Slovak University of Technology

Faculty of civil engineering

Mathematical and computational methods in embryogenesis image processing

Diploma thesis

SvF-5343-17034

Study programme:	Mathematical and computational modelling
Field of study:	9.1.9 Applied mathematics
Department:	Department of mathematics and descriptive geometry
Supervisor:	Prof. RNDr. Karol Mikula, DrSc.

May 20, 2011

Bc. Michal Smíšek

Declaration on word of honour:
I hereby declare that I have worked on this project on my own, using only the resources and literature listed in the bibliography and in correspondence with good scientific practices.

Acknowledgement:
I am thankful to my supervisor, Prof. Karol Mikula, whose encouragement, guidance and support from the initial to the final level helped me to gain insight into the topic. I also want to thank to Dr. Mariana Remešíkova, who supported us with valuable suggestions.
I appreciated collaboration with Dr. Franois Graner and Dr. Yohanns Bellaïche from Institut Curie, Paris, who provided us with the data and with many inspiring ideas.

Michal Smíšek
Bratislava May 20th, 2011

Abstract

In this diploma thesis we present a novel algorithm for tracking cells in time lapse confocal microscopy movie of a Drosophila epithelial tissue during pupal morphogenesis. We consider a 2D + time video as a 3D static image, where frames are stacked atop each other, and using a spatio-temporal segmentation algorithm we obtain information about spatio-temporal 3D tubes representing evolutions of cells. The main idea for tracking is the usage of two distance functions - first one from the root cells and second one from segmented boundaries. We track the cells backwards in time. The first distance function attracts the subsequently constructed cell trajectories to the root cells and the second one forces them to be close to centerlines of the segmented tubular structures. This makes our tracking algoritm robust against noise and missing spatio-temporal boundaries. In this thesis we also describe details of numerical discretizations of the corresponding non-linear partial differential equations and further implementation details. In the end we discuss our computational results.
Keywords nonlinear partial differential equations, finite volume method, morphogenesis, cell tracking

Abstrakt

V diplomovej práci popisujeme nový algoritmus na rekonštrukciu vývoja buniek vínnej mušky (drosophila) v procese morfogenézy z mikroskopického videa. Rekonštrukciou vývoja buniek v obrazových postupnostiach rozumieme identifikáciu jednotlivých buniek v obraze a nájdenie ich korešpondencií v obrazových sekvenciách. Náš prístup spočíva v uvažovaní 2D videa ako 3D statického obrazu, ktorý vznikne naskladaním jednotlivých snímok videa na seba. Následne s využitím časopriestorovej metódy subjektívnych povrchov získame informáciu o časopriestorových 3D útvaroch pripomínajúcich trubice, ktoré reprezentujú evolúcie buniek. Hlavnou myšlienkou je potom využitie dvoch vypočítaných funkcií vzdialenosti - jednej ako vzdialenosti od počiatočných buniek a druhej ako vzdialenosti od okrajov časopriestorových útvarov, pomocou ktorých získame trajektórie bunkových evolúcií. Prvá funkcia vzdialenosti prit'ahuje postupne konštruovanú trajektóriu smerom k počiatočnej bunke a druhá funkcia vzdialenosti ju udržuje v strede vysegmentovaného trubicovitého útvaru. To robí náš algoritmus robustným voči šumu a chýbajúcim hranám časopriestorových útvarov. V práci taktiež popisujeme detaily numerickej diskretizácie zodpovedajúcich parciálnych diferenciálnych rovníc a implementačné techniky. Na záver uvádzame diskusiu o výsledkoch experimentov.

Kl'účové slová nelineárne parciálne diferenciálne rovnice, metóda konečných objemov, morfogenéza, rekonštrukcia vývoja buniek

Contents

1 INTRODUCTION

Cell tracking means extracting spatio-temporal trajectories of cells in a developing organism and detecting moments of cell divisions. It is one of the most interesting topics in the modern biology - a reliable backward tracking method could answer some of the fundamental questions of developmental biology: global and local movement of cells, origin and formation of tissues and organs, cell division rate and localization, etc.

In this paper, we present a new method for tracking cells in 2D + time image sequences. We consider a time sequence of 2D images as a 3D image, where separate frames are stacked atop each other. We identify cell evolutions as a set of spatio-temporal tubes. We achieve this via spatio-temporal segmentation. Having these tubes segmented, tracking means, from a given point in tube interior, to find a trajectory - within this tube - to the cell identifier in the first video frame. In later sections, we will refer to these first-frame cells as to the "root cells". Finding a correct trajectory is achieved via computation and use of two constrained distance functions. The distance function from root cells forces trajectory to approach a root cell. The distance from segmented boundaries, keeps this trajectory centered.

We have at disposal a video of the mono-layered epithelium of the *Drosophila* pupa, which undergoes extensive proliferation and morphogenesis to form the *Drosophila* adult. Upon expression of E-Cadherin-GFP, which localises the adherence junctions, its development can be followed by confocal time-lapse microscopy[1]. The video was acquired with Nikon Ti spinning disk microscopes equipped with a HQ2 Ropper Camera. Video consists of 199 frames and has resolution 569 x 500 pixels and pixel intensity ranges from 0 to 255. In fig. 1 one can see the visualization of image data.

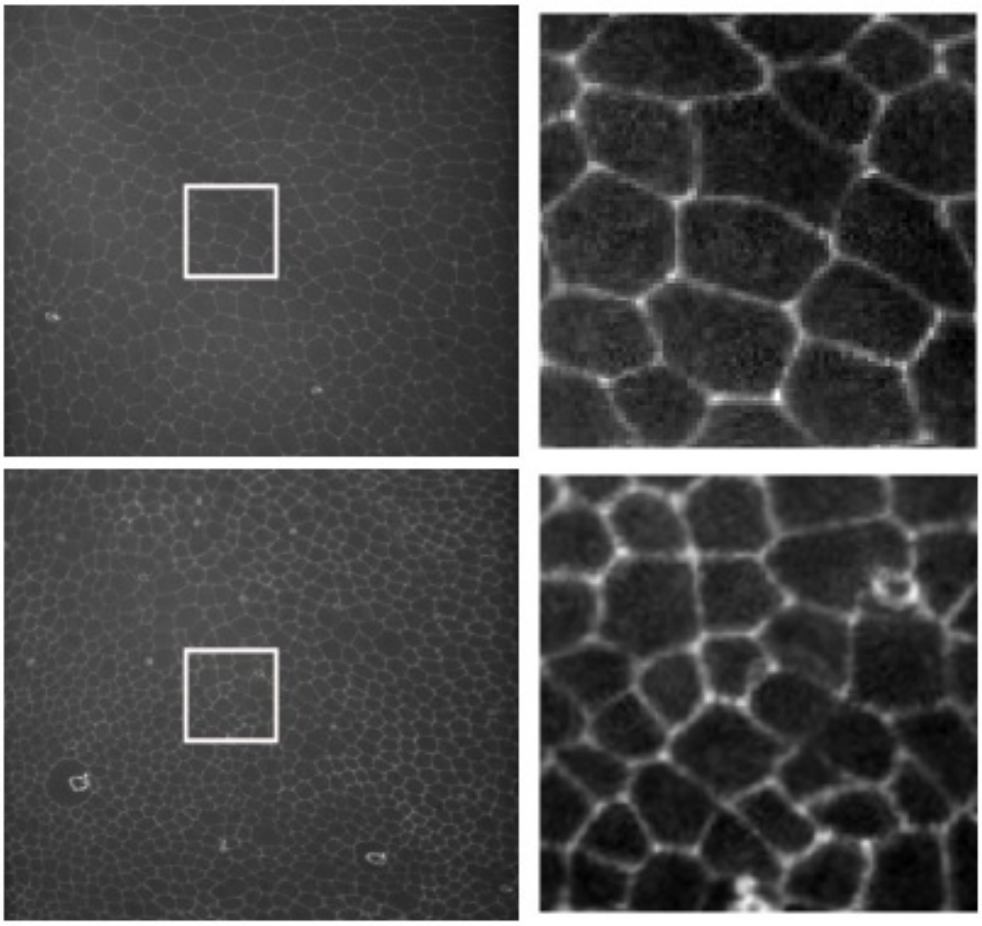

Figure 1: Data example. In the upper row the 40th frame, in the lower row the 140th frame. Left - whole frame, right - selected 100x100 pixel part of the frame under magnification. White box denotes the selected part.

2 ALGORITHM STEPS

Our algorithm consists of these consistent, but independent steps:

A. Cell identification

B. Spatio-temporal segmentation

C. Computing the distance from root cells

D. Computing the distance from the borders of spatio-temporal tubes

E. Extraction of cell trajectories

These steps are modular - one can choose different implementation for some of these steps (e.g. use cell identification/segmentation method), while still taking advantage of our algorithm's performance and robustness.

2.1 Cell identification

Cells in an image are objects with area larger than a certain threshold and smaller than some other threshold. Considering isophotes of the image intensity function, we see that if we approximate the contours of cells with a circle of radius r, this radius lies between some bounds d_1, d_2, $d_1 < r < d_2$. On the other hand, spurious noisy structures are represented by contours of radii significantly less than d_1, $0 < r \ll d_1$. A Level-Set Center Detection (LSCD) algorithm is designed with this property in mind, and we use it to identify cells in an image[2, 3]. In LSCD, we look for a numerical solution to the following equation:

$$u_t + \delta|\nabla u| - \mu|\nabla u|\nabla \cdot \left(\frac{\nabla u}{|\nabla u|}\right) = 0\,, \tag{1}$$

where the initial condition is an input image and boundary condition is zero Neumann. δ and μ are the coefficients of advection in the inward normal direction and the curvature regularization, respectively. Function u is defined as $u : R^2 \times [0, T] \to R$. The result of the algorithm is the set of maxima of u at the end of the evolution.

To illustrate properties of this algorithms, we created an artificial image containing nine cells and many noisy artifacts. We see that noisy structures disappear almost immediately - after the first step of LSCD they are gone. On the contrary, a cell remains in the image after many LSCD time steps. During the evolution, its contours are moved in the inward normal direction

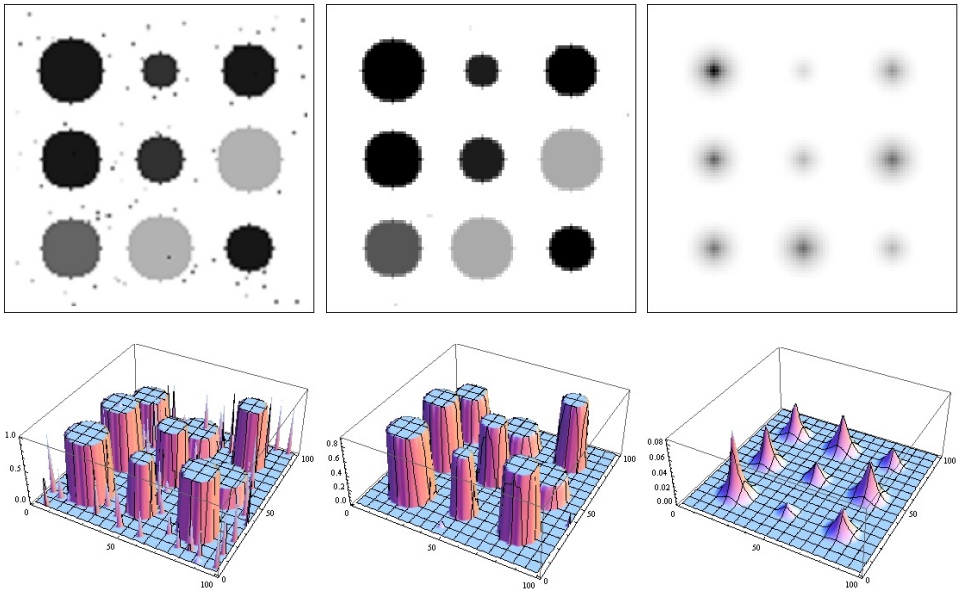

Figure 2: Illustration of the LSCD behaviour. Upper row - image itself, lower row - the same image seen as an intensity function graph. First image - artificial image of nine cells and 100 noisy structures. Cell is a sphere with diameter $\in [12, 24]$ and intensities $\in [0.3, 1.0]$, noisy artifacts are spheres of diameter 1 and intensity $\in [0.0, 1.0]$. Second image - evolution after one LSCD step. Third image - evolution after 50 steps. $\delta = 1.0$, $\mu = 0.000001$, step size $= 0.25$.

and regularized by curvature diffusion. The only heuristic here is the fact that contours of noisy structures have much smaller diameter than cells - cf. fig. 2

Depending on the signal-to-noise ratio of the image, one should consider filtering of the data in the pre-processing, to remove the image noise. Suitable filter is e.g. Geodesic Mean Curvature Flow (GMCF) smoothing algorithm[4, 5]. However, cell identification, as well as segmentation (see next step), both use a curvature regularization, so they implicitly contain smoothing and the filtering step is not always necessary.

The LSCD method is used separately for every 2D frame of the video. One can see the results of cell identification algorithm in fig. 3.

2.2 Spatio-temporal segmentation

For segmentation, we use the spatio-temporal Generalized Subjective Surface (GSUBSURF) algorithm[6, 7, 8, 3]. For this algorithm we need first to

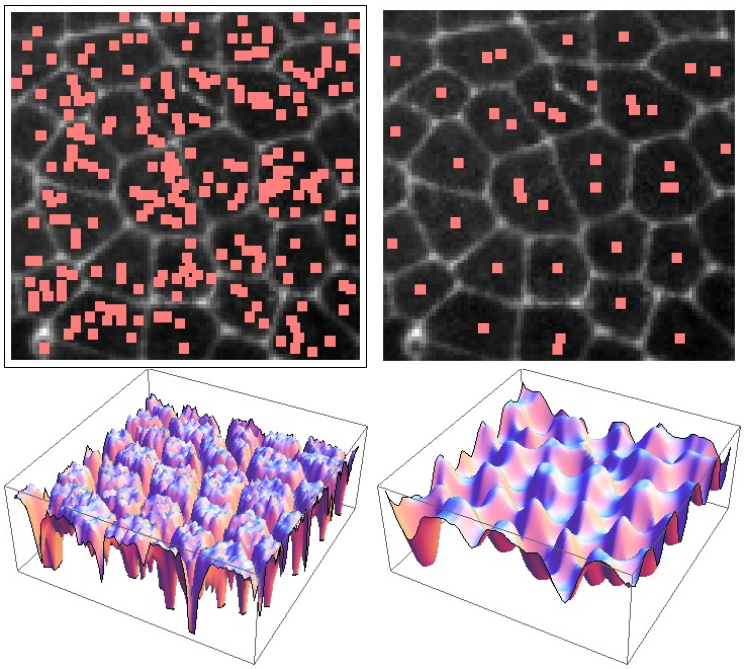

Figure 3: Cell identification visualization. In the upper row: local maxima of the original image (left) and of LSCD-evolved image (right) are visualized, viewed together with the original image as background. In the lower row: the intensity level functions of the original image (left) and the image after the LSCD evolution (right) are displayed.

construct an initial condition. This initial condition should approximate the spatio-temporal tubes according to the information we already have - the better it does, the less time steps of the algorithm we need to perform. We call this initial condition "initial segmentation". For each 2D frame, we create a disc of small radius around each cell identifier and we set pixels inside this disc to value 1, otherwise value stays at 0. GSUBSURF is a numerical solution to this equation:

$$u_t - w_a \nabla g \cdot \nabla u - w_c g |\nabla u| \nabla \cdot \left(\frac{\nabla u}{|\nabla u|} \right) = 0 \,, \tag{2}$$

where g is an edge detector function, w_a and w_c are advection and curvature parameters of the model. Here, u is defined as $u : R^3 \times [0, T] \to R$. It takes an initial segmentation profile and lets its isosurfaces evolve. We solve this equation in the whole spatio-temporal 3D area.

To illustrate the properties of this algorithm, we have created an artificial image and an initial segmentation. The artificial image is a square-shaped cell and initial segmentation is a small sphere created around the identifier of this cell, which is represented by its center. In this experiment, we also simulate the missing boundary problem by creating two holes in the artificial cell boundary - cf. fig 4.

During the evolution, shock profiles are created at the edges of cells [6, 7], and thus, considering pixels bounded by a specific isosurface, we obtain a border between a set of spatio-temporal 3D tubes and the rest of the image.

An important property of our spatio-temporal segmentation is that even if a cell identifier in a 2D frame is missing, we can still recover the shape of this cell by segmentation function evolution in the time direction. Furthermore, the spatio-temporal borders of cells are respected in GSUBSURF evolution, so we get separated 3D tubes. Of course, in real data, this separation may not be perfect, but this is solved in later steps of our approach. Both initial and final segmentation can be seen in fig. 5.

2.3 Distance from the root cells

To compute the distance from the root cells, we use the time relaxed eikonal equation, which looks as follows:

$$d_t + |\nabla d| = 1, d(x, t) = 0, x \in \Omega_0 \,, \tag{3}$$

where d, $d : R^3 \times [0, T] \to R$, as time increases, approximates the distance from the points where zero Dirichlet condition is prescribed. In this step of

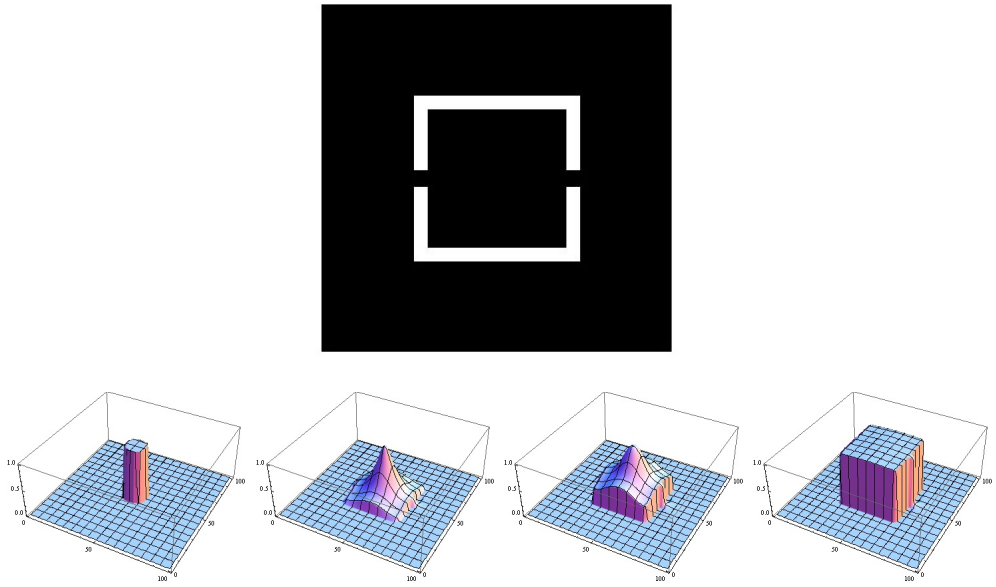

Figure 4: Illustration of the GSUBSURF behaviour. Upper image - artificial image of a square-shaped cell with a missing boundary. Size of the cell is 50x50 pixels. Lower row: first image - initial segmentation profile, created using the information about location of the cell center. Radius of sphere is 8 pixels. Second image - evolution after 600 GSUBSURF time steps. We can see shock profiles created along the cell boundary. Third image - evolution after 1000 time steps - considering the right contour of segmentation profile, here ≈ 0.2, we can get reliable information about the shape of the cell. Fourth image - evolution after 3000 steps, steady state solution. Choosing any contour gives the same answer.

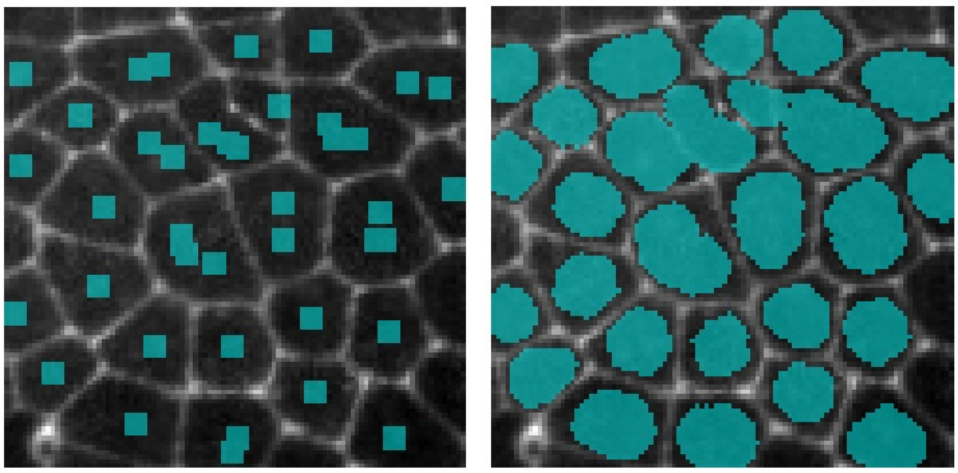

Figure 5: Spatio-temporal segmentation result visualization. Left, one 2D frame of initial segmentation - small seeds created around the cell identifiers. Right, one 2D slice of 3D segmentation result. It is a set of pixels for which the segmentation function has values greater than or equal to the prescribed value.

the algorithm, Ω_0 is given by the root cell identifiers. We will refer to this distance function as d_1. Technically, this equation is solved as in [9].

If we pick a point in our segmented set of spatio-temporal tubes, we want it to follow a path "down", i. e. in the direction of decrease of this distance function, until it reaches a root cell identifier - see fig. 6. This path is our first naive approach to the trajectory extraction. If the spatio-temporal tubes were perfectly isolated from each other, just following d_1 would be sufficient to get the approximate trajectories.

2.4 Distance from the borders of the spatio-temporal tubes

Following just d_1 often forces paths to follow borders of the spatio-temporal tubes, rather than their centers - this can be seen in fig. 6, especially in the top left image. This is not the most accurate path. Furthermore, if tubes are not perfectly isolated, paths can slip through holes in borders and give wrong tracking results - see fig. 6, top right image in the figure. The main idea of this step is to force the paths to follow the approximate centers of cells, while following d_1 distance down. Let us define a cell center as a point in cell with maximal distance from border of its spatio-temporal tube. To find this point, we again need to find a distance function.

The distance from the borders of the spatio-temporal tubes is also computed by the eikonal equation. This time, the points with zero Dirichlet condition are the border points of the spatio-temporal tubes. We denote this distance function as d_2.

An important property of this path modification is that if the spatio-temporal tubes meet, no slipping through this meeting point occurs - see fig. 6, in the lower row.

2.5 Extraction of cell trajectories

For a given point in a 3D spatio-temporal tube, we extract its cell trajectory by minimizing d_1 in a steepest-descent manner while maintaining d_2 maximized. In other words, trajectories go through the spatio-temporal 3D tube, backwards in time, to the root cell identifier, while staying in the cell center in each time frame.

Logically, a cell evolution can be represented as a binary tree. It is rooted at the root cell identifier and it branches out into two children each time the cell divides. As the video starts with many root cells existing already, we should rather talk about a binary forest - forest simply means a set of trees. Tree is constructed in such a way that if we choose a particular node as a representation of a cell, just by following line of its ancestors down to the root, we obtain a trajectory of this cell.

From the data structure point of view, the whole forest consists of nodes. These nodes, besides carrying their temporal and two spatial coordinates, also carry a reference to the parental node, left child node and right child node. As in most of the frames a cell doesn't divide, we use a standard of always following a left branch of a tree - a right child can be different from NULL if and only if a cell division occured at a given frame. A tree is created in such a way that for a given node we search for a list of predecessors, thus the tracking is computed backwards and the tree is constructed in a top-down manner.

In fig. 7 one can see a visualization of spatio-temporal 3D tubes, which were obtained using tracking results.

In order to visualize the tracking results themselves, we assign a color to each cell identified in the beginning of the video. This color is used to identify the cells corresponding to the evolution of the original cells. In fig. 12 and 13 one can see visualization of few frames of tracking results.

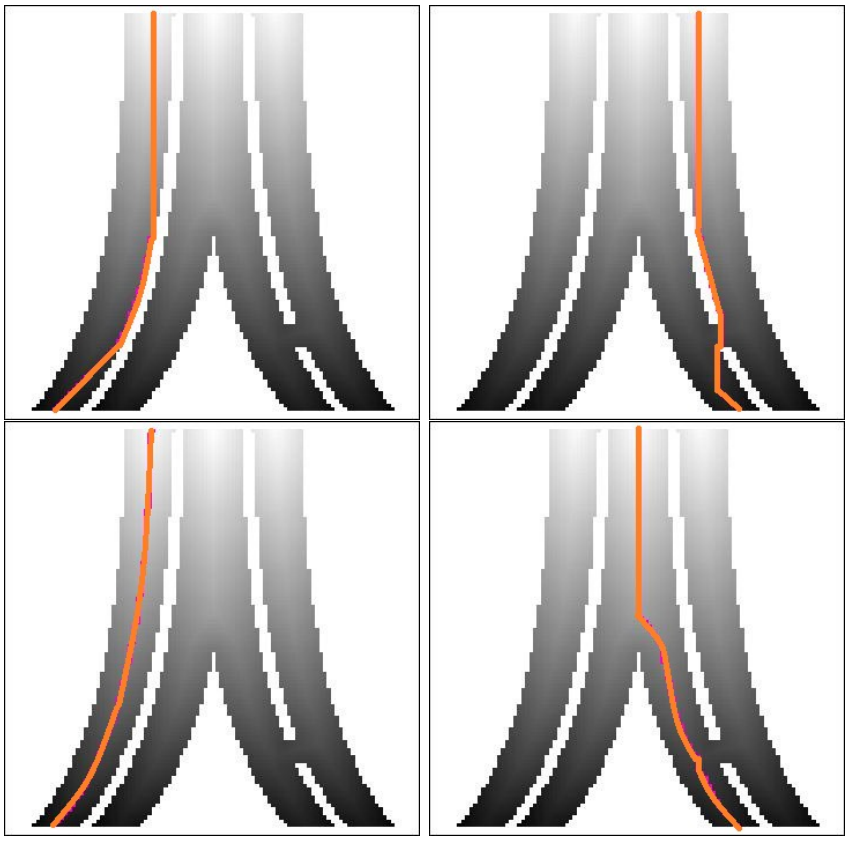

Figure 6: An illustration of the distance function properties. Image represents simulated evolution of the three artificial cells, where the cental one divides at about the half time of the evolution - the time axis goes down. There is an artificial missing boundary between the two cells in the lower right part of image. Upper row - trajectories calculated using only distance to the root cells. Tendency to follow the edges rather than the centers (left) and non-robustness against missing boundaries (right) is visible. Lower row - trajectories constructed using both distance function to the root cells and to the borders of tubes. Trajectories tend to follow centers of cells (left) and are robust against missing boudaries (right).

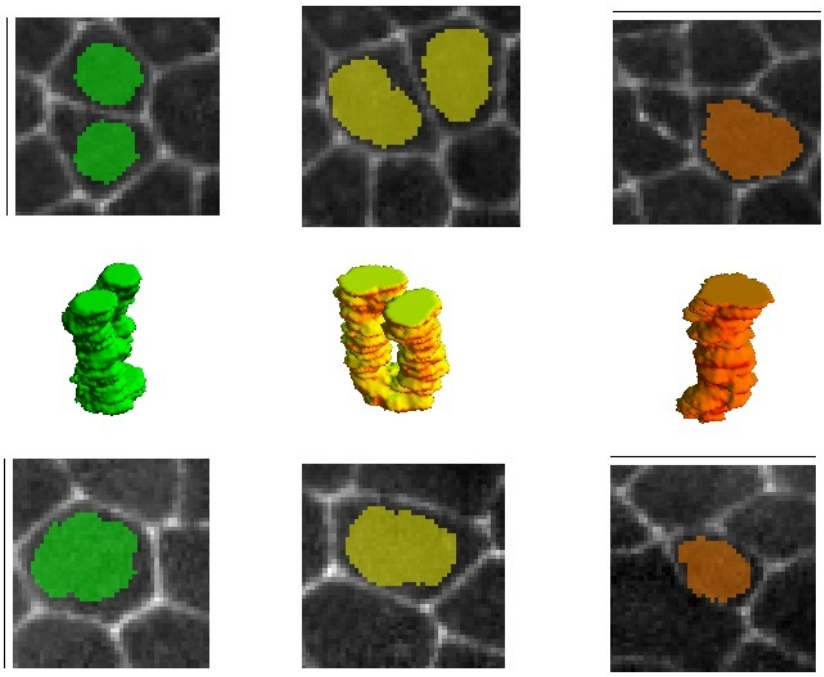

Figure 7: Visualization of the spatio-temporal 3D tubes. We use tracking results in order to obtain one specific tube from the set of tubes. Time axis points up. "Bifurcated tubes" represent evolution of cells which undergo cell divisions, whereas "simple tubes" represent cells that did not divide.

3 DISCRETIZATION

In this section, we are going to describe numerical schemes used to solve partial differential equations (1) and (2), describing LSCD and GSUBSURF models respectively, and the eikonal equation (3).

The LSCD and GSUBSURF are discretized and solved using the finite volume method. LSCD is calculated in a frame-by-frame manner, for each frame independently. GSUBSURF, on the contrary, is calculated considering frames stacked atop each other, as a 3D pile. Both these discretizations are thoroughly explained in [3]. The eikonal equation is solved by the Rouy-Tourin scheme [10] accompanied by the time relaxation as given in [9].

3.1 LSCD discretization

For LSCD therefore we use a two-dimensional finite volume discretization, where a natural choice for a control volume is the pixel. The equation (1) for LSCD is discretized in both temporal and spatial domain. For discretization in time we approximate all time derivatives with time differences using the semi-implicit approach - all the terms that are linear will be considered in new time $n+1$ and all those non-linear are taken at time n. This approach guarantees that our system stays linear - everything that is non-linear goes to the right-hand side. Let the evolution of the level set described by the LSCD equation (1) be computed within the interval $t \in \langle 0, T_c \rangle$. Let the number of discrete time steps taken be N_c - then the uniform time step length is $\tau_c = T_c/N_c$. The equation for LSCD discretized in time looks as follows:

$$\frac{u^{n+1} - u^n}{\tau_c} + \delta|\nabla u^n| - \mu|\nabla u^n|\nabla \cdot \left(\frac{\nabla u^{n+1}}{|\nabla u^n|}\right) = 0\,. \tag{4}$$

Then, we need to discretize this equation in space. An idea is to integrate it over the control volume (pixel) and replace spatial derivatives with differences. Let the pixel be denoted by V_{ij}, let its barycenter be c_{ij} and let the approximate value of the solution at time n be u_{ij}^n. The pixel area is denoted by $m(V_{ij})$. The equation integrated over the control volume reads as follows:

$$\int_{V_{ij}} \frac{u^{n+1} - u^n}{\tau_c} dx + \int_{V_{ij}} \delta|\nabla u^n| dx - \int_{V_{ij}} \mu|\nabla u^n|\nabla \cdot \left(\frac{\nabla u^{n+1}}{|\nabla u^n|}\right) dx = 0\,. \tag{5}$$

Further, in order to write it in the form of a linear system, we need to define few other variables. Let N_{ij}^{pq}, where $p, q \in \{-1, 0, 1\}$ and $|p| + |q| = 1$ be the set of four neighboring pixels of the pixel V_{ij}. Let σ_{ij}^{pq} denote the line

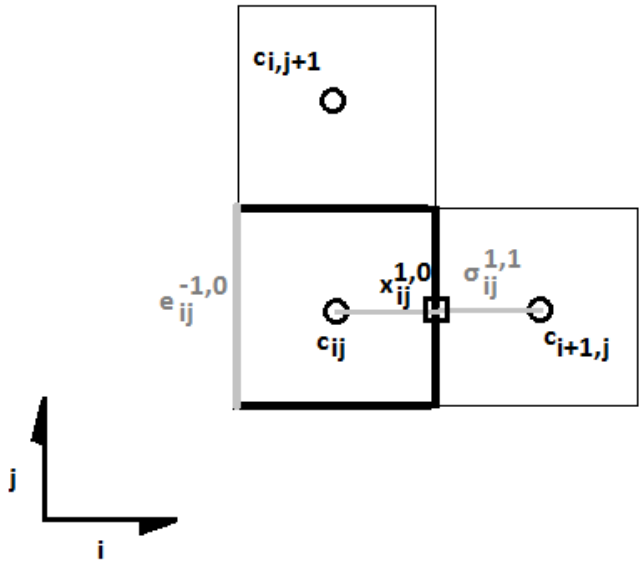

Figure 8: Notations for the 2D finite volume discretization.

between pixels V_{ij} and N_{ij}^{pq} and its length be $m(\sigma_{ij}^{pq})$. Let e_{ij}^{pq} be the pixel edge and its length be $m(e_{ij}^{pq})$. Let ν_{ij}^{pq} be the outward normal of e_{ij}^{pq}. Let x_{ij}^{pq} be the intersection of line σ_{ij}^{pq} and pixel edge e_{ij}^{pq}. Figure 8 presents these definitions in a visual form.

Further, define approximation of gradient modulus on edge, averaged gradient modulus in pixel, velocity vector field and discrete fluxes through the pixel boundary:

$$\begin{aligned} Q_{ij}^{pq,n} &:= |\nabla u_{ij}^{pq,n}|\,, \\ \bar{Q}_{ij}^{n} &:= \frac{1}{4}\sum_{N_{ij}} |\nabla u_{ij}^{pq,n}|\,, \\ v^n &:= \delta \frac{\nabla u^n}{|\nabla u^n|}\,, \\ v_{ij}^{pq,n} &:= \int_{e_{ij}^{pq}} v^n \cdot \nu_{ij}^{pq} d\gamma \approx m(e_{ij}^{pq})\delta \frac{u_{i+p,j+q}^n - u_{ij}^n}{Q_{ij}^{pq,n} m(\sigma_{ij}^{pq})}\,, \end{aligned}$$

where $\nabla u_{ij}^{pq,n}$ is an approximation of the solution gradient in x_{ij}^{pq}, see [3].

We also use the following identity:

$$v^n \cdot \nabla u^n = \nabla \cdot (v^n u^n) - u^n \nabla \cdot v^n\,, \tag{6}$$

and the upwind principle - that means partitioning fluxes into "inflows" ($v_{ij}^{pq,n} < 0$) and "outflows" ($v_{ij}^{pq,n} > 0$). We define two sets of indices:

$N_{ij}^{out} = \{(p,q) \in N_{ij}, v_{ij}^{pq,n} > 0\}$, $N_{ij}^{in} = \{(p,q) \in N_{ij}, v_{ij}^{pq,n} \leq 0\}$. Using divergence theorem and upwind principle we obtain for the first term on the right-hand side of (6):

$$\int_{V_{ij}} \nabla \cdot (v^n u^n) dx \approx \sum_{N_{ij}^{out}} u_{ij}^n v_{ij}^{pq,n} + \sum_{N_{ij}^{in}} u_{i+p,j+q}^n v_{ij}^{pq,n} \,. \tag{7}$$

Using the definitions above, one can discretize all terms from the equation (5) as follows:

$$\begin{aligned}
\int_{V_{ij}} \frac{u^{n+1} - u^n}{\tau_c} dx &\approx m(V_{ij}) \frac{u_{ij}^{n+1} - u_{ij}^n}{\tau_c} \,, \\
\int_{V_{ij}} \delta |\nabla u^n| dx &= \int_{V_{ij}} \delta \frac{\nabla u^n}{|\nabla u^n|} \cdot \nabla u^n dx = \int_{V_{ij}} v^n \cdot \nabla u^n dx \approx \\
&\approx \sum_{N_{ij}^{out}} u_{ij}^n v_{ij}^{pq,n} + \sum_{N_{ij}^{in}} u_{i+p,j+q}^n v_{ij}^{pq,n} - u_{ij}^n \sum_{N_{ij}} v_{ij}^{pq,n} = \\
&= \sum_{N_{ij}^{in}} (u_{i+p,j+q}^n - u_{ij}^n) v_{ij}^{pq,n} \,, \\
\int_{V_{ij}} \mu |\nabla u^n| \nabla \cdot \left(\frac{\nabla u^{n+1}}{|\nabla u^n|} \right) dx &\approx \mu \bar{Q}_{ij}^n \sum_{N_{ij}} \int_{e_{ij}^{pq}} \frac{\nabla u^{n+1}}{|\nabla u^n|} \nu_{ij}^{pq} d\gamma \approx \\
&\approx \mu \bar{Q}_{ij}^n \sum_{N_{ij}} m(e_{ij}^{pq}) \frac{u_{i+p,j+q}^{n+1} - u_{ij}^{n+1}}{Q_{ij}^{pq,n} m(\sigma_{ij}^{pq})} \,.
\end{aligned}$$

The fully discrete form of equation (5) looks as follows:

$$\begin{aligned}
m(V_{ij}) \frac{u_{ij}^{n+1} - u_{ij}^n}{\tau_c} + \sum_{N_{ij}^{in}} (u_{i+p,j+q}^n - u_{ij}^n) v_{ij}^{pq,n} - \\
- \mu \bar{Q}_{ij}^n \sum_{N_{ij}} m(e_{ij}^{pq}) \frac{u_{i+p,j+q}^{n+1} - u_{ij}^{n+1}}{Q_{ij}^{pq,n} m(\sigma_{ij}^{pq})} = 0 \,.
\end{aligned}$$

3.2 GSUBSURF discretization

In GSUBSURF, we are solving a three-dimensional problem by the finite volume method. A natural choice for a control volume is thus the voxel (three-dimensional pixel). The equation (2) for GSUBSURF is discretized in both temporal and spatial domain.

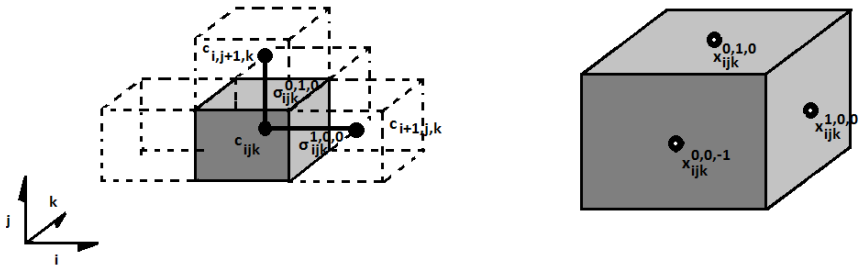

Figure 9: Notations for the 3D finite volume discretization.

For discretization in time, we approximate all time derivatives by using time differences and using the semi-implicit approach again. Let the evolution of level set, described by the equation (2), be computed within the interval $t \in \langle 0, T_s \rangle$. Let the number of discrete time steps taken be N_s - then the uniform time step length is $\tau_s = T_s/N_s$. The equation for GSUBSURF discretized in time looks as follows:

$$\frac{u^{n+1} - u^n}{\tau_s} - w_a \nabla g \cdot \nabla u^n - w_c g |\nabla u^n| \nabla \cdot \left(\frac{\nabla u^{n+1}}{|\nabla u^n|} \right) = 0 \,. \tag{8}$$

In order to discretize the system in space, same idea as in LSCD discretization applies here as well - integrate the equation over the control volume (here represented by the voxel) and replace spatial derivatives with differences. Let the voxel be denoted by V_{ijk}, let its barycenter be c_{ijk} and let the approximate value of the voxel at time n be u^n_{ijk}. The voxel volume is denoted by $m(V_{ijk})$. An equation integrated over the voxel reads as follows:

$$\int_{V_{ijk}} \frac{u^{n+1} - u^n}{\tau_s} dx - \int_{V_{ijk}} w_a \nabla g \cdot \nabla u^n dx - \int_{V_{ijk}} w_c g |\nabla u^n| \nabla \cdot \left(\frac{\nabla u^{n+1}}{|\nabla u^n|} \right) dx = 0 \,. \tag{9}$$

Further, we need to define the same variables as we did previously for the LSCD, but now from a three-dimensional point of view. Let N^{pqr}_{ijk}, where $p, q, r \in \{-1, 0, 1\}$ and $|p| + |q| + |r| = 1$ be the six voxels neighboring to the voxel V_{ijk}. Let σ^{pqr}_{ijk} denote the line between voxels V_{ijk} and N^{pqr}_{ijk} and its length be $m(\sigma^{pqr}_{ijk})$. Let e^{pqr}_{ijk} be the voxel face and its area be $m(e^{pqr}_{ijk})$. Let ν^{pqr}_{ijk} be the outward normal of face e^{pqr}_{ijk}. Let x^{pqr}_{ijk} be the intersection of line σ^{pqr}_{ijk} and voxel face e^{pqr}_{ijk}. Figure 9 presents the definitions in a visual form.

Further we define approximation of gradient modulus on faces, averaged gradient modulus in voxel, velocity field and discrete fluxes on the voxel faces

for the 3D scheme:

$$\begin{aligned} Q_{ijk}^{pqr,n} &:= |\nabla u_{ijk}^{pqr,n}|\,, \\ \bar{Q}_{ijk}^{n} &:= \frac{1}{6}\sum_{N_{ijk}} |\nabla u_{ijk}^{pqr,n}|\,, \\ v &:= -w_a \nabla g\,, \\ v_{ijk}^{pqr} &:= m(e_{ijk}^{pqr})\left(-w_a \frac{g_{i+p,j+q,k+r} - g_{ijk}}{m(\sigma_{ijk}^{pqr})}\right)\,, \end{aligned}$$

where, again, $\nabla u_{ijk}^{pqr,n}$ is an approximation of the solution gradient in x_{ijk}^{pqr} - see [3].

We use the same identity as we did in LSCD discretization:

$$v \cdot \nabla u^n = \nabla \cdot (v u^n) - u^n \nabla \cdot v\,, \tag{10}$$

and the upwind principle for 3D scheme - we partition fluxes into "inflows" ($v_{ijk}^{pqr} < 0$) and "outflows" ($v_{ijk}^{pqr} > 0$). We define two sets of indices: $N_{ijk}^{out} = \{(p,q,r) \in N_{ijk}, v_{ijk}^{pqr} > 0\}$, $N_{ijk}^{in} = \{(p,q,r) \in N_{ijk}, v_{ijk}^{pqr} \leq 0\}$. For the first right-hand term in (10) we obtain:

$$\int_{V_{ijk}} \nabla \cdot (v u^n) dx \approx \sum_{N_{ijk}^{out}} u_{ijk}^n v_{ijk}^{pqr} + \sum_{N_{ijk}^{in}} u_{i+p,j+q,k+r}^n v_{ijk}^{pqr}\,, \tag{11}$$

and further terms of (9) are approximated as follows:

$$\begin{aligned} \int_{V_{ijk}} \frac{u^{n+1} - u^n}{\tau_s} dx &\approx m(V_{ijk}) \frac{u_{ijk}^{n+1} - u_{ijk}^n}{\tau_s}\,, \\ \int_{V_{ijk}} -w_a \nabla g \cdot \nabla u^n dx &= \int_{V_{ijk}} v \cdot \nabla u^n dx \approx \\ \approx \sum_{N_{ijk}^{out}} u_{ijk}^n v_{ijk}^{pqr} &+ \sum_{N_{ijk}^{in}} u_{i+p,j+q,k+r}^n v_{ijk}^{pqr} - u_{ijk}^n \sum_{N_{ijk}} v_{ijk}^{pqr} = \\ &= \sum_{N_{ijk}^{in}} (u_{i+p,j+q,k+r}^n - u_{ijk}^n) v_{ijk}^{pqr}\,, \\ \int_{V_{ijk}} w_c g |\nabla u^n| \nabla \cdot \left(\frac{\nabla u^{n+1}}{|\nabla u^n|}\right) dx &\approx w_c g_{ijk} \bar{Q}_{ijk}^n \sum_{N_{ijk}} m(e_{ijk}^{pqr}) \frac{u_{i+p,j+q,k+r}^{n+1} - u_{ijk}^{n+1}}{Q_{ijk}^{pqr,n} m(\sigma_{ijk}^{pqr})}\,. \end{aligned}$$

The fully discretized GSUBSURF equation looks as follows:

$$m(V_{ijk}) \frac{u_{ijk}^{n+1} - u_{ijk}^n}{\tau_s} - \sum_{N_{ijk}^{in}} (u_{i+p,j+q,k+r}^n - u_{ijk}^n) v_{ijk}^{pqr}$$

$$- \quad w_c g_{ijk} \bar{Q}^n_{ijk} \sum_{N_{ijk}} m(e^{pqr}_{ijk}) \frac{u^{n+1}_{i+p,j+q,k+r} - u^{n+1}_{ijk}}{Q^{pqr,n}_{ijk} m(\sigma^{pqr}_{ijk})} = 0\,.$$

3.3 Computing distance functions

Finally we need to propose a method for solving the time-relaxed eikonal equation (3). For this step, we adopt an explicit scheme for temporal discretization and the Rouy-Tourin scheme, cf. [9], to discretize the equation 3 in space.

The distance function from the boundaries of spatio-temporal 3D tubes is computed frame by frame - this leads to the two-dimensional problem. The space grid elements correspond to the pixels of the image. Let the elements be squares, denoted by V_{ij}, where the length of the pixel side is h_D. For each element V_{ij}, let the approximate value of the solution d at time step n in the center of V_{ij} be d^n_{ij}. If we define M^{pq}_{ij}, $p, q \in \{-1, 0, 1\}, |p| + |q| = 1$ as

$$M^{pq}_{ij} = \left(\min(d^n_{i+p,j+q} - d^n_{ij}, 0)\right)^2, \tag{12}$$

the Rouy-Tourin scheme for the equation (3) in 2D then reads as follows:

$$d^{n+1}_{ij} = d^n_{ij} + \tau_D - \frac{\tau_D}{h_D} \sqrt{\max\left(M^{-1,0}_{ij}, M^{1,0}_{ij}\right) + \max\left(M^{0,-1}_{ij}, M^{0,1}_{ij}\right)}\,. \tag{13}$$

where τ_D is the time step size. This scheme is stable for $\tau_D \leq h_D/2$.

On the contrary, the distance function from the root cell identifiers is computed in the spatio-temporal domain - this leads to the three-dimensional problem. Here, the space grid elements correspond to the voxels. Let the elements be cubes, denoted by V_{ijk}, where the length of the side is h_D. For each element V_{ijk}, let the approximate value of the solution d at time step n in the center the center of V_{ijk} be d^n_{ijk}. Let us define M^{pqr}_{ijk}, $p, q, r \in \{-1, 0, 1\}, |p| + |q| + |r| = 1$ as

$$M^{pqr}_{ijk} = \left(\min(d^n_{i+p,j+q,k+r} - d^n_{ijk}, 0)\right)^2. \tag{14}$$

The Rouy-Tourin scheme for solving the equation (3) in the three-dimensional space then reads as follows:

$$d^{n+1}_{ijk} = d^n_{ijk} + \tau_D - \frac{\tau_D}{h_D} \sqrt{\max\left(M^{-1,0,0}_{ijk}, M^{1,0,0}_{ijk}\right) + \max\left(M^{0,-1,0}_{ijk}, M^{0,1,0}_{ijk}\right) + \max\left(M^{0,0,-1}_{ijk}, M^{0,0,1}_{ijk}\right)},$$

where τ_D is the time step size. Scheme is stable for $\tau_D \leq h_D/2$.

4 IMPLEMENTATION

For the implementation we chose the programming language C++. We have considered using Java or C#, but we decided to use a lower-level language instead to increase speed of computations. Furthermore, we wanted to avoid the use of automatic garbage collector of Java, as the algorithm is very memory-sensitive and we want to have a full control over the memory management. We have also considered using plain C, but we decided not to, taking an advantage of the fact that C++ classes can have methods and overload operators. To give an example of this, compare the two codes. They both add two images and normalize them according to the maximum of the third one. Notice that the latter example is easier to read, write and debug.

Example 1 - Plain C implementation:

```
struct Img{
...
};

Img img1;
Img img2;
Img img3;

...

divide(&(add(&img1, &img2),max(&img3));
```

Example 2 - C++ implementation using member methods and overloaded operators:

```
class Img{
...
};

Img img1;
Img img2;
Img img3;

...

(img1 + img2) / img3.max();
```

When calculating coefficients of numerical schemes, things get even more tangled, with forms containing several levels of brackets, additions, subtractions, multiplications and divisions. We want our code to be as readable as possible, for the purpose of debugging and later modifications, so C++ is our language of choice. It is worth noting here that MPI (Message Passing Interface), which we plan to use later to parallelize the algorithms, only works with plain C and structs, so parallelization of our code adds some work in recasting C++ code into plain C. We implemented our project in the MS Visual C++ 2005 environment.

Throughout the code implementation, we adapt a Java-style convention of naming classes by a capital first letter and their instances, variables and methods by a small letter.

To view the results of our algorithms, we have implemented some programs which we call viewers. For this purpose, we chose Mathematica as a well-suited environment - it contains neat plotting and displaying libraries which are easy to work with.

4.1 Data structures

`class Img` - represents a regular 2D image. `isize` and `jsize` are width and height of the image, `f` is an array of floats representing intensities of image pixels, assumed to have allocated `isize*jsize*sizeof(float)` bytes. Pixel [i,j] is accessed via accessing `f[i*jsize+j]`.

`class Vid` - represents a sequence of 2D images. `tsize` is the number of images in the sequence, `isize` and `jsize` are width and height of frames and `f` is an array of `Img` instances. Pixel [i,j] at time t is accessed via accessing `f[t]->f[i*jsize+j]`.

`class Img3D` - represents a 3D image. `tsize, isize` and `jsize` are the size in temporal direction, width and height, respectively. `f` is an array of floats representing intensities of image voxels. `f` is assumed to have allocated `tsize*isize*jsize*sizeof(float)` bytes. Voxel [t,i,j] is accessed via accessing `f[t*isize*jsize+i*jsize+j]`. `Img3D` and `Vid` have a remarkable property: their encoding, i.e. the way they are saved into a file, are identical. That enables us to convert one of them into the other one via `saveTo/loadFrom` operations:

```
Vid inseg(ts,is,js);
inseg.setToInitSegm( ... );
inseg.saveTo("file.txt");
```

```
Img3D is3D(ts,is,js);
is3D.loadFrom("file.txt");
```

We use this trick in our code.

`class TrajList` - represents trajectories of cells in a 3D image. Its elements are the binary tree nodes - they have a reference to a parental node and references to two child nodes, called left and right child. Further, a node contains an information about its position in the image in the form [t,i,j].

`TrajList` is the list of these nodes. `length` is the number of nodal structures. It contains three positional lists `tlist`, `ilist` and `jlist`. The position of the n-th in the image is `[tlist[n], ilist[n], jlist[n]]`. Further, it contains lists `plist`, `lclist`, `rclist` - these are the references to parental node, left child and right child, respectively.

From a given node, we reach a node representing root cell by following the line of its ancestors - i.e. for n-th node we call `plist[plist[ ... [n] ...]]`. Similarly, from a given root cell node, we can assign any "color" this node and recursively assign this color its left children subtree and right children subtree. If we do this using different color for each root cell node, we obtain a neat visualization of tracking results.

`class COHList` - represents a set of points in a 2D image. `length` is the number of points in the set, `ilist` and `jlist` are lists of two spatial coordinates denoting a specific point. `COHList` only supports adding of points, no removing was needed to be implemented throughout the whole application.

`class COHList3D` - represents a set of points in a 3D image. `length` is the number of points in the set, `tlist`, `ilist` and `jlist` are lists of a temporal and two spatial coordinates denoting a specific point. `COHList3D` only supports adding of points.

4.2 File encoding (save to/load from HDD)

All of the six main data structures use a common pattern of saving their state to harddisk, loading from it and assigning themselves to some other instance of a given data structure. We only describe implementation for `Img`, as the implementations for other data structures are very similar.

File format for `Img` is following: line 1 - `isize`, line 2 - `jsize`, next `isize * jsize` lines - intensities for pixels, pixel [i,j] referring to `f[i*jsize+j]`.

This encoding is not optimal from the file size minimization point of view, however it makes it easy to manually modify a data structure by editing file in a plain-text editor. This human-readable file format also comes handy for debugging purposes.

`Img::saveTo(char * filename)` - saves an image to a "filename" file

`Img::loadFrom(char * filename)` - loads an image from a "filename" file

`Img::setTo(Img & source)` - sets an image to reflect the state of "source" image. As it doesn't allocate memory anew, it assumes that the images have identic sizes and `this->f` to have `isize*jsize*sizeof(float)` bytes allocated.

4.3 Functions

For `Img` and `Img3D`, we often need to perform one of four elementary operations (addition, subtraction, multiplication and division) of either two instances of a given class, or one instance of a given class and a real number. To handle these operations comfortably, we take an advantage of C++ operator overloading. Motivation for this step was presented at the beginning of this section, in the discussion about choice of a programming language.

Thus, `Img` has eight overloaded operator methods. `Img3D` is very similar.

```
Img operator+(Img & first, float scalar);
Img operator+(Img & first, Img & second);
Img operator-(Img & first, float scalar);
Img operator-(Img & first, Img & second);
Img operator*(Img & first, float scalar);
Img operator*(Img & first, Img & second);
Img operator/(Img & first, float scalar);
Img operator/(Img & first, Img & second);
```

Further, each class has its own set of specialized methods.

```
class Img {
        ...

        Img takePart(int, int, int, int);
```

```
        Img ** gradientsOfVoxelFaces();
        Img ** lenOfGrads(float);
        Img avgGradModulus();

        // filtration
        Img gaussStep(float stepsize);
        Img gaussConv(float stepsize, int timesteps);
        Img gFunc(float K);
        Img gmcf(int timesteps);

        // cell detection
        Img lscd(int timesteps);
        COHList getLocalMaximaList();

        // distance function
        Img rtdist(int, Img&, Img&);
        void setToTrousers();
        COHList downwardPath(int,int);

        // segmentation
        void setToInitSegm(COHList& , float);
        Img subsurf(int , Img&);
        Img gsubsurf(int , Img&, float, float, float);
        Img trimSegmentation(float);
}
```

Main purpose of the methods `gaussStep`, `gmcf`, `lscd`, `subsurf`, `gsubsurf` is to compute coefficients from discretized form of given equations and to call a linear system solver SOR function - see next subsection.

`class Img3D` has similar functions, except for the cell detection part.

`class Vid` has very few functions - it contains an array of `Img` instances, so performing an operation is usually implemented using a simple `for (i=0; i<vid.tsize; i++)` cycle.

`class COHList` has a function `add(int ptI, ptJ)`, which adds a point [i,j] to its list. We obtain a list of points via the function

```
COHList Img::getLocalMaximaList();
```

and we use it to construct an initial segmentation profile via the function

```
void Img::setToInitSegm(COHList& cohlist, float radius)
```

`class COHList3D` works similarly to `class COHList`. It has a special function of returning a `COHList`, being a subset of the points in a given frame t:

```
COHList COHList3D::getPtsAtTime(int t);
```

```
class TrajList{
        ...

        int loadFrom(char *);
        void saveTo(char *);
        void setTo(TrajList&);

        void addNode(int,int,int,int,int,int);
        void setNodeTo(int,int,int,int);

        void setChildrenColor(int,float,Vid*);
        Vid getColorVideo(int,int,int,int,float);
        int getDepth(int);
        int getMaxJ(int,int);
};
```

To obtain a `TrajList` from what we have computed, we use a procedure called `twoStepPath`. From a given point, it creates a chain of nodes down to the root cell, using two distance functions. It is an algorithm consisting of two alternating steps. First step is, from the actual point, to find a neighboring grid point (in 3D, each voxel, except for boundary voxels, has 26 neighboring voxels) which minimizes the distance function from the root cell identifiers. Second step is to maximize the distance function from boundaries of spatio-temporal tubes within a given frame - for a fixed t in 3D image. First step guarantees that the path goes back in temporal direction. Second step guarantees that it remains centered within the cell. An algorithm stops if there is no neighboring voxel with distance function smaller than the actual point - this yields a root cell identifier.

As it goes down the path, it checks if a new point is already a member of some other path or is not yet used. If it was not used yet, we simply claim this point to be a parent of the older one, and the older one to be its left child. Since new node is parent and old one is child, we see that the binary tree is created in a top-down manner. If there is an existing path going through this point, we connect these two points by claiming this point to be a parent

of the older one and the older one to be its right child, because since it is a part of the path, it already has its left child assigned. Thus, in this case, we obtain cell division identification.

The algorithm works as follows:

```
void twoStepPath(TrajList * ret,
         int tStart, int iStart,int jStart,
         Vid& dfr, Vid& dfe, int * mem) {

         // ret - return value of the function
         // tStart,iStart,jStart - starting point coords
         // dfr - Distance From Root cell identifiers
         // dfe - Distance From Edges of spatio-temporal tubes
         // mem - memory of references to trajectory nodes

         // ... initialize

         do {
                  if (mem[tAct,iAct,jAct] != -1) {
                           // set ''actual'' to right child
                           // of mem[tAct,iAct,jAct]
                           return;
                  } else {
                           // set mem[tAct,iAct,jAct] to actual
                           // ... minimize dfr
                           if (tMin < tPrev) {
                                    // ... minimize dfe
                  }
         } while (tMin!=tAct || iMin!=iAct || jMin!=jAct);

         return;
}
```

To obtain a complete `TrajList`, we run this for each cell identifier's coordinates as `tStart`, `iStart` and `jStart`, using the same `mem`.

4.4 Linear-system solver: SOR

To solve LSCD and GSUBSURF in the fully discrete form, we use SOR (Successive Over-Relaxation) algorithm. It is an iterative solver, which is guaranteed to converge for M-matrices. It is implemented in 2D and 3D, respectively:

```
Img solnOfSystem(
        Img& a, Img& aip,Img& ajp,
        Img& aim,Img& ajm, Img& b,
        float omega, float toll, int type
);

Img3D solnOfSystem3D(
        Img3D& a, Img3D& atp, Img3D& aip,Img3D& ajp,
        Img3D& atm, Img3D& aim, Img3D& ajm, Img3D& b,
        float omega, float toll, int type
);
```

Here `a` is a list of diagonal coefficients, next 4 (resp. 6) instances are lists of coefficients for neighbors, `b` is a right-hand side of the system. `omega` is a relaxation parameter, `toll` is the precision threshold and `type` is a boundary condition type (0=zero Neumann, 1=zero Dirichlet).

The main cycle of the function updates the initial value until norm of an update is less than `toll`. In each execution of its body, for each pixel it computes neighboring pixel intensities (with respect to boundary condition type) and performs an SOR step. Listed for a 2D solver:

```
...
initialGuess.setTo(b/a);
iter->setTo(initialGuess);
oldIter.setTo(initialGuess);
...

int it = 0;
float alter, ip, jp, im, jm;
while (diff > toll) {
         it = it+1;
         for (int i=0;i<isize;i++) { for (int j=0;j<jsize;j++) {
                  // solving either NEUMANN(0) or DIRICHLET(1)
                  alter = (type == 0) ? iter->f[i*jsize+j] : 0.0;
                  ip = (i<isize-1) ? iter->f[(i+1)*jsize+j] : alter;
                  jp = (j<jsize-1) ? iter->f[i*jsize+(j+1)] : alter;
                  im = (i>0) ? iter->f[(i-1)*jsize+j] : alter;
                  jm = (j>0) ? iter->f[i*jsize+(j-1)] : alter;

                  // SOR step
                  iter->f[i*jsize+j] = (b.f[i*jsize+j]
```

```
                    + (a.f[i*jsize+j] / omega - a.f[i*jsize+j])
                    * iter->f[i*jsize+j]
                    - aip.f[i*jsize+j]*ip
                    - ajp.f[i*jsize+j]*jp
                    - aim.f[i*jsize+j]*im
                    - ajm.f[i*jsize+j]*jm
                    )/(a.f[i*jsize+j] / omega);
          }}
          // compute "diff"
          // set "oldIter" to "iter" for next loop
}
```

4.5 Control

The image processing chain is controlled by this code:

```
int ts=120; int is=100; int js=100;

// Load original data
Img3D orig3D(ts,is,js);
Vid orig(ts,is,js);
orig3D.loadFrom("vid_cube100ext.txt");
orig.loadFrom("vid_cube100ext.txt");

// Noise reduction
Vid smooth(ts,is,js);
for (int i=0;i<ts;i++) {
          smooth.f[i]->setTo((*(orig.f[i])).gmcf(100));
}
smooth.saveTo("vid_cube100ext_gmcf.txt");

// Cell identification - LSCD evolution
Vid cd(ts,is,js);
for (int i=0;i<ts;i++) {
          cd.f[i]->setTo((*(orig.f[i])).lscd(20));
}
cd.saveTo("vid_cube100ext_lscd.txt");

// Cell identification - local maxima
COHList3D coh(COHList3D::MAX_LENGTH);
coh.setTo(cd.centersOfHumps());
```

```
coh.saveTo("vidcohlist_cube100ext_lscd.txt");

// Initial segmentation
Vid inseg(ts,is,js);
inseg.setToInitSegm(coh, 2.9, smooth, 0.1);
inseg.saveTo("vid_cube100ext_is.txt");
Img3D is3D(ts,is,js);
is3D.loadFrom("vid_cube100ext_is_fin.txt");

// Spatio-temporal segmentation
Img3D gss3D(ts,is,js);
gss3D.setTo(orig3D.gsubsurf(1,is3D,
         0.1, // wcon
         0.1, // wdif
         1.0 // sStepSize
));

// ... trim away first and last 10 frames after segmentation
// ... take a specific isosurface as a segmentation result

// Distance functions
Vid dist1(100,100,100);
Vid distProj(100,100,100);
Vid dist2minus(100,100,100);
dist1.rtdist(500, fixed, mask);
dist1.saveTo("vid_cube100_dist1.txt");
for (int t=0;t<distProj.tsize;t++) {
         distProj.f[t]->rtdist(500, *(mask.f[t]), *(zeros.f[t]));
         dist2minus.f[t]->setTo(*(zeros.f[t])-(*(distProj.f[t])));
}
distProj.saveTo("vid_cube100_distproj.txt");
dist2minus.saveTo("vid_cube100_dist2minus.txt");

// ... reduce centers

// Extract cell trajectories
TrajList path(TrajList::MAX_LENGTH);
int * mem = (int *)malloc(100*100*100*sizeof(int));
for (int i=0;i<100*100*100;i++) mem[i] = -1;
char pathfilename[] = "trajlist_cube100.txt";
for (int i=reducedCenters.length-1;i>=0;i--) {
```

```
            dist1.twoStepPath(&path,reducedCenters.tlist[i],
            reducedCenters.ilist[i],reducedCenters.jlist[i],
            dist2minusLSCD,100,mem);
}
path.saveTo(pathfilename);
```

4.6 Result viewers

To view results of the algorithms, we have implemented some programs in Mathematica. We call them "viewers". In general, they read specific inputs from files and they use Mathematica plotting library to visualize them. We use 2D viewing functions to view a specific frame. To select a frame, we use a "slider" command tool.

To view raw and filtered data, we use `viewer_vid.nb`. To consult locations of sets of points, e.g. set of local maxima after LSCD evolution, we use `viewer_vid_centers.nb`. To check spatio-temporal segmentation results of GSUBSURF and to find the right isosurface, we use `viewer_vid_segment.nb`. We use the `viewer_vid_dist.nb` to see distance function results (both spatio-temporal distance from root cells and frame-by-frame distance from borders of tubes). To view instances of `class TrajList`, i.e. lists of trajectories found in image, we use `viewer_vid_trajlists.nb` - blue lines are positions of parental nodes in previous frame, green lines are positions of left-child nodes in next frame, and orange lines are positions of right-child nodes in next frame. A node containing both green and orange line refers to a cell division. Finally, to present tracking results by assigning a color to each root cell and using it to denote that cell and all its children throughout the video, we use `viewer_vid_colorvideo.nb`. In fig. 10 we can see the main control screen of most frequently used viewers.

4.7 Hand correction interface

We are working with biological data and we expect them to have lot of noise and irregularities. After the run of the algorithm, we therefore assume its results will be checked and corrected by the user of the software. This algorithm is optimized to minimize the amount of work to be done by the user in the post-processing.

To characterize the amount of work quantitatively, we define two elementary operations: `add_parent` and `set_parent`. `add_parent` creates a new parent for a selected node, overwriting the original parent-child connection. `set_parent` forces one selected node to become a parent of another selected node - no new node is created in this step. Both these steps modify the

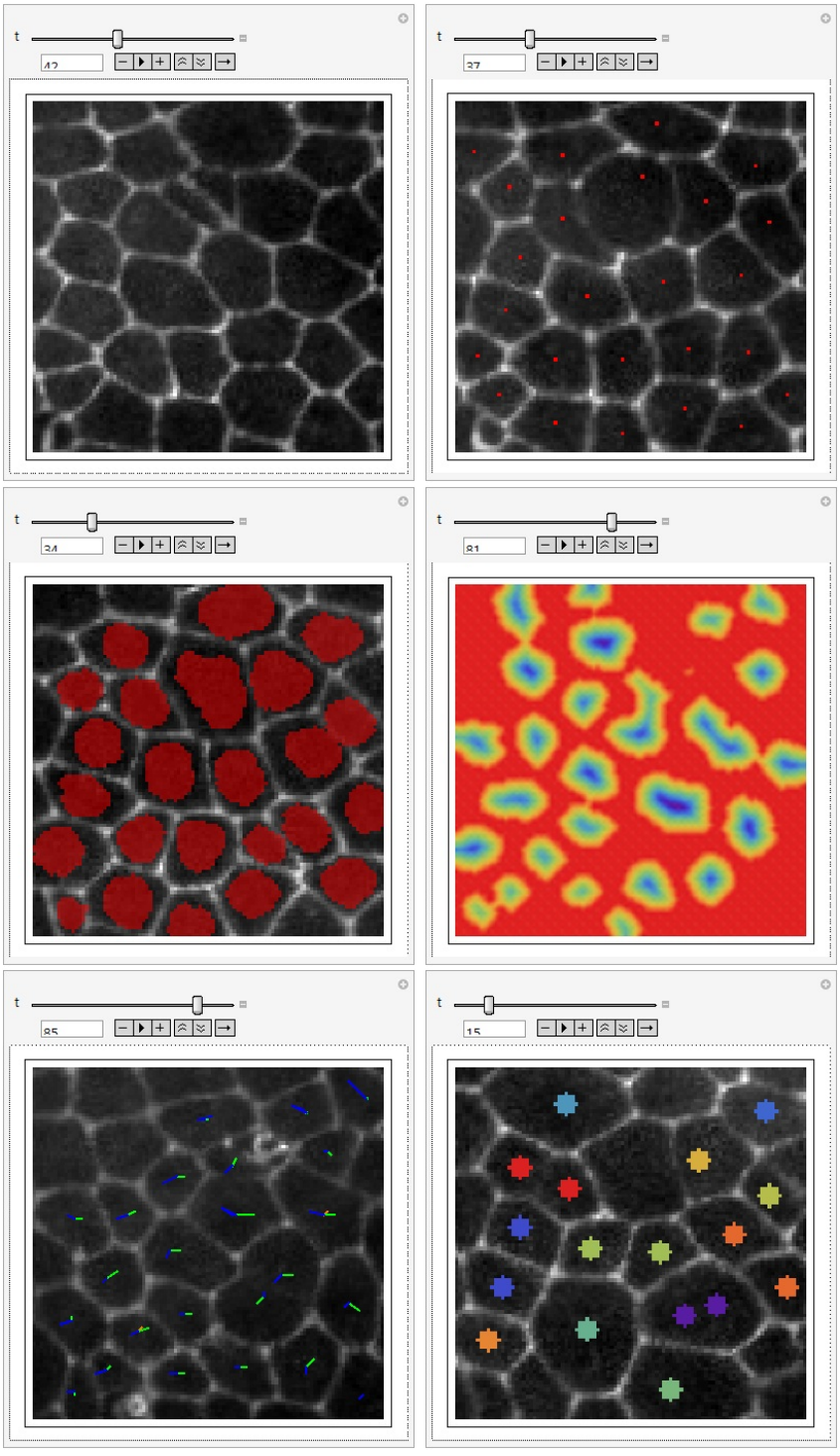

Figure 10: Most commonly used viewers. Implemented in Mathematica. Upper line - simple viewer and viewer of centers, middle line - viewer of segmentation and viewer of distance, lower line - viewer of TrajList and viewer of colorvideo.

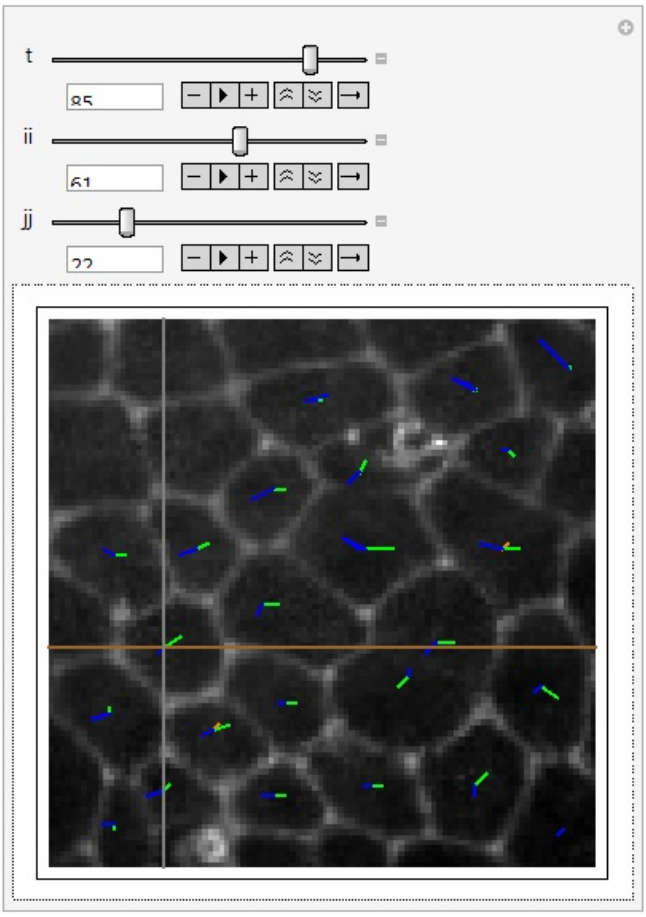

Figure 11: Hand correction interface. Three sliders select a specific point in a 3D image. Via the interface, user can pick a node, add a parent node to a picked node, or set an existing node to become the parent of the selected node.

original instance of `class TrajList`. The amount of work is the count of elementary operations to achieve the perfect tracking.

For this post-processing purpose we have implemented a simple user interface in Mathematica. In the beginning, it loads a `TrajList` from a file. Using three sliders, one can select a specific point in a 3D image, pick a node and perform `add_parent` and `set_parent` operations. User can choose to save modified `TrajList` to a file. To see the interface viewer screen, cf. fig. 11.

5 EXPERIMENTS

We have worked with a part of the video covering 100 frames with resolution 100x100 pixels. In the beginning of the video there are 12 cells in the area. These move, mostly to the right, and most of them undergo cell division a few times during the 100 frames. In the end of the video, there are 11 cells corresponding to the original cells identified in the first frame - the others disappear through the right border of the area while moving to the right. Through the left border, some new cells arrive to the area, but we do not track these, as their root cell identifiers are unknown. Using this video, we chose algorithm parameters so that it gives the best possible results. In the LSCD, $\delta = 1.0$ and $\mu = 0.000001$ and we perform 20 time steps. In the spatio-temporal GSUBSURF, setting $w_a = w_c = 0.1$, 100 time steps are performed.

Then, we took two alternative parts of the video, both covering 100 consecutive time steps, both with resolution 100x100 pixels. We wanted to test, how well do the previously set parameters behave under the new conditions. In the first alternative video, there are 16 cells in the beginning and 9 corresponding ones in the end. In the second alternative video there are 13 cells in the beginning, 19 corresponding in the end. Cells in these parts of the video also move to the right.

We measure the success of our approach by counting number of correct and incorrect links between the cell identifiers in two consecutive frames. Number of incorrect links can be understood as the required amount of work to be performed by a user of the software, in order to achieve perfect tracking. We call it a number of "hand-correction" operations.

In the first video, for which the parameters of the model were optimized, we got 1398 correct links out of 1400 total. That means 99.86% success. In the first alternative video, using the previously set parameters, we got 1759 / 1775 - 99.01% success. For the second alternative video we got 1595 / 1600 - 99.69% success.

Tracking results of the original video part can be seen in fig. 12, alternatives are to be seen in fig. 13.

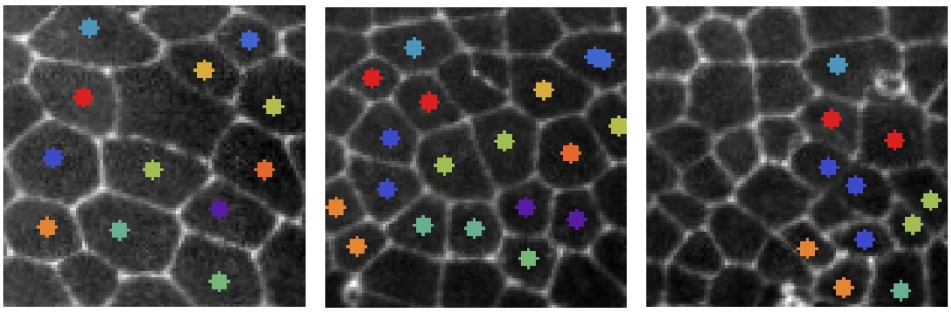

Figure 12: Tracking in the original video part. From left to right - frame 1, frame 50, frame 100.

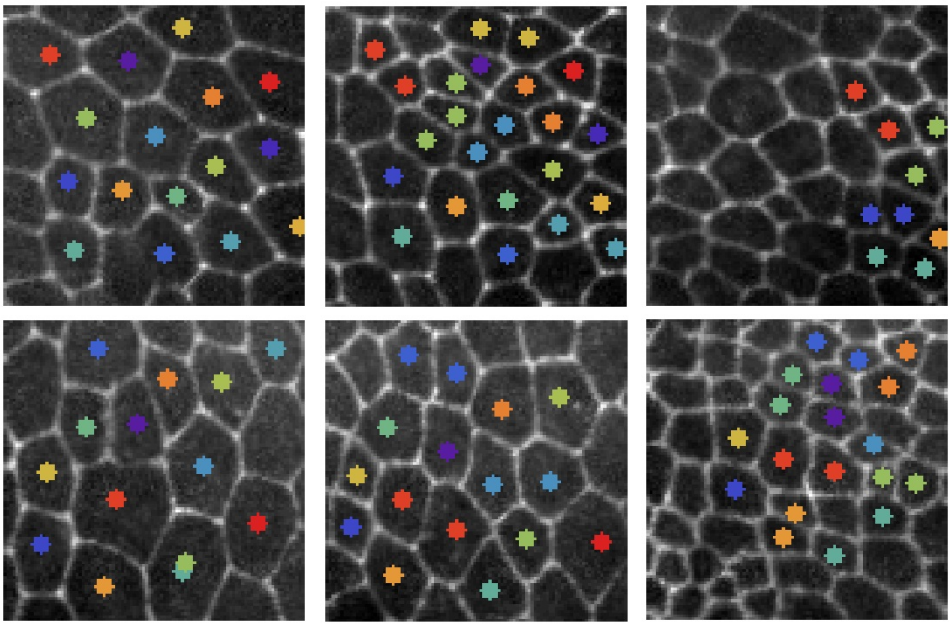

Figure 13: Tracking in alternative video parts. Upper row - first alternative part, lower row - second alternative part. From left to right (both rows) - frame 1, frame 50, frame 100.

6 CONCLUSIONS

In this paper, we have presented an algorithm for tracking cells in image sequences. An algorithm accepts lightly noised input images. It is robust against missing boundaries of cells and missing cell identifiers, thanks to spatio-temporal segmentation. It can overcome imperfections of 3D spatio-temporal tube separation, via combination of two distance functions. Parameters of model, once found, can be used for analysis of similar videos, as we have shown in section about experiments. We have developed this algorithm using a 2D+time video, but its ideas can be extended to 3D+time videos as well, see [11].

References

[1] Y. Bellaiche, M. Gho, J. Kaltschmidt, A. Brand, and F. Schweisguth, "Frizzled regulates the localisation of cell-fate determinants and mitotic spindle rotation during asymmetric cell division," *Nature Cell Biology*, vol. 3, pp. 50–57, 2001.

[2] P. Frolkovic, K. Mikula, N. Peyriéras, and A. Sarti, "A counting number of cells and cell segmentation using advection-diffusion equations," *Kybernetika*, vol. 43, no. 6, pp. 817 – 829, 2007.

[3] P. Bourgine, R. Cunderlík, O. Drblíková, K. Mikula, N. Peyriéras, M. Remesíková, B. Rizzi, and A. Sarti, "4D embryogenesis image analysis using pde methods of image processing," *Kybernetika*, vol. 46, no. 2, pp. 226 – 259, 2010.

[4] V. Caselles, R. Kimmel, and G. Sapiro, "Geodesic active contours," *International Journal of Computer Vision*, vol. 22, pp. 61–79, 1997.

[5] Z. Krivá, K. Mikula, N. Peyriéras, B. Rizzi, A. Sarti, and O. Stasová, "3D early embryogenesis image filtering by nonlinear partial differential equations," *Medical Image Analysis*, vol. 14, no. 4, pp. 510 – 526, 2010.

[6] A. Sarti, R. Malladi, and J. Sethian, "Subjective surfaces: A method for completing missing boundaries," *Proceedings of the National Academy of Sciences of the USA*, vol. 97, no. 12, pp. 6258–6263, 2000.

[7] S. Corsaro, K. Mikula, A. Sarti, and F. Sgallari, "Semi-implicit covolume method in 3D image segmentation," *SIAM Journal on Scientific Computing*, vol. 28, no. 6, pp. 2248 – 2265, 2006.

[8] K. Mikula, N. Peyriéras, M. Remesíková, and A. Sarti, "3D embryogenesis image segmentation by the generalized subjective surface method using the finite volume technique," *Finite Volumes for Complex Applications V: Problems and Perspectives*, pp. 585 – 592, 2008.

[9] P. Bourgine, P. Frolkovic, K. Mikula, N. Peyriéras, and M. Remesíková, "Extraction of the intercellular skeleton from 2D microscope images of early embryogenesis," *Proceeding of the 2nd International Conference on Scale Space and Variational Methods in Computer Vision, Voss, Norway, June 1-5,2009*, pp. 38 – 49, 2009.

[10] E. Rouy and A. Tourin, "A viscosity solutions approach to shape-from-shading," *SIAM Journal on Numerical Analysis*, vol. 29, pp. 867–884, 1992.

[11] K. Mikula, N. Peyrieras, M. Remesikova, and M. Smisek, "4D numerical schemes for cell image segmentation and tracking," *Proceedings of the Sixth International Conference on Finite Volumes in Complex Applications, Prague, June 6-10, 2011*, Springer, 2011.