FSM result and analytical solution is [-0.755, 0.355]. Difference results are shown rescaled, darker intensities map to smaller values and lighter intensities are mapped to higher values.

considered to be the part of the circle, they were included to Ω_0 and d was set to exact analytical value so that no error is introduced in the initialization. Results of all three Euclidean distance methods are numerically equal also in this experiment. Results of all three eikonal equation-based distance methods are also numerically equal. The character of error for Euclidean versus exact and for eikonal versus exact is demonstrated in fig. 4.4.

4.3.5 Round-cornered square experiment

In this experiment, we wanted to measure result precision with a more complicated shape, but with one for which we still can tell the analytical solution. We chose a round-edged square. In an image 200x200 px, we placed 150x150 px square into the centered position. Roundness of the square corners was achieved by substituting them by quarters of circles of radius 25x25 px. We used shell boundary condition also in this case. Results for Euclidean-based methods are all the same, and results for eikonal-based methods are all the same as well. Results can be seen in fig. 4.5.

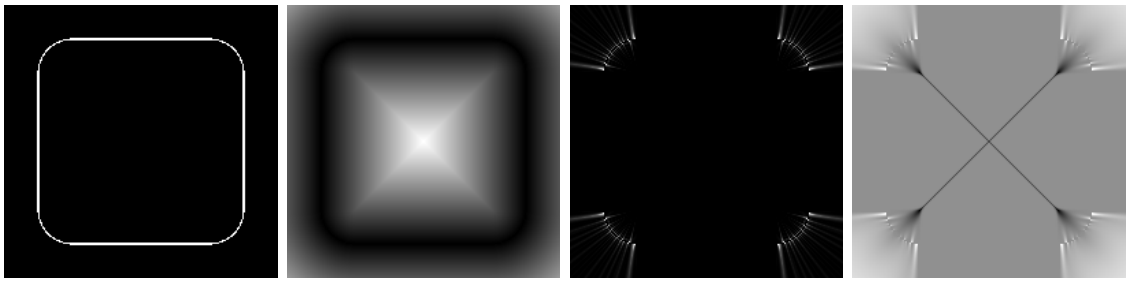


Figure 4.5: Results of round-cornered square experiment. From left to right: (a) input Ω_0, (b) resulting d - result yielded by BF, (c) difference between results of BF and analytical, (d) difference between results of FSM and analytical. Range of difference between BF result and analytical solution is [0, 0.751]. Range of difference between FSM result and analytical solution is [-0.702, 0.538]. Difference results are shown rescaled. In order to understand image scaling, observe that the median intensity (the color present in the vast majority of pixels) happens to represent error= 0 in both difference results.

4.3.6 Maze experiment

To compare the performance of CDF methods, we used them to solve the path finding problem in a complicated maze. Maze image is 810x810 pixels and its pathways are 6 pixels wide, and is taken from [30]. Entrance is the upper-left-most pixel; exit is the lower-right-most. See the input in fig. 4.6.

We measured the time it takes for CDF methods to compute CDF. Results can be seen in tab. 4.4 - see 'CPU time' row.

Having the DF results, we can compute the path from entrance to exit.

While formulating the steepest descent procedure, the advantage of having a source-to-source map in case of BDP becomes obvious - unlike in FMM case, we don't have to take steps in the domain of neighboring pixels, but we can jump from quasi-source to quasi-source, until we reach the true-source (maze exit).

At first, we see that in complicated mazes this dramatically reduces the number of pixels visited - in this case, only approximately one tenth of pixels was visited when compared to FMM path reconstruction in this specific maze. This property may be especially useful in problems where we have to compute steepest descent from many standpoints for a fixed configuration of maze walls and maze exits.

Secondly, the length of the broken line from entrance to exit is the same number

as the DF result value - while in FMM these two numbers differ (in our experiment, the relative difference of these two values was about 2.1% in FMM case). This property of BDP may prove to be of help in scenarios where the agent has to know how long the road will take, before he decides to take it.

And thirdly, while to represent the path reconstruction visually, we could take an advantage of the fact that we have a line rasterization algorithm implemented and use it not only to tell if two points can see each other w.r.t. constraints, but also for broken line painting. Note that we couldn't use line rasterization while painting FMM path though - the problem is that we don't have a notion of which pixels are the line-breaking ones. As a result, path reconstructed from BDP and visualized using line rasterization algorithm resembles the movement of an agent in a maze more closely than the path reconstructed by FMM, which has an 8-way character.

We tried to analyze and compare results of FMM and BDP also quantitatively. They can be seen in tab. 4.4 and fig. 4.7. Results show, that:

1. BDP's broken line is shorter than FMM's.

2. BDP had to visit approximately 10x less points than FMM in this specific case, in order to reconstruct the steepest descent path, taking advantage of $s(a), \forall a \in \Omega$.

3. value of $d(a)$ is the same value as the length of the broken line from a to the maze exit, in BDP. In FMM, these values are different.

4. Path characters are different: while FMM's broken line has 8-way character, BDP's path looks like pixelated broken line, which resembles more closely the shortest path in a maze. This can be seen in the fig. 4.7.

A 4D implementation of BDP algorithm was used to reconstruct cell lineage trees, by computing CDF in spatio-temporal 4D tubular cell structures and subsequent steepest descent computation. More information can be found in [29].

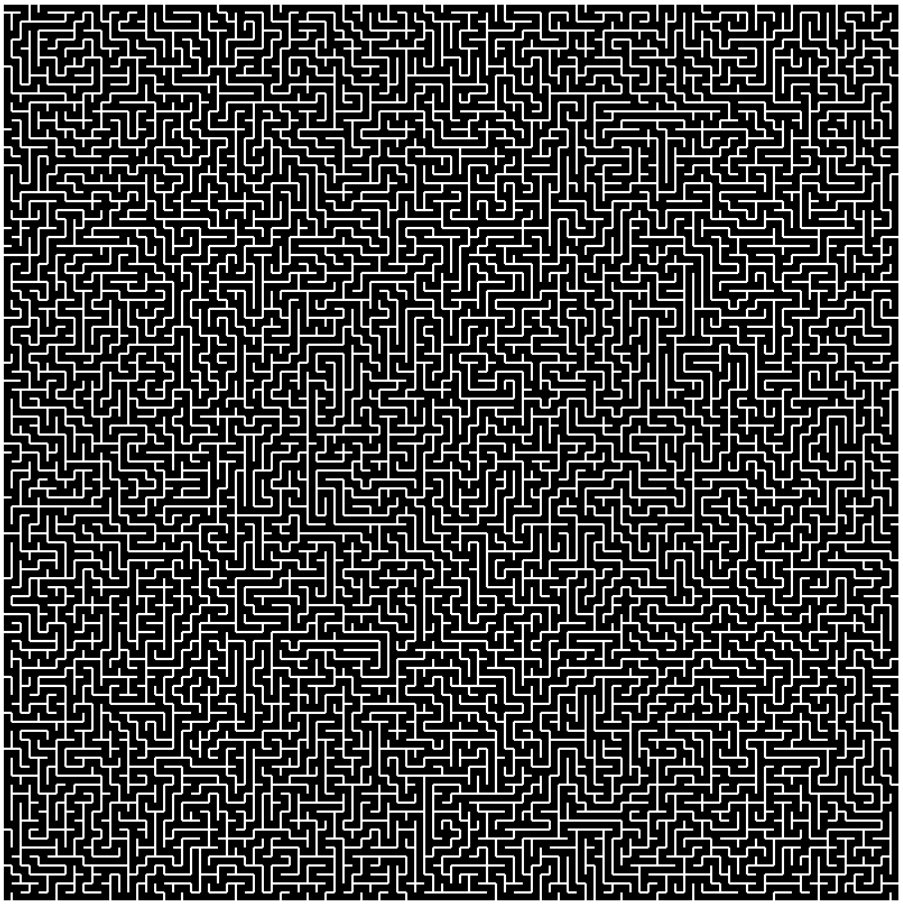


Figure 4.6: Maze experiment: input image, a maze of 810x810 pixels, pathways are approximately 6 pixels wide, walls are white, entrance is top left, exit is bottom right.

Algorithm	FMM	BDP
CPU time	1s	2s
Path length	12016.4	11332.9
d(start)	11760.1	11332.9
# of visited points	10265	1201
Path character	8-directional	rasterized line

Table 4.4: Comparison of FMM and BDP behavior. BDP's DF value at the start equals the length of shortest path, while for FMM these quantities are different. Number of visited points differs, since different steepest-descent methods are used: FMM goes pixel-by-pixel, while BDP goes source-by-source. Therefore, this value for BDP is related to number of corners visited, while FMM results represents number of pixels in the 8-connected path. The last quality, path character, is best illustrated by fig. 4.7

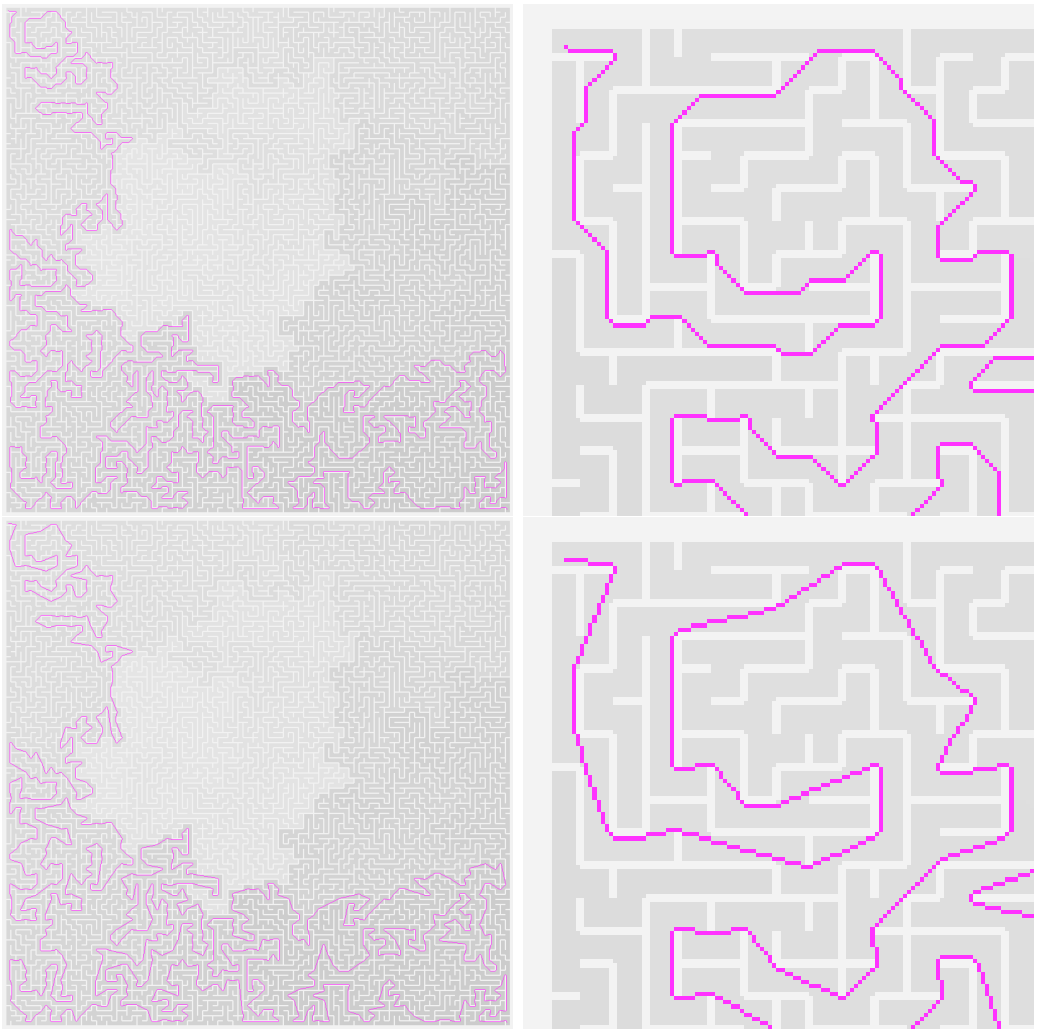


Figure 4.7: Results of maze experiment. Upper row: DF yielded by FMM and steepest-descent solution, left - full solution, right - zoom. Lower row: DF yielded by BDP and steepest-descent solution, left - full solution, right - zoom. Compare the different path character of FMM (left) and BDP (right) - better visible while looking at zooms. FMM's steepest descent has to construct path in a pixel-by-pixel manner, as it only has local information. BDP's steepest descent can be formulated using source information and path goes from corner to corner – it has a source-to-source character. BDP's path was painted using line rasterization algorithm DDA.

4.3.7 Trying to solve maze with Fast Sweeping Method

In this experiment, we attempt to use Fast Sweeping Method to solve constrained distance function problem. The algorithm (4) can be easily modified to ignore relaxation updates for wall pixels. Notice, however, that while 4 sweeps in each of the diagonal directions are enough to obtain a solution in 2D case for unconstrained distance function problem, this number of sweeps doesn't assure that signal gets 'sweeped' throughout the whole domain, for this constrained distance function case, due to the walls preventing it to go around the corners.

Let us call the original approach a *4-sweep*: that means making 4 sweeps in all 4 diagonal directions always in the same order. A natural way to cover larger part of the computational domain Ω increase the number of 4-sweeps performed in succession. The only way to tell the number of sweeps which cover the entire domain beforehand, is to know the character of maze beforehand. In a very complicated maze, like the one from previous experiment - see fig. (4.6), it is, however, difficult to tell the upper bound of the number of sweeps.

We tried to make 1, 10, 100 and 1000 4-sweeps in order to solve this case. Results can be seen in fig. (4.8). We can conclude that it took several hundreds of 4-sweeps to get the solution, which took about a minute of CPU time. Notice that both wavefront-type methods, FMM and BDP, studied in the previous experiment, can obtain the solution of this example faster. This method is therefore suitable only in cases, where we know the character of maze, so that we can bound the number of 4-sweeps needed to cover the domain of interest.

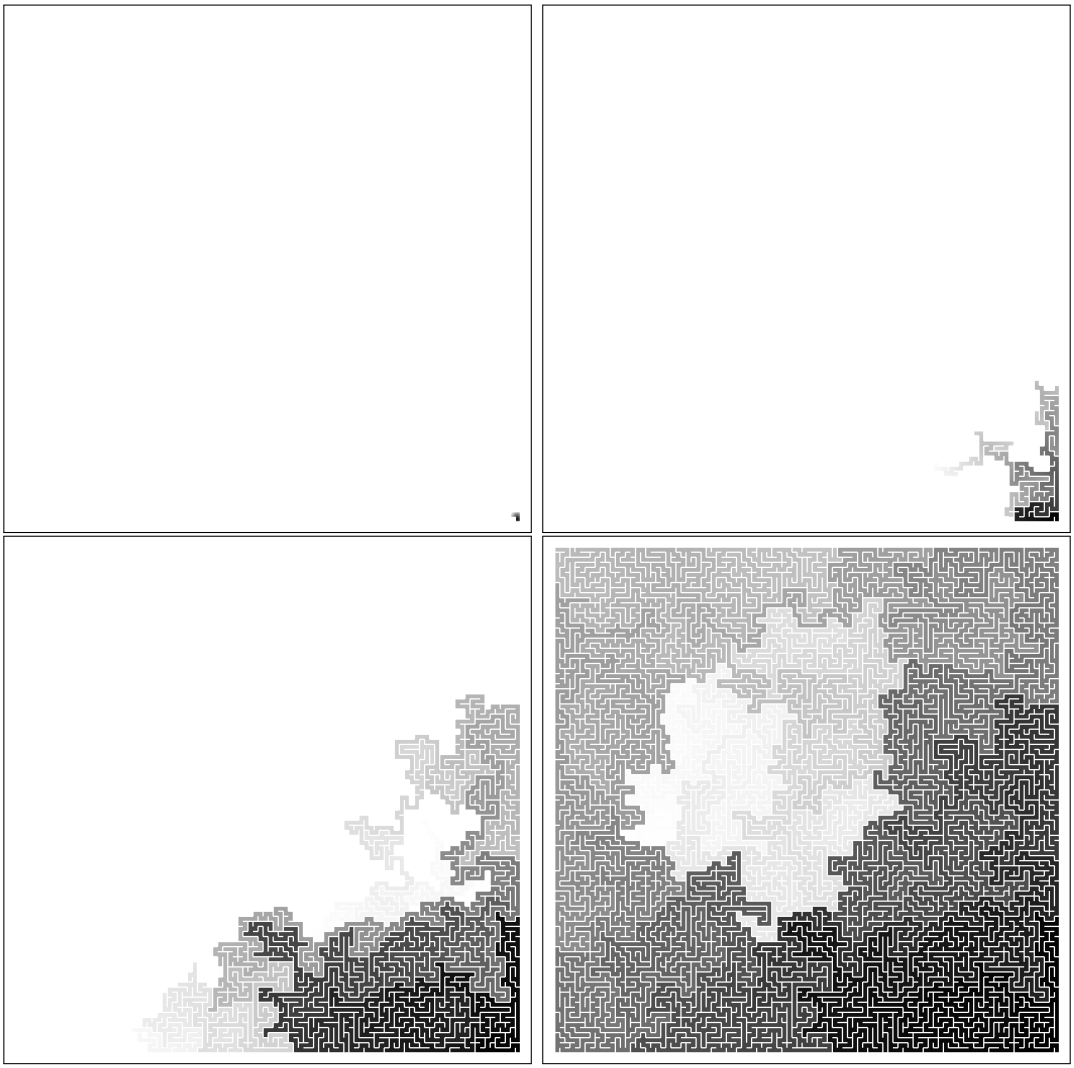


Figure 4.8: Illustration of FSM in CDF environment. Top left: 4-sweep - notice the small black part in bottom right corner of this image - this is how far signal travels from a single source point $\in \Omega_0$, in bottom right, until it is stopped by maze walls. Top right: 4-sweep repeated 10 times. Bottom left: 4-sweep repeated 100 times. Bottom right: 4-sweeps repeated 1000 times. We can see that it took at least several hundred 4-sweeps to obtain a solution for the whole domain. CPU time spent obtaining these solutions were 0, 1, 8 and 66 seconds.

5 Software

This section is written less formally, serving as a combination of developer's diary, specification of the SliceViewer software, and the user manual.

5.1 State-of-the-art software

In order to visualize the data, we started by using an existing software, a robust scientific data viewer, called ParaView. We identified three viewing options as being useful:

1. Volume rendering of 3D volume + small 3D spheres centered at FBLSCD-given cell identifier. Volume rendering is performed as mapping image intensity to color tone and transparency
2. Translucent isosurface view (depending on intensity value) + small 3D spheres
3. 2D slice of data + 2D or 3D marks for cell identifiers

An example of these viewing modes can be seen in the figure 5.1. We found out, that the inability of first two approaches of showing the interior cells of the organism volume in detail, due to surface cells occluding the interior ones, happened to be a major drawback. For the third method, we have seen the speed of file loading and data manipulation as being insufficient. So we decided to create a more specialized software to serve our needs. We needed it to view spatio-temporal volumes of 3D+time data, and we needed to tackle the challenge of projecting them onto the 2D surface of computer monitor, with as little information loss as possible. For the sake of simplicity and speed, we decided it was not needed to rotate the view. The software was meant to show 2D slices based on 2D graphics libraries, and we called it SliceViewer.

5.2 SliceViewer

This software was designed to be able to load and view a large amount of 3D+time data. We chose the Microsoft Visual Studio platform, programming language C++,

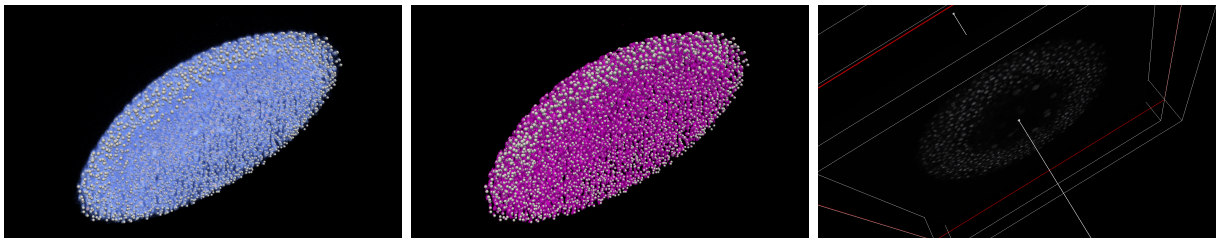


Figure 5.1: Visualizing with ParaView. Left - volume rendering, middle - translucent isosurfaces, right - viewing as a 2D slice.

and the standard graphics library. The data files are of type .vtk, containing $x_{max} * y_{max} * z_{max}$ 8bit chars, and there are θ_{max} of them. In the dataset 070418a, $z_{max} = 104$, $y_{max} = 512$,$x_{max} = 512$, and there are 320 time steps recorded. We stored the data as bitmaps of dimension $x_{max} \times y_{max}$), and we would have $z_{max} * n_{Timesteps}$ of them. $n_{Timesteps}$ is limited by computer's RAM size. We tested it on a computer with 8GB RAM, and we were able to load approximately 70 time steps of $x_{max} * y_{max} * z_{max}$ data, using 24bit RGB color encoding (one byte for each of the color channels R, G and B). This limit may be pushed further via color pallette encoding, if needed, or considering 8bit greyscale pallette.

We required it to be able to view specific z-coordinate slice and θ-time without further computation, so the bitmaps are initialized at the time of loading from HDD. Two sliders (one for z, one for θ) are used to control the viewed slice. Computational burden of changing z and θ values of the sliders is therefore slim - GraphicsView only has to swap the currently shown bitmap for another.

The sketch of the software user interface layout and the detailed view of the control panel can be seen in the figure 5.2.

We wanted to make sure the marks representing the cell identifiers are as clear to see in the data as possible, but don't cover up much of the data, so we chose upright equilateral crosses. Using 3D crosses means user can see the presence of the cross even several slides over- and under its occurence in the z-direction. Lenght of the 3D cross arm was chosen to be 5 pixels, and the color combination was chosen red-green, green are arms of the cross from center in the axes directions, red are arms against axes directions. This combination of colors is well visible in the grayscale

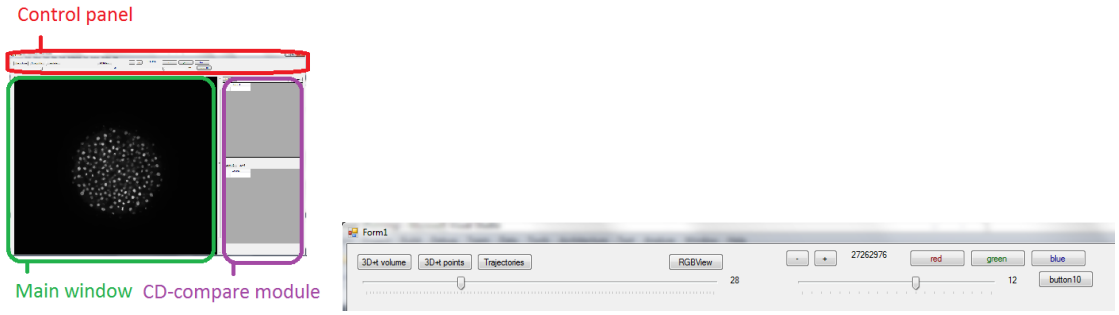


Figure 5.2: SliceViewer design. Left - miniature of the user interface. Right - detailed view of the control panel.

environment.

5.3 RGB viewing mode

We wanted to have an option to see time correspondence between time steps, which is not an easy task to do, in 2D slice approach. Fast reaction time of the viewer when moving the θ-axis slider is a help towards achieving this goal, but we wanted to see it even without moving sliders. We took the advantage of having intensities in range 0..255 and, using standard 3-channel RGB color encoding of computer graphics, created an RGB viewing mode: for a θ, we view a 2D slice in such a way that the red channel represents slice intensity in the previous frame $(\theta - 1)$, the green channel views the current frame (θ) and the blue one views the next frame $(\theta + 1)$.

It was interesting to see various cell behaviors visualized in this mode. Cells standing more or less still are seen as predominantly white-color blobs as the three channels overlay in the same location, sometimes with colourful edges (caused by small cell motion). Cells traveling in a certain direction are seen as R-G-B blob patterns. Cells undergoing mitoses are seen as B-G-R-G-B patterns. The result of this color encoding is the simplification of the non-automatic human-eye cell mitosis detection. All this patterns are visualized in the fig. 5.3

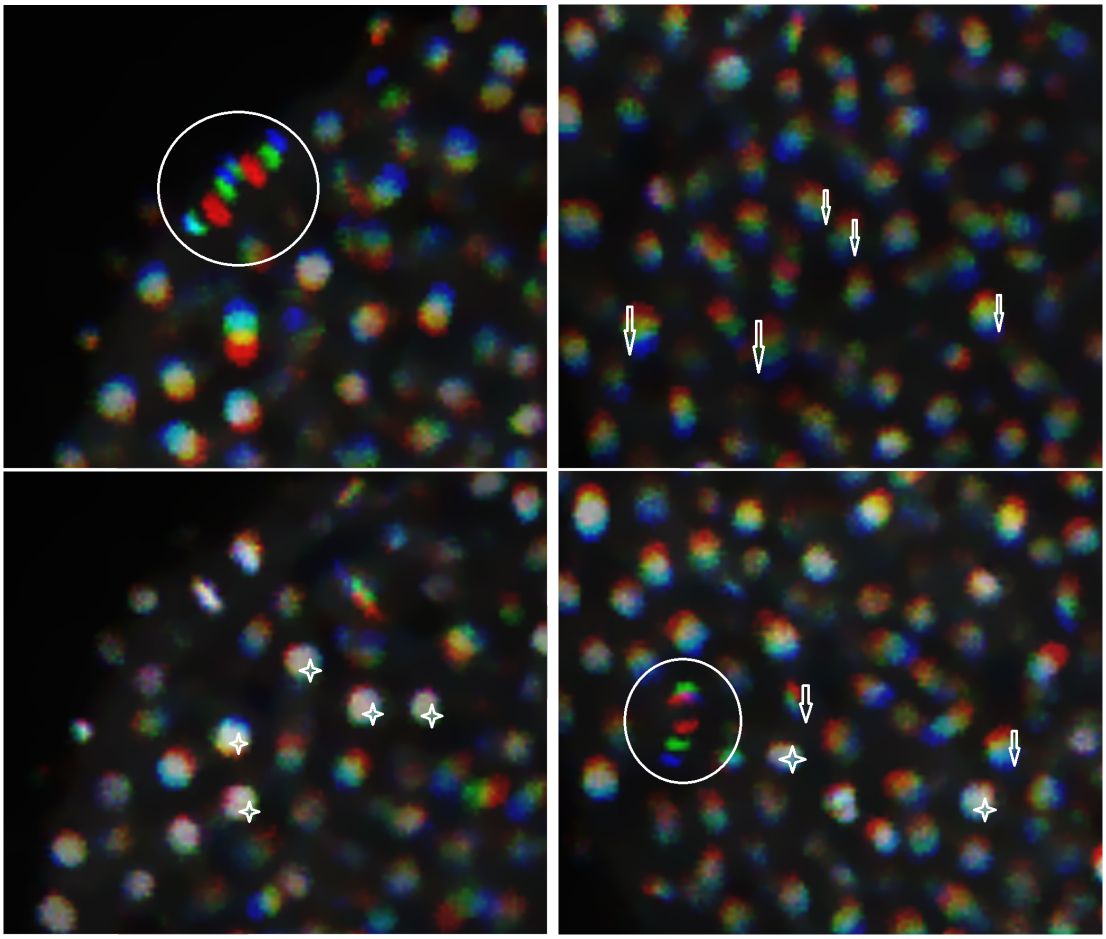


Figure 5.3: RGB mode of the SliceViewer software. From top left to bottom right: Top left - see two neighboring cells undergoing cell divisions at the same time, each creating two daughter cells in the circle. Top right - see multiple cells traveling in the direction from top to bottom of the image, denoted by the arrows. Bottom left - see a few cells standing more or less still, with large 3-channel overlays creating predominantly white areas. Bottom right - a typical view of random embryo subvolume, where all these types of behavior can be observed.

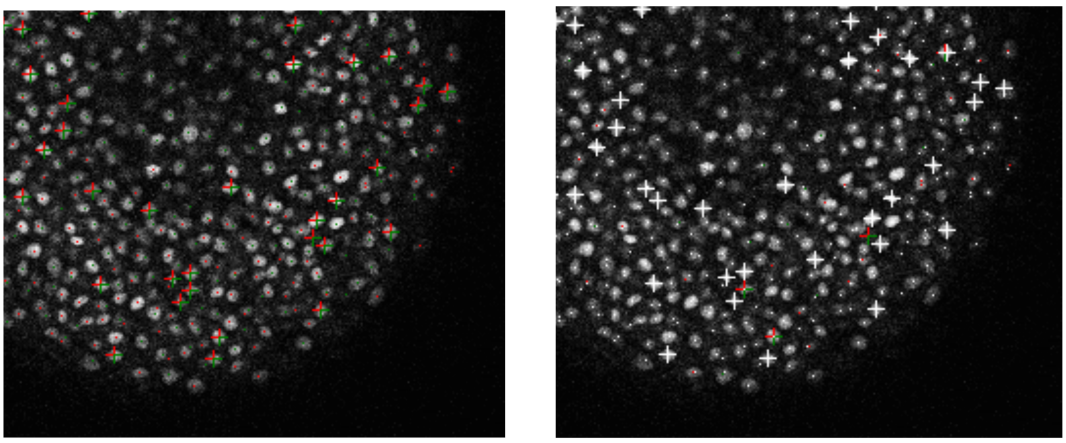


Figure 5.4: Basic strategy results visualization, as a comparison. Left - detail of input image with 3D crosses representing results of FBLSCD at time $\theta = 16$ (optimal stop-time for this dataset, this method and at this frame according to chapter 3.4). Right - the same content, with results of LSOpen at time $\theta = 17$ (optimal stop-time for this dataset, this method and at this frame) added. Viewing data this way, user can spot false positives of the first method, compared to the second method. (FBLSCD yields a false positive here, which can be seen in the right image - it is the one slightly to the right and down from the image center. Notice how difficult it is to spot it in the left image, among many other red-green crosses.)

5.4 Mutual comparison of two cell detection results

We specified the requirements for being able co compare two different cell detection results in chapter 3.4. Slice Viewer therefore enables the user to load two cell identifier sets. On loading the first of them, crosses are painted in color. On loading the second, its crosses are superimposed, painted white now. All those points, which are detected by the first strategy, but left undetected by the second, remain painted in color, while the others are covered over by white. In order to see the vice-versa results, i.e. points detected by the second and left out by the first, user simply loads the sets in the switched order. This initial mode of comparing two cell detection strategy results can be seen in the fig. 5.4.

The second requirement stated in chapter 3.4 was for the software to be able to compare the two sets of points not only binary, but also via a certain metric. The rightmost part of the main window contains two tables. To compare two center detection strategies, 1 and 2, at first the user clicks the button above the first table

and selects centers of method 1 and distance function of centers found by method 2. Then he clicks the button above the second table and selects centers of method 2 and distance function of centers found by method 1. This way, table shows list of cell centers by each method, with their ID, $x - y - z$ coordinates, and the distance to its closest center in the other method, sorted descending w.r.t. distance value.

Having this interface implemented, we already were able to load two results of different cell detection strategies and perform a basic comparison. At this stage, the interface was able to highlight the differences: for each method, it pointed out the points detected as cell identifiers, which were missed by the other method.

In addition to this functionality, we required to not only show missed cells, but also point out, if one cell was detected by more than one identifier. So, to the two tables to the right, we added two more columns - for strategy A, the first of them carrying the ID of corresponding closest point in the steepest descent sense in the strategy B, and the second column carrying the number of identifiers in method B, who consider a given point in strategy A as their closest, in the steepest descent sense, again. A button to compute steepest descent from each point in the strategy A results to strategy B results points, and vice versa, was added.

With this last important addition, we were able to not only compare two methods w.r.t. missing detected centers, but also w.r.t. multiple detected centers.

Both tables are interactive. On selecting an arbitrary table entry, z-dimension slider is set to this entry's z, a red-green 3D cross is painted in the location, an orange-cyan 3D cross is painted in the location of its steepest-descent neighbor in the other strategy result set. Furthermore, all cell centers from the other method, who call this selected point to be their closest peer, are marked by blue-pink 3D crosses.

For further reference, see [28].

This final user interface is viewed in fig. 5.5.

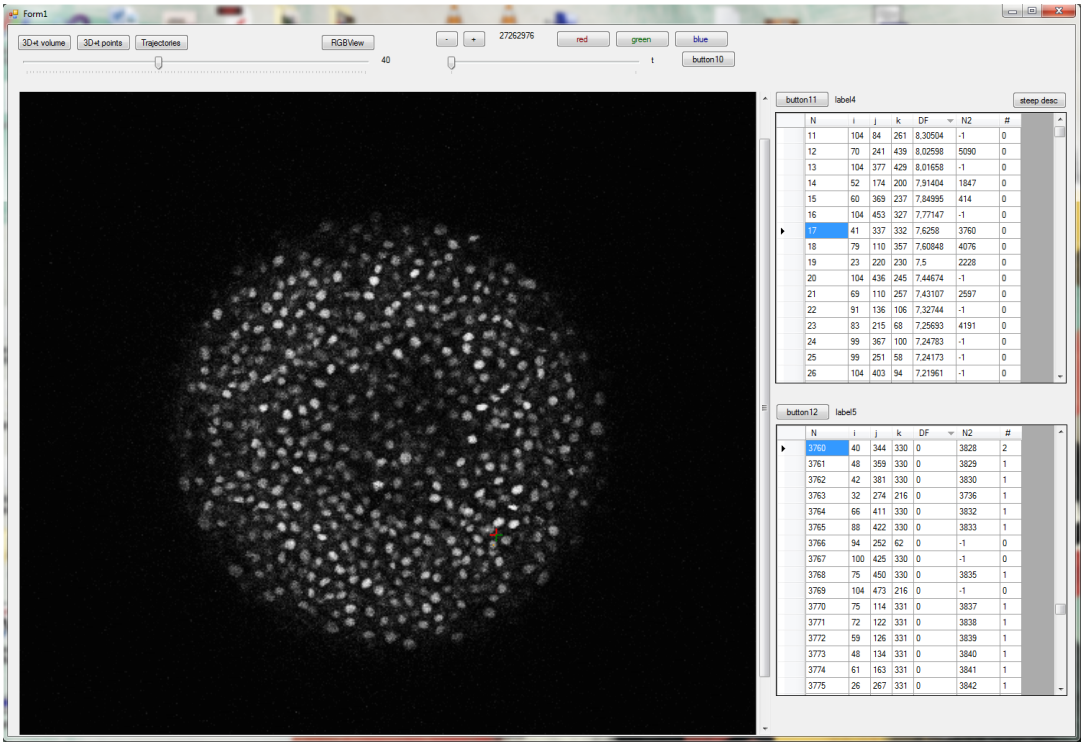


Figure 5.5: User interface designed for comparing two cell detection strategy results. Notice the mutual correspondence of highlighted lines in the right control panel, as well as visual representation of this event in the main viewer area - pink tip of the second 3D cross can be seen few pixels lower from the red-green one.

6 Experiments - comparing cell detection results

The first experiment was comparing the state-of-the-art results, according to [2], with centers obtained as results of FBLSCD with stop time computed automatically, using the algorithm proposed in the chapter 3.3. In addition to this, we also compared FBLSCD autostop with LSOpen autostop. Both FBLSCD and LSOpen stop times are chosen as first local minima of 5th degree polynomial fit. We compared and classified visually all the centers for which their closest peers in the other method have distance ≥ 6.0. See the results in the fig. 6.1. Using LSOpen results, it can be seen that FBLSCD indeed contains some false positives, which would be difficult to spot without comparing it to some other method. It depends upon the application, if this is an acceptable number of errors.

It is worth noting at this point, the different impact of false positives and false negatives on the results of our goal, which is either computing statistical characteristics of the zebrafish embryo, or reconstruction of the cell lineage tree. For both of this goals, there is a feasible method for processing multiple detected cell centers in one nucleus designed, see [2], [6], resp. A missing cell center in one frame is also a problem that can be solved with relative ease, in the cell lineage tree reconstruction task[6]. On the other hand, false positives may cause much more errors in both these frameworks, therefore, minimizing this type of error seems to be a feasible metric of the cell detection algorithm's quality.

The second experiment was comparing the state-of-the-art, optimal stop time FBLSCD and optimal stop time LSOpen results, to biological ground truth data, which is available for this dataset. Results show that in terms of cell detection differences, the state-of-the-art dataset resembles biological ground truth data the most closely of all three methods, considering the number of false negatives. While LSOpen is the furthest from it in this criterion, in the number of false positives it yields the results closest to the biological ground truth data. Results can be seen in the fig. 6.2.

Both experiments show however, that FBLSCD may yield some potentially

	A	B	C	D	E	F	G	H	I
1	θ	0	50	100	150	200	250	300	
2									
3	**FBLSCD: visual vs. automatic**	15 vs 16	15 vs 16	15 vs 16	15 vs 17	15 vs 16	15 vs 16	15 vs 16	
4	false positives	1					1	2	
5	correct positive, missed by the other method	8	4	2	2	7	8	10	
6	correct but noisy, missed by the other method		1				2	4	
7	a single nucleus detected multiple times		1	1	3	3		2	
8	SUM	9	6	3	5	10	11	18	
9									
10	**FBLSCD: automatic vs. visual**								
11	false positives	1		1			1	1	
12	correct positive, missed by the other method		1	1	3	1		2	
13	correct but noisy, missed by the other method								
14	a single nucleus detected multiple times			1	1				
15	SUM	1	1	3	4	1	1	3	
16									

	A	B	C	D	E	F	G	H	I
1	θ	0	50	100	150	200	250	300	
2		16 vs 17	16 vs 17	16 vs 17	17 vs 17	16 vs 17	16 vs 17	16 vs 17	
3	**FBLSCD: automatic vs LSOPEN: automatic**								
4	false positives	4	11	1	1	4	6	8	
5	correct positive, missed by the other method	6	17	26	27	51	71	97	
6	correct but noisy, missed by the other method	1	1			1	8	3	
7	a single nucleus detected multiple times	4		6	3	2	1		
8	SUM	15	29	33	31	58	86	108	
9									
10	**LSOPEN: automatic vs. FBLSCD: automatic**								
11	false positives	2			1				
12	correct positive, missed by the other method	21	24	3	3	7	9	10	
13	correct but noisy, missed by the other method						6	8	
14	a single nucleus detected multiple times	1							
15	SUM	24	24	3	4	7	15	18	
16									

Figure 6.1: Visual comparison of different cell detection strategies. Upper table - classification of dissimilarities between FBLSCD state-of-the-art stop time and automatic stop time. Upper half of this table compares state-of-the-art centers against distance function to the autostop centers, lower half vice versa. Lower table - dissimilarities between FBLSCD and LSOpen automatic stop time results. Upper part of this table compares FBLSCD autostop centers against the distance function to the LSOpen autostop centers, the lower part vice versa. Notice the red lines - while upper table contains no significant differences, in the first line of the lower one, we can see that LSOpen seems to accuse FBLSCD of yielding some false positives.

	A	B	C	D	E	F	G	H	I
1		θ=0	θ=50	θ=100	θ=150	θ=200	θ=250	θ=300	
2	GOLDEN STANDARD (GS) vs FBLSCD: visual stop time								
3	positives in GS, not found in FBLSCD (false negatives in FBLSCD*)	137	123	34	31	65	138	187	
4	positives in FBLSCD, not found in GS (false positives in FBLSCD*)	9	9	5	6	5	10	6	
5									
6	GOLDEN STANDARD (GS) vs FBLSCD: automatic stop time								
7	positives in GS, not found in FBLSCD (false negatives in FBLSCD*)	146	127	36	32	72	144	199	
8	positives in FBLSCD, not found in GS (false positives in FBLSCD*)	9	9	8	7	4	9	4	
9									
10	GOLDEN STANDARD (GS) vs LSOPEN: automatic stop time								
11	positives in GS, not found in LSOPEN (false negatives in LSOPEN*)	135	123	63	57	127	213	292	
12	positives in LSOPEN not found in GS (false positives in LSOPEN*)	9	3	3	3	3	3	1	
13									
14									
15	* - if we assume golden standard is flawless								
16									

Figure 6.2: Comparing state-of-the-art, FBLSCD autostop and LSOpen autostop to biological ground truth data. Notice that while LSOpen yields the most false negatives, it contains the lowest count of false positives. When considering which of these to use in an application, one has to pay attention to this tradeoff.

harmful false positives, although their number is relatively low.

The third experiment can be seen as an experimental insight into the LSOpen algorithm's behavioral analysis. This information is of value, since due to composite character of LSOpen method, we cannot use standard tools of mathematical analysis to study it theoretically. We computed 17 LSOpen steps with a certain step size. Then we halved the step size, and wanted to see, which LSOpen step yields the closest results to the results of the former computation. From steps 31, 34 and 37 of LSOpen halfstep, 34 has the fewest differences - 31 halfstep seems to happen sooner than 17 fullstep, and 37 halfstep seems to be later than the 17 fullstep. It is therefore reasonable to assume, that halving step size and doubling step count yields approximately the same results as the original setup. It is a known property for advection-diffusion-type equations, but it seems to work also for the non-atomic morphological operation such as LSOpen. Results of this experiment can be seen in the fig. 6.3.

All the experiments were performed using the SliceViewer software.

	A	B	C	D	E	F	G	H	I
1		t000	t050	t100	t150	t200	t250	t300	
2	LSOPEN_17 vs LSOPEN_HALFSTEP_31								
3	positives in 17, not found in hs_31	0	2	2	2	1	5	3	
4	positives in hs_31, not found in 17	10	10	4	10	13	25	30	
5									
6	LSOPEN_17 vs LSOPEN_HALFSTEP_34								
7	positives in 17, not found in hs_34	1	0	2	6	3	8	7	
8	positives in hs_34, not found in 17	2	1	1	0	1	1	0	
9									
10	LSOPEN_17 vs LSOPEN_HALFSTEP_37								
11	positives in 17, not found in hs_37	11	9	9	17	15	27	30	
12	positives in hs_37, not found in 17	0	2	0	1	3		0	
13									

Figure 6.3: Analyzing the behavior of LSOpen. Compute 17 LSOpen steps. Then, take half the step, and observe, how many steps are to be taken, in order for the results to be closest to former. It seems that 34 is the correct answer, out of three options: 31 yields far too many positives and 37 too many negatives.

7 Conclusions

We started by the discussion about the problem of detection of cells in biological image data, and described the FBLSCD algorithm. We introduced a novel method for defining a fully automatic stop time criterion. We formulated a novel algorithm called LSOpen, as a fusion of morphological and PDE-based image processing, which can perform the detection of cells and is robust against the salt-type noise.

Then, we discussed cell detection algorithm benchmarking methods. We argumented, that by not checking just one result set, but comparing two different result sets by computing their mutual distance, in a defined metric, may relieve human inspector of a significant amount of work. This motivated us to think about ways the distance function to a set of points can be computed.

After we analyzed commonly-used distance function methods and studied their differences, we formulated a novel method to compute the distance function. This one computes distance function to a set of points with the $O(n \log_2 n)$ complexity, like the state-of-the-art algorithm Fast Marching Method does, but using exact Euclidean values, not eikonal ones, as Fast Marching Method does. In addition to this, we proposed an enhancement for this method to work also in an environment containing walls and corners, relying on the usage of a line rasterization algorithm.

In various experiments in both unconstrained and maze-like environments, we have demonstrated the viability of the newly-formulated method. In practice, BDP algorithm was used to reconstruct cell lineage trees in a developmental biology application - for details, see [29].

Then, we described the motivation, specification, design and implementation of the software SliceViewer.

The experiments discussed in the end of the work served as validation for the aforementioned modifications and as a test of the SliceViewer software.

8 Resumé

Technológia dvojfotónového laserového mikroskopu nám umožňuje pozorovať embryo zebričky pruhovanej pár hodín po fertilizácii, a skúmať jeho správanie počas nasledujúcich zopár hodín vývoja, na úrovni samotných buniek. Dáta získané mikroskopom sú sekvenciou 3D snímkov v čase a obsahujú pre každý voxel hodnotu intenzity.

Zebrička pruhovaná je organizmus, s ktorým sa pracuje v mnohých súčasných projektoch skúmajúcich otázky vývojovej biológie - najmä vďaka jeho priehľadnosti, schopnosti absorbovať kontrastné látky, rezistencii voči mechanickému poškodeniu a vyššej odolnosti voči teplu zo svetla mikroskopu. Obraz je získavaný v dvoch kanáloch - na prvom sú bunkové membrány a na druhom bunkové jadrá.

V oblasti porozumenia tomuto biologickému obrazu sa riešia dva typy úloh - prvou z nich je kvantitatívne popísať rôzne biologické parametre organizmu a druhou je realizovať sledovanie buniek v čase, čiže rekonštrukcia rodokmeňov buniek.

V oboch úlohách je identifikácia buniek súčasťou postupov pre riešenie. Na identifikáciu buniek sa zvykne používať algoritmus FBLSCD - Flux-Based Level Set Center Detection, alebo novo navrhnutá metóda LSOpen - morfologické otvorenie na množine úrovní. Taktiež jestvujú mnohé iné metódy detekcie buniek. V tejto práci diskutujeme o možnosti porovnania výsledkov týchto metód, a taktiež pre túto úlohu navrhneme a implementujeme softvérové riešenie.

Ďalej uvedieme zoznam kapitol s krátkym popisom každej z nich.

Kapitola 3 uvádza problematiku v spracovaní obrazu zvanú detekcia objektov. Sú uvedené základné príznaky, spolu s náčrtom štandardne používaných metód na ich detekciu.

Podkapitola 3.1 popisuje zvyčajne používaný algoritmus na detekciu buniek zvaný FBLSCD. Tento algoritmus je prezentovaný ako evolúcia v škálovom priestore formulovaná ako parciálna diferenciálna rovnica. Táto spôsobuje, že izočiary intenzity v obraze sa zmršťujú v smere vnútornej normály. Ďalej je vysvetlené, ako je táto vlastnosť využitá na riešenie problematiky.

V podkapitole 3.2 diskutujeme o netriviálnom probléme nastavenia správnych parametrov pre FBLSCD. Identifikujeme štyri parametre: δ ako koeficient modulujúci silu advekčného člena rovnice, μ ako koeficient krivostnej difúzie v rovnici, R ako prahovú hodnotu, nad ktorou musí byť intenzita lokálnych maxím, aby boli uznané za maximá reprezentujúce bunkové jadrá a T, čas zastavenia evolúcie v škálovom priestore. Všetky tieto hodnoty závisia od dát a kvalita ich nastavenia determinuje kvalitu výsledkov algoritmu. V tejto práci prezentujeme plne automatický spôsob voľby parametra T pre dané δ , μ a R. V predchádzajúcich prácach navrhli autori pozorovať vývoj veličiny LMCE (Local Maxima Count Evolution), ale neboli spokojní s výsledkami pre reálne dáta. Stavali sme na ich skúsenostiach tým, že sme definovali T ako minimum kvantity, ktorú sme pomenovali LMCED (Local Maxima Count Evolution Decline). Navyše sme navrhli aplikovať štatistickú metódu najmenších štvorcov na zhladenie riešenia. Táto metóda minimalizuje varianciu snímok od snímky a dáva hodnoty T, ktoré sú plynulé naprieč videom.

V podkapitole 3.3 popisujeme kombináciu dvoch pohľadov na spracovanie obrazu, konkrétne prístup cez parciálne diferenciálne rovnice a morfologický prístup. Najprv zredukujeme rovnicu FBLSCD odňatím krivostného člena a normalizovaním advekcie: $\mu = 0$, $\delta = 1$. Táto operácia sa dá vnímať ako morfologická erózia v škále šedi a pomenovávame ju LSErode. Dá sa ukázať, že táto operácia vie rozlíšiť medzi veľkými objektami reprezentujúcimi bunkové jadrá a osamelými malými píkmi vysokej intenzity, reprezentujúcimi šum v obraze. Problém však nastáva, ak sa vyskytuje v obraze opačný typ šumu: malé píky nízkej intenzity priamo v útvaroch reprezentujúcich bunkové jadrá. V tom prípade tento šum nemizne, ale sa naopak rozpína, na úkor intenzity samotných jadier, čo dramaticky znižuje schopnosť algoritmu takéto jadro detekovať. Preto rozvíjame koncept LSErode a formulujeme opačnú operáciu, zvanú LSDilate, pomocou otočenia znamienka pri advekčnom člene. Táto metóda sa správa opačne ako LSErode - šumy s nízkou intenzitou miznú, ale šumy s vysokou intenzitou expandujú. Využívame preto štandardný koncept morfologickej školy spracovania obrazu pre kombináciu efektu týchto dvoch metód - krok LSErode, po ktorom nasleduje krok LSDilate je kompozitná operácia, ktorá

sa volá morfologické otvorenie - v našom prípade LSOpen. Ukážeme, že táto nová metóda je robustná voči obom spomínaným typom šumu.

V podkapitole 3.4 diskutujeme o metódach evaluácie výsledkov detekčných algoritmov. Začneme vysvetlením, že vzhľadom k veľkému množstvu výstupných dát je prakticky neúnosné, aby tieto dopodrobna evaluoval ľudský inšpektor. Oproti tomu uvádzame stratégiu, kde inšpektor porovnáva výsledky dvoch metód navzájom. Toto redukuje množstvo práce v prípade, ak dávajú metódy výsledky veľmi podobné a odlišujúce sa iba v malom percente rozdielov. Navyše, ak zvládneme porovnať výsledky dvoch metód aj sofistikovanejšie ako iba binárne, napríklad pomocou merania vzdialeností medzi jednotlivými identifikátormi, dosiahneme ďalšiu redukciu množstva práce.

V kapitole 4 popisujeme metódy pre výpočet funkcie vzdialenosti. Posledná podkapitola predošlej kapitoly nám slúži ako motivácia. Na začiatku tejto kapitoly definujeme problém a špecifikujeme požiadavky, ktoré kladieme na riešenie.

Podkapitola 4.1 analyzuje funkciu vzdialenosti bez obmedzení. Tu analyzujeme 5 často implementovaných a používaných metód: Brute-force (BF), Time relaxed Rouy Tourin (TRRT), Fast marching method (FMM), Fast sweeping method (FSM) a Vector Distance Transform (VDT). Okrem popísania týchto metód uvádzame aj pseudokódy pre každú z nich. Tieto metódy sa pokúsime klasifikovať podľa 1) definície vzdialenosti - táto môže byť eikonalová alebo Euklidovská a 2) poradia navštívenia pixelov - multi-visit, čelo vlny alebo zametacie. Naformulujeme taktiež novú metódu, ktorá vyplní prázdne miesto v klasifikácii dosiaľ jestvujúcich metód - táto nová metóda je postavená na Euklidovskej definícii vzdialenosti a má charakter čela vlny. Nazveme ju Dijstra-Pythagoras (DP). Táto metóda sa spolieha na spočítanie najprv odhadu vzdialenosti, podobným spôsobom ako Dijkstrova metóda v teórii grafov, a potom sa pri poslednej návšteve pixela ustáli na správnej Euklidovskej hodnote, pomocou Pytagorovej vety.

Ďalšia podkapitola, 4.2, formuluje zložitejší problém: počítanie funkcie vzdialenosti s obmedzeniami typu stena a roh, v čoho dôsledku je úloha ekvivalentná hľadaniu najkratšej cesty v bludisku. V tomto prípade uvidíme, že algoritmy fun-

gujúce na princípe čela vlny sú na riešenie vhodnejšie ako zametacie algoritmy. Na to, aby nový algoritmus DP v tomto prostredí fungoval, je nutné uviesť mierne modifikácie: konkrétne implementovať test viditeľnosti zdroja. Na vyriešenie tohto problému implementujeme algoritmus na rasterizáciu čiary. Túto novú metódu nazveme Bresenham-Dijkstra-Pythagoras.

Podkapitola 4.3 obsahuje mnoho experimentov z domény funkcii vzdialenosti. Meria presnosť, rýchlosť a kvalitu výsledkov jednotlivých metód na výpočet funkcie vzdialenosti. Tu uvidíme, že nová metóda BDP má zopár kvantitatívnych a kvalitatívnych výhod oproti zvyčajne používanej metóde FMM pre riešenie problému bludísk. Jedná sa o záver diskusie o funkcii vzdialenosti.

Kapitola 5 je skôr technickejšia. Softvér SliceViewer, riešenie slúžiace na semiautomatickú evaluáciu výsledkov detekcie buniek, je uvedený a popísaný. Softvér umožňuje používateľovi zobraziť 4D dáta na 2D obrazovke počítačového monitora pomocou dvoch bežcov. Ďalej vie vizualizovať pohyb v čase mapovaním troch snímkov (predošlý, aktuálny a nasledujúci) na tri farebné kanály (červený, zelený, modrý). No a napokon vie softvér vizualizovať relatívnu kvalitu algoritmov pomocou vzájonných korešpondencii a odlišností v bunkových identifikátoroch.

Napokon, kapitola 6 je diskusiou o výsledkoch experimentov. V prvom experimente tejto kapitoly uvidíme porovnanie správania FBLSCD a LSOpen metód. Tento experiment demonštruje, že kým väčšina výsledkov týchto dvoch algoritmov je zhodná, vyskytuje sa aj malé percento odlišností. Konkrétne, vo výsledkoch LSOpen chýba viacero centier, ktoré boli správne detekované pomocou FBLSCD, avšak výsledky LSOpen ukázali, že FBLSCD uvádza do sady výsledkov zopár identifikátorov centier, ktoré v skutočnosti v dátach nie sú. V druhom experimente porovnáme výsledky oproti biologickému zlatému štandardu. Pozorujeme, že obe metódy dávajú rozumné výsledky. V treťom experimente sa pokúsime empiricky overiť korektnosť numerickej diskretizácie a implementácie metódy LSOpen.

Bibliography

[1] R. Mikut, T. Dickmeis, W. Driever, P. Geurts, F.A. Hamprecht, B.X. Kausler, M.J. Ledesma-Carbayo, R. Mare, K. Mikula, P. Pantazis, O. Ronneberger, A. Santos, R. Stotzka, U. Strähle, N. Peyriéras *Automated processing of zebrafish imaging data: A survey*, Zebrafish, doi: 10.1089/zeb.2013.0886 (2013)

[2] P. Bourgine, R. Čunderlík, O. Drblíková, K. Mikula, N. Peyriéras, M. Remešíková, B. Rizzi, A. Sarti, *4D embryogenesis image analysis using PDE methods of image processing*, Kybernetika, Vol. 46, No. 2 (2010) pp. 226-259.

[3] K. Mikula, N. Peyriéras, M. Remešíkova, O. Stašova, *Segmentation of 3D cell membrane images by PDE methods and its applications*, Computers in Biology and Medicine, Vol. 41, No. 6 (2011) pp. 326-339

[4] P. Bourgine, P. Frolkovič, K. Mikula, N. Peyriéras, M. Remešíková, *Extraction of the intercellular skeleton from 2D microscope images of early embryogenesis*, in Lecture Notes in Computer Science 5567 (Proceeding of the 2nd International Conference on Scale Space and Variational Methods in Computer Vision, Voss, Norway, June 1-5,2009), Springer (2009) pp. 38-49

[5] O. Tassy, F. Daian, C. Hudson, V. Bertrand, P. Lemaire, *A quantitative approach to the study of cell shapes and interactions during early chordate embryogenesis*, Current biology vol. 16, 345 - 358, Elsevier (2006)

[6] Y. Bellaïche, F. Bosveld, F. Graner, K. Mikula, M. Remešíková, M. Smíšek, *New robust algorithm for tracking cells in videos of drosophila morphogenesis based on finding an ideal path in segmented spatio-temporal cellular structures*, Proceeding of the 33rd Annual International IEEE EMBS Confe-

rence, Boston Marriott Copley Place, Boston, MA, USA, August 30 - September 3, 2011, IEEE Press, 2011.

[7] K. Mikula, N. Peyriéras, M. Remešíková, M. Smíšek, *4D numerical schemes for cell image segmentation and tracking*, in Finite Volumes in Complex Applications VI, Problems & Perspectives, Eds. J. Fořt et al. (Proceedings of the Sixth International Conference on Finite Volumes in Complex Applications, Prague, June 6-10, 2011), Springer Verlag, 2011, pp. 693-702.

[8] K. Mikula, R. Špir, M. Smíšek, E. Faure, N. Peyriéras, *Nonlinear PDE based numerical methods for cell tracking in zebrafish embryogenesis*, Aplied Numerical Mathematics, accepted

[9] P. Frolkovič, K. Mikula, *Flux-based level set method: a finite volume method for evolving interfaces*, Applied Numerical Mathematics, Vol. 57, No. 4 (2007) pp. 436-454.

[10] P. Frolkovič, K. Mikula, N. Peyriéras, A. Sarti, *Counting number of cells and cell segmentation using advection-diffusion equations*, Kybernetika, Vol. 43, No. 6 (2007) pp. 817-829.

[11] K. Mikula, M. Smíšek, *Optimization of cell center detection method in morphogenesis image processing*, Proceedings of the Advances in Architectural, Civil and Environmental Engineering Conference, Bratislava, Slovakia, November 15, 2012.

[12] R. C. Gonzalez, R. E. Woods, *Digital image processing, 3rd ed.*, Prentice Hall, Upper Saddle River, NJ (2008)

[13] D. Lowe, *Distinctive image features from scale-invariant keypoints*, International Journal of Computer Vision, (2004)

[14] L. Alvarez, F. Guichard, P. Lions, J. Morel, *Axioms and fundamental equations of image processing*, Archive for Rational Mechanics and Analysis, Volume 123, Issue 3, pp 199-257 (1993)

[15] A. Sarti, R. Malladi, J. A. Sethian *Subjective surfaces: A method for completing missing noundaries*, Proceedings of the National Academy of Sciences of the United States of America 12 (97) (2000), 6258-6263

[16] Y. Chen, B.C. Vemuri, L. Wang, *Image denoising and segmentation via nonlinear diffusion*, Comput. Math. Appl., 39(5-6):131149, 2000

[17] M. Jones, J. A. Baerentzen, M. Šrámek, *3D distance fields: a survey of techniques and applications*, IEEE Transactions on Visualization and Computer Graphics (Volume: 12, Issue: 4), 2006

[18] R. Fabbri, L. Costa, J. Torelli, O. M. Bruno, *2D Euclidean distance transform: A comparative survey*, ACM Computing Survey, 2008

[19] K. Mikula, J. Urbán, *3D curve evolution algorithm with tangential redistribution for a fully automatic finding of an ideal camera path in virtual colonoscopy*, In Proceedings of the Third International Conference on Scale Space Methods and Variational Methods in Computer Vision, May 29th - June 2nd, 2011, Ein Gedi, Dead Sea, Israel, Lecture Notes in Computer Science Series, Springer, 2011

[20] E. Rouy, A. Tourin, *Viscosity solutions approach to shape-from-shading* , SIAM Journal on Numerical Analysis 29 No. 3 (1992), 867-884

[21] J. Sethian, *A fast marching level set method for monotonically advancing fronts*, Proceedings of the National Academy of Sciences, Vol. 93, pp. 1591-1595, (1996)

[22] J. N. Tsitsiklis, *Efficient algorithms for globally optimal trajectories*, IEEE Transactions on Automatic Control, Vol. 40, No. 9, September 1995, pp. 1528-1538

[23] T. Cormen, C. Leiserson, R. Rivest, C. Stein, *Introduction to algorithms (3rd ed.)*, MIT Press and McGraw-Hill. ISBN 0-262-03384-4. (2009)

[24] H. Zhao , *A fast sweeping method for eikonal equations*, Mathematics of Computation, Volume 74, Number 250, Pages 603-627

[25] P.-E. Danielsson, *Euclidean distance mapping*, Computer Graphics and Image Processing, 14:227-248, 1980

[26] J. Bresenham *Algorithm for computer control of a digital plotter* IBM Systems Journal, VOL. 4, NO. 1 ., (1965)

[27] M. W. Jones *The production of volume data from triangular meshes using voxelisation* Computer Graphics Forum, 15(5): 311-318, 1996

[28] M. Smíšek *Analysis of cell detection algorithms for microscopic image processing* Advances in Architectural,Civil and Environmental Engineering : 23rd Annual PhD student conference, Bratislava : ISBN 978-80-227-4102-6. p. 61-68 (2013)

[29] K. Mikula, R. Špir, M. Smíšek, E.Faure, N.Peyriéras *Nonlinear PDE based numerical methods for cell tracking in zebrafish embryogenesis* Aplied Numerical Mathematics, accepted

[30] Image taken from http://en.wikipedia.org/wiki/Fast_marching_method (link checked 18. March 2015)