Schnitzcell User's Manual

Software for Quantitative Analysis of Time-Lapse Movies

Release 1.1

November 22, 2005

Table of Contents

Preface

Some of the conventions used in this manual are described as follows. Names of specific files, e.g. `movie2-10_Schnitz.mat`, software modules, e.g. `segmoviephase.m`, or workspace variables e.g. `schnitzcells` are shown in 10-point Courier font, as are examples of commands to execute within MATLAB, e.g.:

```
>> p = initschnitz('VNmovie-01','2005-04-03','e.coli');
```

In the prior example, the MATLAB command prompt is shown by the characters ">>" and should not be typed at the MATLAB command prompt on your system. The generic directory named `/path/to/file` is used as an example whenever an absolute path to a file needs to be specified. In Windows, a complete path would be `D:/path/to/file` while in UNIX (including Mac OS) it would simply be shown as `/path/to/file`. When names of parameters used to control Schnitzcell software components, such as *segRange*, are described in text they are shown in italics using internally-capitalized (aka "camel-case") words and the normal font. These parameter names should be entered exactly as printed in the user's guide, as they are case-sensitive.

I. Introduction

The Schnitzcell software permits users to perform a quantitative analysis of time-lapse movies of fluorescent proteins in living cells constrained to grow in the focal plane of a microscope. Movie analysis proceeds in four stages:

1. Preparing for the analysis (specifying directories and analysis parameters)
2. Segmenting individual movie frames to define cell boundaries
    a. Manually checking/correcting the cell segmentation
3. Cell tracking to associate individual cell data across multiple frames and to construct a cell lineage tree (recognizing cell division events)
    a. Manually checking/correcting the cell tracking
4. Extracting fluorescence intensity data from each cell across all lineages

The latter three stages of the Schnitzcell analysis are depicted in Figure 1.

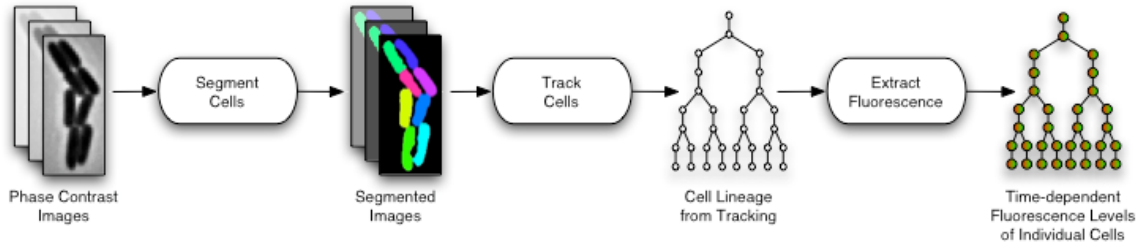

Figure 1. An overview of the Schnitzcell time-lapse movie analysis data flow.

For a more complete description of the overall single-cell tracking methodology, see the related communications brief [1].

The software is implemented as a collection of MATLAB scripts, and is designed to work within MATLAB 7 R2 on Windows. It should also work under MATLAB 6.5 and can run on Linux/Unix and Macintosh OS X operating systems, but we have not actively tested the package with these alternatives and do not support them.

The software was designed to serve the needs of a research laboratory, and attention and effort was paid to ensure the software is flexible enough to meet the diverse needs of our lab members and collaborators. Not all possible uses and misuses have been rigorously tested, and some bugs may still be lurking up some dark alleys (please report them to us!).

II. Installation

Requirements

Because the Schnitzcell software is implemented in MATLAB scripts, the software requires that you have access to a recent copy of the MATLAB software (http://www.mathworks.com). In particular it was designed to work within MATLAB version 7 SP2. The MATLAB Image Processing Toolbox is also required.

The software was developed for the Windows XP operating system. The software should also work on Mac OS X and Linux, but we do not support these operating systems. As mentioned, the software is implemented primarily as a collection of MATLAB scripts, plus an executable program used in cell tracking. That tracking program is implemented in the C programming language and is delivered as separate binary executable programs for Windows on x86 processors, for Linux on x86 processors, and for Macintosh OS X on PowerPC processors.

Obtaining the Software

You can obtain the latest release of the Schnitzcell software by downloading from the Elowitz Lab web site: http://www.elowitz.caltech.edu/software.html. This folder is password protected, and requires public users to register prior to downloading the software. Private users may be given a distinct web address, username and password to obtain the software.

The Schnitzcell software distribution comes in the form of a compressed .zip archive (e.g. `schnitzcell_1.0.zip`). Download the archive file, place it in an appropriate "software" directory and unzip it. The archive includes this users manual, `Schnitzcell_Users_Manual.pdf`, that provides complete installation and usage instructions.

Along with the software, you can also download a sample movie set to experience how the analysis system works. The sample data includes both the raw image data (required if you want to execute Quick Start Usage example), as well as the segmentation products of later stages of analysis (optional, as you can generate these yourself given the raw images only). You can place these files in another "data" directory that we will refer to later.

The image data contains 27 consecutive frames of an E.coli microcolony growth, taken at phase contrast (two slices), CFP filter set, and YFP filter set (every other frame).

- `Rawimagedata.zip` contains the 95 images.

Also downloadable are two sets of segmentation files for these movies:

- `UncorrectedSegfiles.zip` contains the direct output of running `segmoviephase` on this set of images. This set can be used to experience the manual correction interface and the rest of the software.
- `CorrectedSegfiles.zip` contains the set of manually corrected segmentation results, in which a total of 6 corrections have been made to the >600 segmented cell locations. This

set can be used directly to track the cells and to obtain fluorescence-dynamics data for these cells.

Note that you should choose to unzip only one of the above sets of segmentation files (either the corrected or uncorrected results). Because these archives contain files with identical names, unzipping one will overwrite any files unzipped by the other.

Also available are two QuickTime format movie files, `Phasemovie.mov` and `Fluormovie.mov`, that animate the progression of phase and fluorescence images, respectively. These are purely supplemental, as they are neither used nor generated by the analysis.

Setting your MATLAB Path

The Schnitzcell software distribution consists of a small directory tree containing primarily MATLAB scripts in multiple directories. When using the software for the first time the paths of these directories must be added to one's MATLAB environment. This can be done using the MATLAB GUI File->Set Path menu option, or using the MATLAB `addpath` command. The folders within the Schnitzcell software distribution that you need to add to your MATLAB session include:

- `0_setnamesdata`
- `3_segmentation`
- `4_manual_correct_seg`
- `5_tracking_correct`
- `6_manual_correct_track`
- `general_mfiles`
- `analysis_routine`

The simplest way to add these paths is to use the MATLAB Set Path tool and use the "Add with Subfolders…" option. Select the `schnitzcells` software directory, and press "okay". After adding all of the relevant subdirectories, your MATLAB search path should something like that shown in Figure 2. (Note the "Add with Subfolders" option also adds the top-level `schnitzcells` directory in your path, but it is not necessary that it be in your path.)

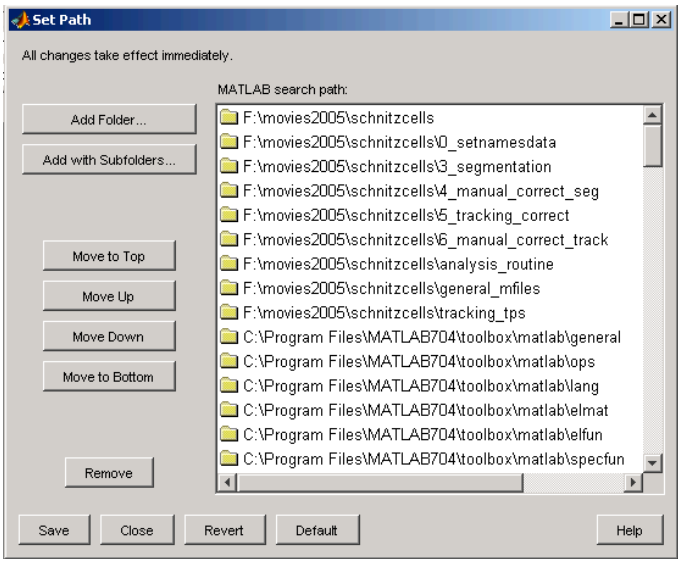

Figure 2. The MATLAB path tool can be used to add the various Schnitzcell software directories containing MATLAB scripts to your MATLAB path.

III. Quick Start Usage

The software was designed to be very easy to use for pre-defined experimental scenarios (e.g. Escherichia coli microcolonies), but flexible enough to handle more diverse experimental conditions.

Here is a quick summary of the simplest series of commands necessary to perform complete analysis of the sample movie provided:

```
>> p = initschnitz('VNmovie-01','2005-04-03','e.coli','rootDir','D:\movies');
>> p = segmoviephase(p);
>> p = manualcheckseg(p);
>> p = trackcomplete(p);
>> p = schnitzedit(p);
>> [p,schnitzcells] = compileschnitz(p);
```

In the above example, the movie name is 'VNmovie-01', which was collected on 2005-04-03, and the movie type is 'e.coli'. The main data directory under which your images and analysis results for this movie, and possibly all of your movies, is referred to as the `rootDir`. This directory will contain a movie analysis directory `2005-04-03/VNmovie-01` that in turn has sub-directories containing the images, cell segmentation results, cell lineages produced by cell tracking, and data describing the time-dependent fluorescence levels of individual cells. In this example `rootDir = 'D:\movies'` (note that Windows or UNIX paths will work here). If you followed the instructions in the earlier section titled Obtaining the Software, you should change the above value of `rootDir` to specify the "data" directory in which you unzipped the supplementary data files.

The above commands are the MATLAB routines that most users are expected to execute in order to extract fluorescence signal information from a time-lapse movie. Generally such user-level routines are located under the Schnitzcell package's `analysis_routine` subdirectory. Each of these user-level commands is described in greater detail in the subsequent sections of this manual.

To give an example of the kind of output that can be plotted once the `schnitzcells` cell lineage data structure is obtained, we list here a few sample commands that you can use to generate some illustrative plots:

```
>> [p,schnitzcells]=compileschnitz(p,'yfpOffset',0.3,'getFig3DemoFields',1);
```

This instructs the schnitz-compiler to correct for yfp autofluorescence offset of 0.3 fluorescence units. Additionally, it instructs `compileschnitz` to create the a few special fields. These fields are not recommended for standard data analysis but are specifically constructed to allow for easy plotting of the sample data. The demonstration plots are then generated using these commands:

```
>> drawschnitzbaum(schnitzcells,4);
>> plotschnitzme(schnitzcells,'mins','lengthMicrons',[],'k-');
>> plotschnitzme(schnitzcells,'FYsmins','demo_MYsNorm',[],'r.-');
>> plotschnitzme(schnitzcells,'FCsmins','demo_FCsNorm',[],'g.-');
>> plotschnitzme(schnitzcells,'demo_dFCtime',...
      'demo_dFCvals',[],'b.-','keepnans');
```

Other measured variables can be plotted in a similar straightforward way. By running the command listed on this page on the sample data set, the Schnitzcell system performs the entire process from raw images to numeric data plots.

IV. Time-Lapse Analysis Using Schnitzcell

Preparing for Analysis

To prepare for the analysis, you must specify a few analysis parameters and prepare directories to hold the images and image analysis results. At a minimum you must specify a date for the movie (e.g. "2005-04-03") and a movie name (e.g. "VNmovie-01"). You must also specify the kind of experiment movie to be analyzed. Presently this can only be "e.coli". In future releases other movie types for other species may be supported. The software is designed in a modular fashion to permits the addition of segmentation routines customized for particular species.

The command to prepare the directories for movie analysis is `initschnitz`, and the simplest example of its usage is:

```
>> p = initschnitz('VNmovie-01','2005-04-03','e.coli');
```

This will create an analysis directory (under a default `G:\` root analysis path) based on the movie name and the movie date. This new directory is named `2005-04-03/VNmovie-01` under which additional directories will be created, in particular `2005-04-03/VNmovie-01/images`, `2005-04-03/VNmovie-01/segmentation, and 2005-04-03/VNmovie-01/data`. You need to move your phase and fluorescence images into the `images` subdirectory prior to performing segmentation, though with optional arguments described later you can have greater control over where your images are located. Phase and fluorescent image files should be named according to the conventions described in the following section, Image File Naming Conventions.

The parameter structure `p` returned by `initschnitz` contains the parameters needed to perform cell analysis on the movie images. You can override any default parameters by supplying extra pairs of arguments in the form '*Parameter','Value* to `initschnitz` (or any of the analysis subroutines) using the MATLAB "properties" style of passing parameters. For more information and a list of parameters see the Command Reference section below or type "`help initschnitz`" in the MATLAB environment. Note that if the parameter structure is set to accept the changes made to it by the program functions (e.g., `p=trackcomplete(p);`) then its values may be changed by that function from their default values and some of these may need to be reset before running that or other functions again (e.g., running `p=trackcomplete(p);` can change the value of *p.trackRange*, see below).

The most common parameter you might want to override in `initschnitz` is the default root directory in which analyzed movie subdirectories should be placed. You can easily modify the root of all of these directories using the *rootDir* property. By default, this directory is set to `G:\` but to change this, for example on a Windows machine you might specify:

```
>> p = initschnitz('VNmovie-01','2005-04 03','e.coli',…
                   'rootDir','H:\MovieAnalysis');
```

In the above examples, after initializing the movie parameters and directories, you would be expected to copy all of your movie images for this movie into the `images` subdirectory that

`initschnitz` creates. `initschnitz` will generate a message will remind you to do this. Alternatively, you can override the *imageDir* property to specify an independent (possibly networked) directory that already contains your images, e.g.:

```
>> p = initschnitz('VNmovie-01','2005-04-03','e.coli',
                   'imageDir','Z:\MyMovies\Sample');
```

Specifying a pre-existing movie directory may be more convenient, as it allows you to keep all of your images from multiple movies in the same directory, rather than always copying or moving them into the movie analysis image sub-directory.

Image File Naming Conventions

The image naming convention is related to the directory naming convention described in the manual and the comments at the top of `initschnitz.m` (which you can see in MATLAB by typing "`help initschnitz`"). The movie name (i.e. the *movieName* parameter in `initschnitz`) is used as part of the analysis directory path as well as the prefix for all of the image files. For example, if the movie name is "VNmovie-01", the path to the images might be:

```
G:\2005-04-03\VNmovie-01\images
```

and the software will recognize a sequence of phase images named:

```
VNmovie-01-p-001.tif
VNmovie-01-p-002.tif
VNmovie-01-p-003.tif
VNmovie-01-p-004.tif
(... and so on ...)
```

The above naming convention is generally (in MATLAB syntax) summarized as:

```
[movieName,'-p-',frameNumber,'.tif']
```

where *movieName* is a variable containing the name of your movie (that you specify using `initschnitz`), and *frameNumber* is a 3-digit number (string) having values 000 thru 999. The `'-p-'` portion of the file name designates that these are phase contrast images, rather than fluorescence images.

The software can also recognize a sequence of phase images having two or three phase "slices" taken in different focal planes at approximately the same point in time (indicated by frame number). In this case, the phase file names should be:

```
VNmovie-01-p-1-001.tif
VNmovie-01-p-2-001.tif
VNmovie-01-p-3-001.tif
VNmovie-01-p-1-002.tif
VNmovie-01-p-2-002.tif
VNmovie-01-p-3-002.tif
```

```
VNmovie-01-p-1-003.tif
VNmovie-01-p-2-003.tif
VNmovie-01-p-3-003.tif
(... and so on ...)
```

The naming convention for movies having 2 or 3 phase slices per frame can be summarized as (in MATLAB syntax):

```
[movieName,'-p-',sliceNumber,'-',frameNumber,'.tif']
```

Where *movieName* is a variable containing the name of your movie, and *sliceNumber* is 1,2 or 3, and *frameNumber* is a 3-digit number (string), 000 thru 999.

The software will recognize movies with either of the above conventions, but uses the image names it sees in the directory to determine if there is one phase slice per frame, or 2 or 3. If you have one phase slice per frame you have to use the first (simpler) naming convention. If you have 2 or 3 you need to use the second naming convention.

It's also worth noting that the software is robust to missing frame numbers. Users are not required to provide a phase image per frame- missing frame numbers will simply be skipped.

The file naming convention for fluorescent files is similar to the 1 phase slice convention, but replaces the "p" letter of the file name with one of {c,y,g,r} to designate images acquired using CFP, YFP, GFP and RFP filter sets, respectively. Thus the software recognizes a sequence of CFP images named:

```
VNmovie-01-c-001.tif
VNmovie-01-c-002.tif
VNmovie-01-c-003.tif
VNmovie-01-c-004.tif
(... and so on ...)
```

In summary, the software will recognize fluorescence images that match any of the following patterns:

```
[movieName,'-c-',frameNumber,'.tif']
[movieName,'-y-',frameNumber,'.tif']
[movieName,'-g-',frameNumber,'.tif']
[movieName,'-r-',frameNumber,'.tif']
```

Where *movieName* is a variable containing the name of your movie, and *frameNumber* is a 3-digit number (string), 000 thru 999.

You may provide the software fluorescent images for as few as 1 filter, or as many as 4 fluorescent filters. For a given filter, the software does not require (in principle) that one fluorescent image exist per frame number, i.e. it may be possible to take one color on odd frames frame and another color on even frames. However the software is not well-tested under these conditions. To be safe we recommend you provide one fluorescent image per frame.

The frame numbers for fluorescent images must correspond to the frame numbers for the phase images. If a phase image is missing, its corresponding fluorescent image will be ignored. The

software should also be robust to any fluorescent images missing from the sequence, i.e. a phase image without a corresponding fluorescent image, although again this is not well-tested.

Segmenting Movie Frames

Segmentation can be initiated very simply, using the command:

```
>> p = segmoviephase(p);
```

This will perform cell boundary segmentation using the phase contrast images for each frame in the movie.

As mentioned earlier, this function, as with most of the other Schnitzcell movie analysis functions, will return an updated Schnitzcell parameter structure (here named `p`) containing additional fields and values defined by (or resulting from) this routine, including any fields provided in the optional list of property, value pairs.

By default `segmoviephase` will perform cell boundary segmentation using the 'first' phase image. Using our lab's acquisition software, the first phase image is slightly out of focus and produces better cell edges (see submitted paper for more details). If you wish the segmentation routine to use another phase slice during segmentation you can again override the default parameter by providing an extra *segmentationPhaseSlice,value* argument pair to the `segmoviephase` call, e.g.:

```
>> p = segmoviephase(p,'segmentationPhaseSlice',2);
```

As an alternative, you could have provided this optional parameter-value pair earlier as extra arguments to `initschnitz`, and this parameter would be stored in the resulting Schnitzcells parameter structure. E.g.:

```
>> p = initschnitz('VNmovie-01','2005-04 03','e.coli',…
                   'segmentationPhaseSlice',2);
```

As a final alternative, you can directly define or adjust any of the Schnitzcell movie parameter fields, e.g.:

```
>> p.segmentationPhaseSlice = 2;
```

One additional parameter is commonly used to control the segmentation process. By specifying the segRange variable you can control exactly frames will be segmented, e.g.:

```
>> p = segmoviephase(p,'segRange',10:25);
```

This automated segmentation process takes on the order of half a minute to a minute per frame, depending on the density of cells and the image quality (and on your processor speed, of course).

Following the automated segmentation process you can review and edit as needed each/any of the individual frame's segmentation result, using the `manualcheckseg` routine, e.g.:

```
>> p = manualcheckseg(p);
```

`manualcheckseg` is an interactive program to permit users to modify the cell segmentation result produced by the automated segmentation routine. As figure 3 shows, the routine shows the phase image and the resulting mask output of the automated cell-segmentation. The purpose of the phase image is to allow the user to verify errors in the output of `segmoviephase`.

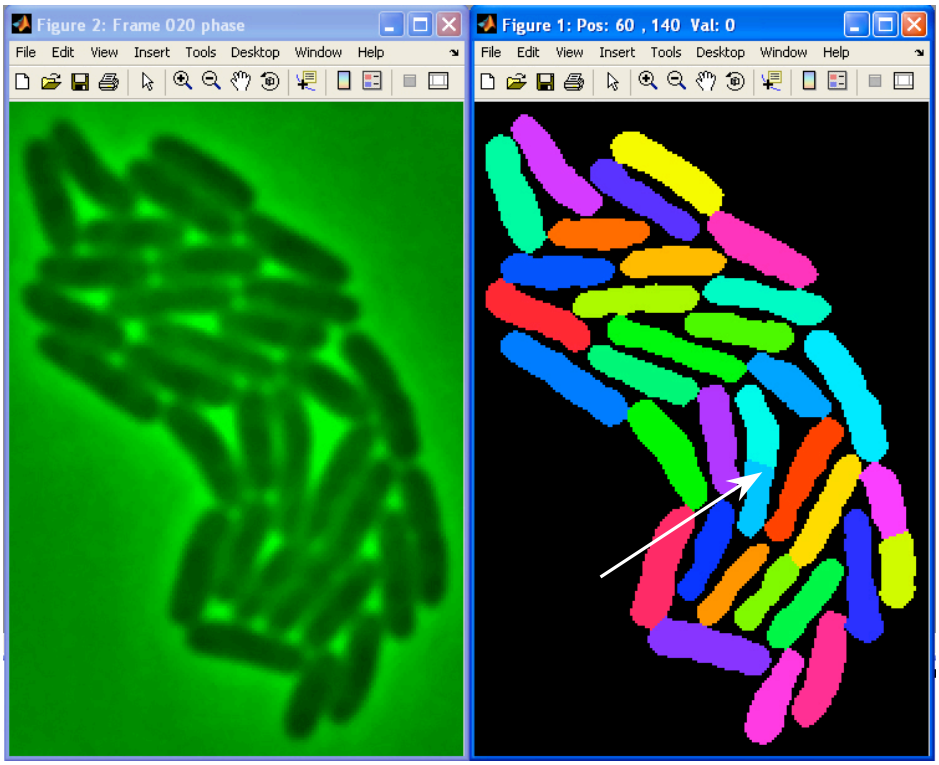

Figure 3. The `manualcheckseg` interface shows the phase image (left) and the mask output of the automated cell-segmentation (right). A white arrow, which is not part of the interface, shows a simple segmentation error - a cell that has been incorrectly split in the segmentation. This type of error can be easily corrected by consecutively left-clicking on the two halves of the cell. This joins the two halves to one cell.

The simplest usage is by going over the consecutive frames, pressing space to accept each one and to go to the next one. Occasionally, due to the random coloring of the cells, it will not be obvious whether neighboring cells have been correctly split or not - right-click on a background (black) area to reshuffle the colors. Most common segmentation errors can be corrected by left-clicking cells that have been mistakenly split (figure 3), or right-clicking on cells to split them. The point where they are split depends on where exactly they are clicked - the program searches for opposing cell edges which lie closest to the clicking point. Pressing 'a' opens a new window showing the zoomed-in phase image, on which the user can (and must) mark a polygon by left-clicking consecutive points (double click to end). The marked polygon will then be added to the

mask as a new cell. If the polygon overlaps an existing cell, the additional cell will not be created. At any time before the frame is accepted (or saved/written) the changes can be undone by pressing <esc>, reverting to the original image. Pressing space accepts the mask as it appears on the screen (with or without changes) and cycles to the next (uncorrected) frame. 'q' quit immediately without saving. Table 1 lists these and additional more "exotic" features.

Table 1 describes the mouse and keyboard actions that are used to review and modify the segmentation. (The top few actions satisfy most ordinary usage.)

`manualcheckseg` Action	Description
left mouse button on two consecutive cells	Join two cells
right mouse button inside a cell	cut cell at that point (find closest edges to cut at)
right mouse button in the background area	shuffles the label colors in the mask figure
shift + left mouse button	erase current cell
<space>	mark frame as corrected, save it and proceed to next frame
<escape>	redo this frame from the original automatic segmentation file
q	Quit (without saving current frame)
a	Add a new cell where the mouse is pointing - opens a new figure showing the zoomed-in phase image, on which the user clicks a polygon
c	crop out only populated area
b	mark the cell you are pointing to, on phase image
.	Skip frame (without saving)
,	go back a frame (without saving)
r	renumber the cell you are pointing to
p	show square around the position, on phase image
o	obliterate all but the cell you're pointing to
l	Add frame to list of badly segmented frames
x	black out an area
t	mark terraced area
f	"fine-tuning" (to avoid renumbering the image)
g	goto indexnum = ... (i.e. to frame number...)
s	save work to memory without writing to the file
w	write a temporary partial correction to the file
R	renumber all cells

Table 1. A list of actions / controls available within the Schnitzcell manual segmentation correction program `manualcheckseg`.

Again, you can provide extra arguments to `manualcheckseg` in order to modify its default parameters. On startup, `manualcheckseg` normally determines the frames that are not yet corrected, and only allows you to review those that are not yet corrected. If you wish to rerun `manualcheckseg` to review frames you have already corrected, you can set the optional *override* argument pair or `p.override` flag. If you wish to review a specific set of frames you can provide an optional *manualRange* argument pair or set `p.manualRange`. E.g.:

```
>> p = manualcheckseg(p,'manualRange',10:20,'override',1);
```

For more details on how to use or customize parameters for `manualcheckseg`, see the Command Reference section below or type "`help manualcheckseg`" in the MATLAB environment.

Tracking cells across Movie Frames

To perform automated tracking of cells across all of the frames that have been segmented into distinct cells, you simply execute:

```
>> p = trackcomplete(p)
```

The tracking proceeds by running a point-matching executable program from within MATLAB on successive pairs of frames. Once all pairs of frames have been matched, the function loads all of the pairwise point matching results, creates a cell lineage representation in memory and writes it to the *lineageName* property (which by default is placed in the movie analysis data directory and ends with '`_lin.mat`'). This file holds the results of automated cell tracking.

By default, tracking is performed only on frames for which the segmentation has been verified or corrected. The program checks the *segmentationDir* and determines which of the segmentation files contain the corrected segmentation variable `Lc`. If you wish to perform tracking prior to verifying or correcting your segmentation results, you can provide an optional *trackUnCheckedFrames* argument pair or set the Schnitzcell parameter structure's `p.trackUnCheckedFrames` flag field.

If you wish to perform tracking on a specific range of frames, you can specify an optional *trackRange* argument pair or define the Schnitzcell parameter structure's `p.trackRange` field. For example,

```
>> p = trackcomplete(p,'trackRange',10:25)
```

Again, the program will first check all of the segmentation products in the given segmentation range and proceed with it's own *segRange* for the subset of those frames that are available and corrected (unless *trackUnCheckedFrames* is set). Note that while initialization of `p` by `initschnitz` creates a parameter structure with no *trackRange* field, which indicates to `trackcomplete` to use the available manually verified frames, running `p=trackcomplete(p);` will set the value of *trackRange* to the value used in this run, and if the user needs to rerun `trackcomplete` he may need to reset that parameter manually.

Once tracking has been performed for a given pair of frames there is no need to redo the pair-wise point matching for that pair, unless the segmentation for one of the pair of frames has changed, or perhaps if some tracking control has been adjusted. If you wish to rerun the tracking you can set an optional *override* input argument or `p.override` field in the Schnitzcell parameter and then rerun the tracking. E.g.:

```
>> p = trackcomplete(p,'override',true)
```

The tracking algorithm based on a point-matching optimization algorithm (thin plate splines, aka TPS). Our tests have shown it to be faster and more accurate than the custom approach to tracking that we originally developed. To use the original tracker instead of the TPS tracker, you can simply type:

```
>> p = trackcomplete(p,'trackMethod','original')
```

Following the automated tracking process you can review and edit as needed each/any of the individual branches (aka "schnitzes") of the resulting lineage tree, using the `schnitzedit` routine.

Correcting and editing the tracking

```
>> p = schnitzedit(p);
```

`schnitzedit` can be called with an optional second argument, permitting schnitz review to begin at the specified schnitz number (a schnitz' index within the `schnitzcells` structure array). Otherwise it will start with the first schnitz.

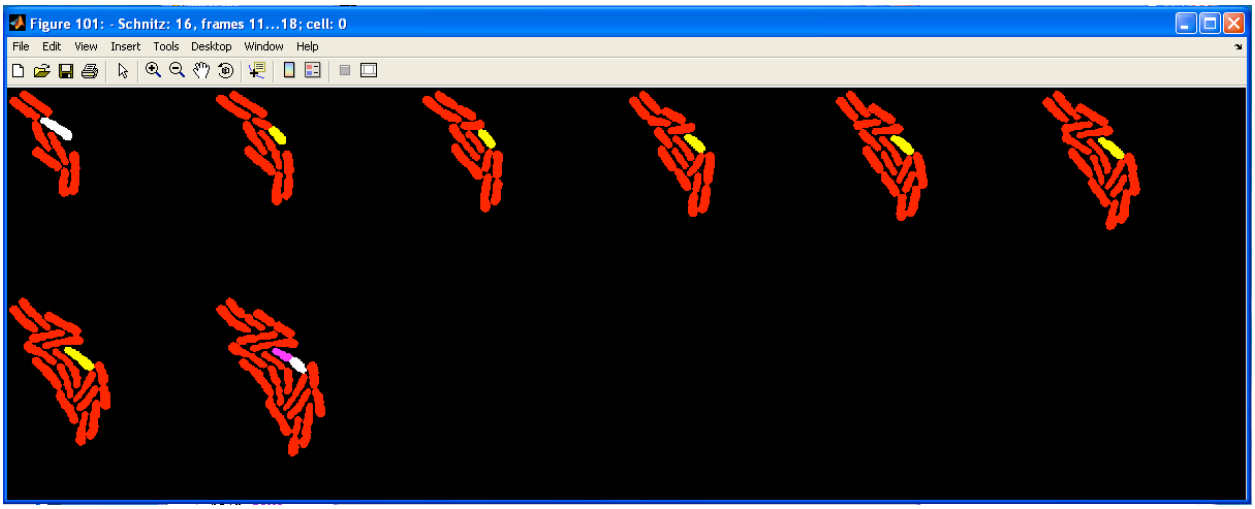

Figure 4. The `schnitzedit` interface shows successive frames of the movie with certain cells highlighted. The white cell in the earliest frame is the parent of the active 'schnitz' (a branch of a cell lineage), while the yellow cells highlight the cells of the current schnitz. The last frame shows the daughters of the division event shown in purple and white. Red cells are not a part of the active schnitz.

`schnitzedit` is an interactive program that permits you to manipulate the cell lineage structures. Figure 4 shows an illustration of the interface, while Table 2 below shows all of the available actions. A causal glance at the actions may be daunting, but don't be intimidated. Much of the work of reviewing and correcting tracks can be done using just the top few actions.

The simplest way to run `schnitzedit` is to go through each branch of the lineage tree (each schnitz), and approve it by pressing space bar. That takes you to the next one, and so on. It is also possible to click on any displayed cell at any time to go the corresponding data for that cell. It is possible to encounter at least two main types of error: segmentation or tracking errors.

If you find a segmentation error in a frame, put the mouse near the problem, and press 'e' to edit that frame. This brings up a segmentation editor much like manualcheckseg. When you have corrected the error, press 's' to save and 'q' to return back to the main `schnitzedit` program.

If you find a tracking error (i.e. a cell lineage seems to incorrectly connect cells), you may correct the problem in one of two ways, by shift+clicking on two cells in consecutive frames to connect them into a single lineage, or by ctrl+clicking on three cells, one in any frame, and its 2 daughters in the subsequent frame. This will make the three cells a division event.

Tips: Clicking in unexpected places may cause the program to stall. It is advisable to save your results frequently by pressing 'r' to renumber schnitzes, followed by 's' to save your results. The renumber operation cleans up problems in the lineage tree.

`schnitzedit` Action	Description
left click	view the tree element (schnitz) containing the cell on which you clicked
shift click on two cells on neighboring frames	connect two cells into a single schnitz
shift click on a cell and then on the black part of the next frame	terminate the cell lineage
ctrl click on three cells (a parent on one frame and two daughters on the next frame)	create a parent/daughters relationship between those cells
hover over a cell	display cell number in the title bar- very handy
, [comma]	go backward in the schnitz sequence
[enter]	go forward in the schnitz sequence
. [period]	go forward in the schnitz sequence
<space>	approve schnitz and jump to next schnitz
<ESC>	disapprove schnitz
s	save your work; this is important, as saving also automatically backs up the previous version
x	toggles Extend mode on/off; extend mode extends the time series to 3 frames before and after the first and last frame of the schnitz
j	jump to the next unapproved schnitz. Very useful during the last stages of schnitzediting.
backspace	go back to the previous frame you were looking at (kind of like clicking the "back" button on a web browser)
n	opens a new image window, leaving the previous window there for you to look at.
q	quits
a	toggles "All Color Mode"; in all-color mode,

	all the lineages are color coded with random colors that are consistent from frame to frame (except when the cell divides). More or less useless.
e	edit segmentation; this takes you to a segmentation editor for the particular frame that your mouse was pointing to when you typed E. For image editor commands, see Table 3 below
g	goes to a specific schnitz number; you will be asked to enter a number in the main window
c	checks for continuity problems; this essentially looks for errors in the tree structure
d	find Double schnitzes; this is another error-checking thing; you shouldn't need it
r	Renumber everything in the schnitz tree; this is important to do after a tracking change.
o	toggle orphan mode; n orphan mode, cells with no parents light up in blue
b	toggle barren mode; in barren mode, cells with no offspring light up green
ctrl+L	load a different schnitz file; you will be asked for a filename in the MATLAB command window
p	find a likely problem; don't use this, I don't think it works
k	enter debugging mode, using the MATLAB keyboard command...

Table 2. A list of actions / controls available within the `schnitzedit` manual tracking correction program.

Segmentation Editor Actions	Description
Q	quit the image editor and return to the main program.
G	go to a different frame number
S	save
A	add a new cell where the mouse is pointing
left-clicking on two cells	join them together
right-clicking on a cell	cuts it at the point you clicked
shift+click	deletes the cell

Table 3. A list of actions / controls available within the `schnitzedit` segmentation correction window.

Again, you can provide extra arguments to `schnitzedit` in order to modify its default parameters. For example, you can specify an optional *lineageName* argument pair or define `p.lineageName` if you wish to use a lineage file other than the default one, which is

`[p.tracksDir, p.movieName,'_lin.mat']`. For more details on how to use or customize parameters for `schnitzedit`, type "`help schnitzedit`" in MATLAB.

Collecting Fluorescence Data

Following verification and correction of the cell lineages produced by the automated tracking step, you can extract the time-dependent fluorescence information for individual cells. The routine that performs fluorescence extraction is `compileschnitz`, and it is typically called as follows:

```
>> [p,schnitzcells] = compileschnitz(p);
```

The function returns the updated Schnitzcell movie parameter structure, but also returns a second argument, a structure array representation of the cell lineage that is now highly annotated to measure for each movie frame's time point a cell's fluorescence signal levels (total, mean, median and standard deviation of each reporter's fluorescence), time stamp, cell position, area, length, width, volume and orientation, and derivatives of many of these quantities. A description of each of the many attributes derived by `compileschnitz` is shown in Tables 4 and 5.

This is the first function to return the cell lineage, in fact the first function to return anything other than the Schnitzcell movie parameter structure. Up to this point the movie analysis has proceeded entirely by performing manipulations on the Schnitzcell movie parameter structure `p` and by saving segmentation and tracking data files. Since the goal of these time-lapse analyses is to extract the cell lineages and associated fluorescence information, from this point of the analysis on, the operations are performed on the lineage that has been derived from a movie, rather than on the movie parameter structure itself.

If your images do not contain additional meta-information describing the image acquisition parameters, in particular the time of acquisition, you can provide an optional Schnitzcell parameter argument pair to define the number of minutes per frame for your movie. E.g.:

```
>> [p,schnitzcells] = compileschnitz(p,'minutesPerFrame',17);
```

The `compileschnitz` routine will multiply this value with the frame number to obtain an approximate time base for subsequent fluorescence analysis. For more information, see the Advanced Usage section entitled Image Acquisition Information.

`schnitzcells` Structure Field Name	Field Description
B	birth frame
N	life time
SZ	cell size (area)
ang	orientation (from regionprops) (given in degrees)
wid	cell width (minoraxislength)
len	cell length (majoraxislength)

volume	our best estimate of the volume of the cell in micron^3
cenX	x coordinate of centroid in full image
cenY	y coordinate of centroid in full image
thetas	angle of orientation axis (given in radians) defined from old pole (at origin) to new pole
datetime	actual date/time that the (phase) image was taken - this is a MATLAB datenumber object
mins	number of minutes elapsed since the first frame in the movie.
gen	number of generations since beginning of movie (first cells are gen=0)
gens	number of generations with respect to the number of frames for this track
phase	how far is the cell into its cell-cycle (current frame / total frames)

Table 4. A description of each of the non-fluorescence-related `schnitzcells` structure fields generated by `compileschnitz`.

The `schnitzcells` lineage structure's fields listed in Table 5 are created for each color of fluorescence {C,Y,R,G,etc.}. The first three have *NaN* for frames without fluorescence images in this channel. The rest have values only for frames with fluorescence images in this channel, their length is equal to the number of values that aren't *NaN* in the first three. The time-derivatives are calculated using the difference between consecutive frames, and their length is therefore one less than the fields used in the differentiation. They have corresponding time fields, which are the average time values for the later and earlier frames (who's difference is the derivative).

`schnitzcells` Structure Field Name	Field Description
FC	total fluorescence intensity (for CFP)
MC	mean fluorescence intensity (for CFP)
SC	standard deviation of fluorescence intensity (for CFP)
FCs	total fluorescence intensity (for CFP)
MCs	mean fluorescence intensity (for CFP)
LCs	fluorescence intensity (CFP) divided by (length·12) (12=average width)
FCsframes	frames which had fluorescence images in this channel (CFP)
FCsmins	times of fluorescence data (CFP)
dFCdt	time derivative of fluorescence (dFC/dt)
dFCmins	time of derivatives, = mid-point between the times of the two FCs
FCsmins	times of fluorescence data (CFP)
dFCmins	time of derivatives- the mid-point between the times of the two FCs

Table 5. An example and description of each of the fluorescence-related `schnitzcells` structure fields generated by `compileschnitz` corresponding to fluorescence color "C". Such fields are created for each color of fluorescence {C,Y,R,G,etc.}.

Plotting The Data

To load the Schnitzcell cell lineage data structure (`schnitzcells`) you can either use

```
>> load([p.tracksDir 'movieName-Schnitz.mat']);
```

or

```
>> [p,schnitzcells] = compileschnitz(p,'load',1);
```

You can see any of your data by looking at the specific fields, e.g.

```
>> schnitzcells(1)
```

This will also show what fields are available for one schnitz (a single cell's track) and what is their length. To plot the data, use

```
>> plotschnitzme(schnitzcells,{xaxis-field},{yaxis-field},...
      {cells-to-plot},{plot-color-and-style});
```

Where `{xaxis-field}` is the abscissa variable and `{yaxis-field}` is the ordinate, and the data will be plotted for the schnitzes numbered `{cells-to-plot}` and for their ancestors. `{plot-color-and-style}` is in the normal format for the MATLAB plot function. For example:

```
>> plotschnitzme(schnitzcells,'mins','len',1,'ko-');
>> plotschnitzme(schnitzcells,'FYsmins','MYs',[],'ro-');
>> plotschnitzme(schnitzcells,'FCsmins','FCs',[],'gs-');
```

You can choose any pair of fields to plot against each other, so long as they are the same length. E.g. volume array is same length as `mins`, but the derivative and "interp" fields are one shorter since they're calculated between original time points. So `'dFCdt','dFCmins'` could go together.

V. Schnitzcell Command Reference

This section provides additional detail on each of the Schnitzcell software's main analysis routines. The additional info includes descriptions of all optional (in some cases obscure) parameters, as well as programmer notes that describe how the function(s) are designed to perform and how they can be modified.

Initschnitz

The `initschnitz` command is used to get MATLAB session variables and file system directories set up properly before performing cell tracking and analysis operations. It lets users define and control a number of variables (aka Schnitzcell movie analysis parameters, or simply Schnitzcell parameters) used during cell segmentation, tracking and analysis, and, if necessary, will create directories to store the movie analysis results.

Running initschnitz as follows

```
>> p = initschnitz('VNmovie-01','2005-04-03','e.coli',…
                   'rootDir','F:\movies2005new');
```

returns a Schnitzcell parameter structure `p` that contains the following fields and values:

```
p =
           movieName: 'VNmovie-01'
           movieDate: '2005-04-03'
           movieKind: 'e.coli'
             rootDir: 'F:\movies2005new\'
             dateDir: 'F:\movies2005new\2005-04-03\'
            movieDir: 'F:\movies2005new\2005-04-03\VNmovie-01\'
            imageDir: 'F:\movies2005new\2005-04-03\VNmovie-01\images\'
     segmentationDir: 'F:\movies2005new\2005-04-03\VNmovie-01\segmentation\'
           tracksDir: 'F:\movies2005new\2005-04-03\VNmovie-01\data\'
```

The meaning of each Schnitzcell parameter structure field is described in Table 6.

Schnitzcell Parameter Structure Field Name	Field Description	Example
movieName	short name of movie, with no spaces	'VNmovie-01'
movieDate	date of movie in YYYY-MM-DD format	'2005-04-03'
movieKind	type (species) of movie, e.coli is presently the only option	'e.coli'
rootDir	top-level directory containing results of cell segmentation/tracking	'F:\movies2005new\'

	analysis for multiple movies	
dateDir	directory of analyzed movies (images, results) obtained on a given date	'F:\movies2005new\2005-04-03\'
movieDir	directory for analysis of a single movie (images and results)	'F:\movies2005new\2005-04-03\VNmovie-01\'
imageDir	directory of phase and fluorescence images	'F:\movies2005new\2005-04-03\VNmovie-01\images\'
segmentationDir	directory of frame segmentation files	'F:\movies2005new\2005-04-03\VNmovie-01\segmenetation\'
tracksDir	directory of cell lineage data files	'F:\movies2005new\2005-04-03\VNmovie-01\data\'

Table 6. A list of the basic Schnitzcell parameter structure fields.

As mentioned in the section titled Preparing for Analysis, any additional argument pairs in the form '*Parameter',Value* provided to Schnitzcell will create an additional field (or override a basic field) in the resulting parameter structure.

segmoviephase

```
function p = segmoviephase (p,varargin)
% SEGMOVIEPHASE   Segment movie from phase images.
%
%   SEGMOVIEPHASE(P) segments the movie from phase images according to the
%   parameters and controls described in the schnitzcells parameter structure P,
%   typically generated using INITSCHNITZ.
%
%   SEGMOVIEPHASE(P,'Field1',Value1,'Field2',Value2,...) segments the movie
%   from phase images according to the parameters and controls described in the
%   schnitzcells parameter structure P, but only after P has been updated by
%   setting P.Field1 = Value1, P.Field2 = Value2, etc.  Thus any schnitzcells
%   parameters can be updated (overridden) in the function call via these
%   optional field/value pairs.  (This is in the style of setting MATLAB
%   properties using optional property/value pairs.)  For a complete list of
%   schnitzcells parameter structure fields, see INITSCHNITZ.
%
%   SEGMOVIEPHASE returns a struct array, the updated schnitzcells parameter
%   structure p that reflects fields updated via any optional field/value
%   arguments, as well as fields updated to reflect the status of the
%   segmentation at the time the segmentation exited.
%
%   For example, P = SEGMOVIEPHASE(P,'segRange',[5:10]) would perform
%   segmentation on frames 005 thru 010 of the movie described in P, and
%   return the updated schnitzcells parameter structure P.
%
%--------------------------------------------------------------------------
% Schnitzcells Fields / Parameters that you can adjust to control segmoviephase:
%
%   segRange                range of frame numbers to segment; by default
%                           all image frames in imageDir will be segmented
%
```

```
%   prettyPhaseSlice        the phase image number with best focus; with more
%                           than one phase slice this defaults to 2, otherwise
%                           defaults to 1
%
%   segmentationPhaseSlice  the phase image number used when performing cell
%                           boundary segmentation; defaults to 1
%--------------------------------------------------------------------------
```

The cell segmentation process is carried out by the `segmoviephase` function and it's subroutines. Although the function was designed initially for segmenting e.coli cells, we believe the approach will be applicable to additional cell types. Generally, the routine analyzes one or more phase contrast images in order to identify the edges of distinct cells. The specific steps involved in cell segmentation, carried out by the subroutine `segphase`, are described below:

1. find cell edges in phase image (slice specified by *segmentationPhaseSlice* value)
2. find image mask (the smallest region containing cells)
3. fill cells (using `imcomplement`)
4. fill edges using cells found from `imcomplement`
5. break up cell clumps (using `cutcurvbpts`)
6. cut up individual cells (using `cutcurv`)
7. cut up any long cells remaining (using `breakcell`)
8. cut up any "kinky" (bent) cells remaining (using `breakcell` and `dekinker`)
9. remove any small cells left
10. relabel cell segmentation image

While the `segmoviephase` function and its subroutines were designed initially for segmenting e.coli cells, the routines were also designed to be customized to permit segmentation of different types of cells by specifying the *movieType* Schnitzcell parameter. The *movieType* variable permits the segmentation to be customized in two ways. First, the segmentation functions can simply perform different segmentation operations based on the *movieType*. Second, any segmentation operations that are shared across movieTypes can have parameter settings that vary with *movieType*. For example, the parameter settings for *movieType* = 'e.coli' that control various aspects of the segmentation process are listed below.

```
        edge_lapofgauss_sigma: 2
                    minCellArea: 30
    minCellLengthConservative: 20
                minCellLength: 30
                 maxCellWidth: 7
                 maxThreshCut: 0.3000
                maxThreshCut2: 0.2000
                    maxThresh: 0.2500
                    minThresh: 0.2500
                    imNumber1: 2
                    imNumber2: 1
                       radius: 5
                    angThresh: 2.7000
               numphaseslices: 3
```

These cell-type-specific segmentation parameters are described in more detail in Table 7.

Schnitzcell Parameter Structure Field Name	Field Description	Default Value for movieType = 'E.coli'
edge_lapofgauss_sigma	Controls the width (the sigma, in pixels) of the Laplacian of Gaussian filter used in edge operations. The size of the filter is n-by-n, where n = ceil(sigma*3)*2+1. Use a higher number for fatter cells.	2
minNumEdgePixels	When defining colony mask using edge image, this is the minimum number of edge pixels in a 40x40 pixel region needed to make a location count as outside the mask of the cell colony area.	250
minCellArea	Minimum number of pixels of which all cells must be composed.	30
minCellLengthConservative	Minimum major axis length for all cells.	20
minCellLength	Minimum length of a cell produced by cell cutting routines.	30
maxCellWidth	Maximum distance between cell edges at possible cut locations examined by cell cutting routines. I.e. two points must be closer than this for a cut to be accepted.	7
maxThreshCut	During cell clump cutting (`cutcurvbpts`), this is a threshold on steepness of maxima when determining how many cuts to make. The smaller *maxThreshCut* is, the more branch points are cut.	0.3
maxThreshCut2	During individual-cell cutting at narrow waists (`cutcurv`), this is a threshold on the steepness of maxima when determining how many cuts to make. The smaller *maxThreshCut* is, the more narrow points are cut.	0.2

maxThresh	While breaking up big cells (`breakcell`), this is a threshold on the steepness of phase image maxima when determining how many cuts to make. The smaller maxThresh is, the more phase-image maxima points are cut.	0.25
minThresh	While breaking up big cells (`breakcell`), this is a threshold on the steepness of segmented image minima when determining how many cuts to make. The smaller *minThresh* is, the more segmented-image minima points are cut.	0.25
imNumber1	The phase image slice number used when breaking up big cells.	1
imNumber2	The phase image slice number used when breaking up big cells having low solidity.	2
radius	The routine for cutting up kinky cells (`dekinker`) finds the angle between two points along "thin" of cell at a distance *radius* pixels away from a centre point.	5
angThresh	While cutting kinky cells (`dekinker`), this is a threshold on the minima of angles when determining how many cuts to make. The larger *angThresh* is, the more kinky points are cut.	2.7

Table 7. A list of the cell-type-specific Schnitzcell parameter structure fields available for `movieType = 'E.coli'` used by the cell segmentation component.

manualcheckseg

```
function p = manualcheckseg (p, varargin);
% function P = manualcheckseg (p, varargin);
%
%   MANUALCHECKSEG allows users to review the results of image segmentation
```

```
%   in order to manually correct any merged cells or delete false positives.
%
%   MANUALCHECKSEG(P,'Field1',Value1,'Field2',Value2,...) also performs
%   manual/interactive correction of segmentation results, but permits users
%   to adjust any parameters describing the movie or parameters controlling
%   the manual segmentation checking process.  The routine first sets all of
%   the movie analysis parameters to their default values defined for the given
%   movieKind, and then overrides any specific parameters provided by setting
%   P.Field1 = Value1, P.Field2 = Value2, etc.  Thus any/all schnitzcells
%   parameter values can be defined in the function call via these optional
%   field/value pairs.  (This is in the style of setting MATLAB properties
%   using optional property/value pairs.)
%
%   MANUALCHECKSEG returns a struct (1x1 struct array) referred to as the
%   schnitzcells parameter structure that contains fields and values
%   contained in P, including unchanged/original parameters plus any of those
%   added or overridden in the list of properties & values provided in the
%   optional arguments.
%
%--------------------------------------------------------------------------
% Schnitzcells Fields / Parameters that you can adjust to control manualcheckseg
%
%   outprefix     overrides [p.movieName 'seg']
%   manualRange   specify frame range to check
%   override      if 1, program will redo frames already having Lc; default=0
%   expandvalue   increases each segmentation image border by this amount
%                 (in case the segmentation cut off some cells); defaults to 30
%   frnum         check one frame only, also sets p.override = 1
%   Dskip         (what does this do? probably frame skipping?), defaults to 1
%   upend         frame size / location; default is 680
%   leftend       frame size / location; default is 516
%   min_size      [height width] of figure; default [upend-60 leftend-10]
%   finetuneimage if 1, program will not renumber the image; default = 0
%   regsize       maximum size in pixels of any translation between phase and
%                 fluorescent images; default is 3
%--------------------------------------------------------------------------
```

`manualcheckseg` is a straightforward GUI routine that permits users to view and optionally modify the cell segmentation image one frame at a time. There is a main loop that iterates through each frame in the *manualRange* frame range. `manual_kant` is the most significant subroutine that handles the mouse clicks and character presses following the display of a single frame's segmented cells. After `manual_kant` processes key presses it returns control to `manualcheckseg`, which is responsible for saving the modified segmentation result to the segmentation file for the current frame and moving on to the next frame.

By convention, a segmentation file has a field named `Lc` if it has been manually corrected. The `Lc` field is first created from the initial `LNsub` and subsequently modified by `manualcheckseg`. Traditionally the subsequent step (cell tracking) requires every frame tracked first be corrected. A new feature in `trackcomplete`, *trackUnCheckedFrames*, permits tracking of frames that have not been corrected.

One `manualcheckseg` action, 'c' for crop, presently uses a hard-coded variable `extra` (who's default value is 15 pixels) to control the spacing of cells from the edge of the segmented frame. Because cropping often assists the cell tracking phase we are considering making cropping an automatic feature, either in `manualcheckseg` or directly in tracking.

It should also be noted that presently the action 'e' for expand image is not functioning, though we intend to re-introduce it in the future.

trackcomplete

```
function p = trackcomplete (p, varargin)
% TRACKCOMPLETE     Track cells across frames in one complete movie.
%
%   TRACKCOMPLETE allows users to track cells identified by cell segmentation
%   across all frames for a movie.  Cells in one frame are matched to cells
%   in adjacent frames, and cell division is recognized by associating each
%   parent cell with its children.
%
%   TRACKCOMPLETE(P,'Field1',Value1,'Field2',Value2,...) also performs cell
%   tracking using segmentation results, but the extra arguments permit users
%   to adjust any parameters describing the movie or parameters controlling
%   the cell tracking process.  The extra arguments can override any specific
%   parameters provided in P by setting P.Field1 = Value1, P.Field2 = Value2,
%   etc.  Thus any/all schnitzcells parameter values can be defined in the
%   function call via these optional field/value pairs.  (This is in the style
%   of setting MATLAB properties using optional property/value pairs.)
%
%   TRACKCOMPLETE produces an output file (a MATLAB binary file) that
%   contains the 'schnitzcells' variable describing each cell's lineage. The
%   path name of this file defaults to [p.tracksDir,p.movieName,'_lin.mat'],
%   but can be specified by providing an optional 'lineageName' field in the
%   schnitzcells parameter structure, or by providing an extra
%   'lineageName',value pair of arguments to this function.
%
%   TRACKCOMPLETE returns a struct (1x1 struct array) referred to as the
%   schnitzcells parameter structure that contains fields and values
%   contained in P, including unchanged/original parameters plus any of those
%   added or overridden in the list of properties & values provided in the
%   optional arguments.
%
%--------------------------------------------------------------------------
% Schnitzcells Fields / Parameters that you can adjust to control trackcomplete
%
%   lineageName   the schnitzcells cell lineage structure is written to this
%                 file, by default named [p.tracksDir p.movieName '_lin.mat']
%
%   trackMethod   name of tracking method, one of {'tps','original'}; tps is
%                 the default tracking method
%
%   trackRange    range of frame numbers to track across; by default all
%                 frames containing segmentation files with corrected
%                 segmentation fields will be used in tracking
%
%   trackUnCheckedFrames  flag that when set to true or 1 permits this
%                 routine to perform tracking on segmentation files that were
%                 not manually verified or corrected (i.e. they do not have Lc)
%
%   override      flag that when set to true or 1 permits this routine to
%                 perform tracking on pairs of frames even if tracking was
%                 previously performed for those frames
%
%   transMax      maximum translation accepted; default is 30
%
%   len           size of subimage used for cross correlation; default is 80
%--------------------------------------------------------------------------
```

`trackcomplete` performs cell tracking by matching points in successive pairs of frames, e.g. by first finding matches for cells in frames 0 and 1, then finding matches for cells in frames 1 and 2, and so on. The cell matching is carried out by an external executable program that performs somewhat generic point matching using an optimization approach that permits points to be

shifted some subject to a thin-plate spline (TPS) transformation. The routine also permits cells to appear in only one of the two frames, and has special provisions to detect cells that have divided in the newer frame. Once cell matches have been determined for all pairs of adjacent frames, all of the pair-wise frame's results are loaded into memory and cell "tracks" following matched cells across frames are constructed.

The point-matching program requires two simple text file inputs and generates an output text file titled `Match.dat`. The tracking wrapper function, `tracker_tps`, renames each of these output files and saves them in the movie's data (aka tracks) directory with file names such as '`VNmovie-01-tps-output-005-to-010.txt`' so that they may all be reused at a later time. The point-matching program's standard output should be written to a similarly named file ending with the '`.log`' extension. The point matching program is currently designed to accept as input two text files containing one line per cell, each cell described by parameters `x`, `y`, `z` (`z` is unused, always set to 1), `length`, `angle`, and `cell_id`. The program also requires that a text file named softconfig.dat be in the current working directory. This file contains a variety of parameters that can be used to control the point-matching process.

Parameter Name	Default Value of Parameter
BetaInit: initial temperature	0.1
BetaIncr: temperature increment	1.075
BetaMax; final temperature	100.0
lambda1; warp/deformation penalizing coefficient	1.0
lambda2; affines penalizing coefficient	0.0000001
TH; theta-parameter: defines number of slacks, should be around 1	1.0
rEvar; reversals of variance	.0009
alphaS; shrinking coefficient; the anisotropy scaling parameter. An alpha value equal to one corresponds to the square of the distance between the matched points as the energy term	.8
MaxSplit; 1,2,or 3: 1 for non-splits, 2 for 2 perm. Matrices, 3 for 3	3
# of X column	1
# of Y column	2
# of Length column	4
# of Angle column	3
# of ID column	5
# of Z column if there is any	3

Table 8: Parameters controlling the point-matching program and their default values, as specified in the softconfig.dat file.

For a more detailed description on how the point-matching program operates, see the 2005 Nature Methods brief and its supplemental information.

It is also worth noting that our original ad-hoc cell tracking routine is implemented within the `trackcomplete` function. This original tracking method can be selected by setting the optional *trackMethod* parameter to 'original'.

schnitzedit

```
function p = schnitzedit(p,varargin);
%
% edit lineages one schnitz at a time: The term "schnitz" refers to
% one element of the schnitzcells array.  It represents a single
% branch of the tree.
%
% 1. Running the program: schnitzedit(schnitzcells, startnum).
% a.    The second argument is optional.  If you include it, the program
% will start showing you the specified schnitznum.  Otherwise it will
% start with the first schnitz.
%
% Optional Parameters (varargin):
%
%     'lineageName',filename: this pair of arguments will make schnitzedit
%        load the schnitzcells lineage structure from the given filename
%        instead of the default file, [p.tracksDir p.movieName '_lin.mat'].
%
%     'schnitzNum',value: this pair of arguments makes schnitzedit start
%        displaying/editing the schnitz track number specified by value.
```

`schnitzedit` is a substantial GUI function, and most of its subroutines are contained within the `schnitzedit.m` module. The main loop iterates over all schnitzes in the file, providing the user an opportunity to review and optionally correct the cell tracks produced by the cell tracking phase. The small callback routines that handle the many and various schnitzedit actions comprise a significant portion of the code. Another set of subroutines deals with properly reconnecting the schnitzes upon receiving clicks that change the lineage. One subroutine, `mouse2coords`, handles the mapping of mouse clicks on the image canvas to specific x,y location within one frame of a schnitz.

A number of constants control the appearance (e.g. window sizes, number of columns) and initial modes of operation (e.g. extended mode, orphan mode). These constants are all defined at the top of the module, and in the future can be made optional arguments if desired.

compileschnitz

```
function [p,schnitzcells] = compileschnitz(p,varargin)
% COMPILESCHNITZ  given a lineage, suck out fluor info & calc area, length, etc
%
%   COMPILESCHNITZ allows users to extract fluorescence info from a
%   cell lineage tree that describes how cells are tracked across frames of
%   a movie.  The fluorescence info is attached to the tree, and the resulting
%   annotated tree, called the "schnitz", is written to the schnitzName file.
%
%   COMPILESCHNITZ(P,'Field1',Value1,'Field2',Value2,...) also performs
%   cell fluorescence extraction, but the extra arguments permit users
%   to adjust any parameters describing the movie or parameters controlling
%   the fluorescence extraction process.  The extra arguments can override
%   any specific parameters provided in P by setting P.Field1 = Value1,
%   P.Field2 = Value2, etc.  Thus any/all schnitzcells parameter values can
%   be defined in the function call via these optional field/value pairs.
%   (This is in the style of setting MATLAB properties using optional
%   property/value pairs.)
%
%   COMPILESCHNITZ returns a struct (1x1 struct array) referred to as the
%   schnitzcells parameter structure that contains fields and values
%   contained in P, including unchanged/original parameters plus any of those
```

```
%   added or overridden in the list of properties & values provided in the
%   optional arguments.
%
%--------------------------------------------------------------------------------
% Schnitzcells Fields / Parameters you can adjust to control COMPILESCHNITZ:
%
%   lineageName    name of un-annotated input lineage name (default filename is
%                  [movieName '_lin.mat'])
%
%   schnitzName    name of annotated output schnitz file (default filename is
%                  [movieName '-Schnitz.mat'])
%
%   flatFieldName  name of file with flat field parameters
%
%   quickMode      flag, if true, will make it run in quick mode, i.e. it
%                  will skip extracting the fluorescence from the images
%                  and only calculate derived fields.  Default value is false.
%
%   load           flag, if true, will load the saves schnitz file
%
%   autoCFL        default = 0
%   yfpOffset      Minimal yfp values? exposure times? corrected? default = 0.
%
%   crosstalkRatioC2Y    default = 0.0005
%   micronsPerPixel      default = 1/15
%
%--------------------------------------------------------------------------------
```

`compileschnitz` was written to be general with respect to specific fluorescence colors. In particular, two key subroutines, `add_interps`, (which adds an interpolated new field named "newfieldname" that interpolates an old field's values at the times defined by the `interpatfield` value), and `add_derivs` (computes derivatives of fields within a schnitz), are completely general with respect to color. This should allow the routine to easily scale to handle many more than the four 'C','Y','G','R' colors it is currently designed to handle.

VI. For the Advanced User

Image Acquisition Information

The Schnitzcell software attempts to read some image acquisition information from each movie image (both phase contrast and fluorescence images). If the TIFF image contains an optional tag field named *ImageDescription* it will attempt to parse information from this field's value. Our lab's image acquisition software automatically creates this additional field for each image including its acquisition parameters. Here is an example value of such *ImageDescription* information:

```
Transmission Image, Acquired at 2005-09-30 17:29:51, Exptime= .25, Gain= 1,
comments: ND=1.3
```

From this string software will extract the image acquisition date (2005-09-30), time (17:29:51), exposure time in seconds (0.25), and the gain setting (1 meaning "low"). The exposure time is used by `compileschnitz` when normalizing fluorescence values. Values for gain are expected to be one of {1,2,3} meaning "low", "medium" or "high" gain, respectively, though at present the software does not use this information.

If an *ImageDescription* field is not included in the TIFF images, then the software is not aware of any acquisition parameters. In this situation the software presently assumes the number of minutes elapsed is equal to an image's frame number times the number of minutes per frame. This value can be controlled by providing an optional Schnitzcell parameter field *minutesPerFrame*, which defaults to 10 minutes / frame. In the routine `schnitzcells/analysis_routine/compileschnitz.m` look for *minutesPerFrame* to see where the default value is defined, and "`fakeMin`" to see how it is used.

The Schnitzcell Lineage Data Structure

The Schnitzcell cell lineage structure is represented by the MATLAB variable `schnitzcells` contained in the binary MATLAB cell lineage file pointed to by `p.lineageName`, which defaults to `[p.tracksDir,p.movieName,'_lin.mat']`. The `schnitzcells` cell lineage variable is a 1 x *NS* structure array containing one structure per lineage "track", aka "schnitz", where *NS* is the number of tracks used to describe the lineage of all cells tracked within the analyzed frames of a movie. A single schnitz describes one cell's location in time and space within the movie, from the time the cell is born (it's parent divided) until the cell itself divides. A schnitz is usually associated with a parent schnitz and two children schnitzes.

A schnitz is a track that begins at the first frame a distinct cell appears in, and ends at the last frame that distinct cell appears in. When a cell divides its track ends, and two new tracks being at the first frame it's two children appear in. A schnitz (track) is numbered by its position (index) within the structure array. A minimal schnitz structure is described by `P` (the schnitz number of it's parent), children schnitz numbers `D` and `E`, sister schnitz number `S`, `frames` (an array) listing

each frame this schnitz occurs in, `cellno` (an array) listing each cell number of the tracked cell within each of the segmented frames, and `N`, the number of frames that distinct cell appears.

`schnitzcells` Structure Field	Field Description
`P`	parent schnitz number, or -1 if this is an orphan schnitz
`D`	first daughter schnitz number (containing oldest pole), or -1 if this schnitz has no daughters (barren)
`E`	second daughter schnitz number, or -1 if this schnitz has no daughters (barren)
`frames`	frame numbers during which this schnitz' cells existed
`cellno`	cell number within each frame's segmented image
`cenx`	x coordinate of centroid in LNsub
`ceny`	y coordinate of centroid in LNsub

Table 9. A description of each of the `schnitzcells` structure fields generated by `trackcomplete`. A collection (a structure array) of these objects is used to represent the many tracks or "schnitz" that together describe lineage of cells throughout a movie.

Beyond the above schnitz fields used to describe the basic cell identities and relations to each other, `compileschnitz` uses the knowledge of a cell's occurrence within a movie to extract a variety of measures of the cell's appearance including fluorescence signal.

Tracking Methods, Output Files and Track Representations

The default tracking method is 'TPS', which is short for thin-plate-spline. The tracking routine is actually a variant of the TPS method, and is described in greater detail in the following section. Prior to using the TPS tracking method we employed a custom tracking approach. Today this older tracking approach is referred to as the 'original' track method. It can be used within trackcomplete by specifying *trackMethod* and providing the value `'original'`, for example:

```
>> trackcomplete(p,'trackMethod','original')
```

The original tracking program produces an intermediate tracking file (specified by *trackName,* defaults to `[p.tracksDir,p.movieName,'_Tdata.mat']`) that contains a number of fields, the most important being `cellmat` and `trackRange`. In the past the *trackName* file was the primary output of the original tracker, and it was used to contain this alternative tracking representation. In the past this file also included yet another representation, `lincellnum`, but this has been removed from the *trackName* file because it is useful elsewhere (for efficient computation

purposes), in particular at the beginning of `compileschnitz`. Presently this representation is simply derived on-the-fly by such routines that need this representation.

`lincellnum` is a 1 x *NF* cell array that contains the schnitz number that each cell (in each frame) belongs to. `lincellnum` has as many cells (*NF*) as there are frames in trackRange, and each cell corresponds to a frame number in `trackRange`. For each frame, it has an array with as many entries as there are cell numbers in that frame. Each array's entry lists the schnitz structure number for the cell with the same cellno as the array index.

The TPS Tracking Algorithm and Code

The algorithm has been implemented in the C++ kernel of the tracking part of the package for the general case of N-dimensions. It was then wrapped with the MATLAB function `tracker_tps.m`

The most essential parameters are contained in the configuration file "`softconfig.dat`" created after the first run of the code. The default values can then be modified if necessary.

They are as follows

1. *BetaInit*, *BetaMax* – reversals of initial and maximum temperature.
2. *lambda1* and *lambda2* – deformation and affine transformation penalizing coefficients.
3. *alphaS*- the anisotropy scaling parameter. The alpha value equal to one corresponds to the square of the distance between the matched points as the energy term.

All configuration parameters are contained in the header file `soft.h`, the code must be recompiled if `soft.h` is edited.

The run of the code can be optionally illustrated in real-time monitoring window with plotted positions (crosses) of the points from both frames as well as the current positions (squares) of the transformed points. PGPLOT package was used to implement the real time monitoring. When temperatures are lowered sufficiently and elements of probability matrices exceed the threshold value 0.5, links are created between appropriate pairs of points, denoted by arrow in the monitoring window.

Ideally, when running from very high temperatures, the process in the monitoring window would show gradual movement of the squares from some central region of the population to their final positions with gradually appeared link arrows. However, too high initial temperature showed increased probability of sticking to a state with local minimum of the energy function instead of the global one. This is especially true for fewer number of points. To avoid this, initial positions of the "seeking points" were set increasing *BetaInit* to a value around one.

This algorithm can be somewhat sensitive to large jumps in the segmentation sub-image rectangle, which can be controlled by using the "crop" feature in `manualcheckseg`. In the future the software may address this issue automatically by effectively performing an image crop before calling the tracking routine.

VII. Bibliography

1. Rosenfeld N, Bacarian T, Young JW, Roden J, Gor V, Alon U, Swain PS, Mjolsness E, Elowitz, MB: **Automated tracking of cell proliferation and gene expression in time-lapse movies**. *Nature Methods* 2005 (in review).