This user guide contains basic instructions for installing and running the cell lineaging software package "Tracking with Gaussian Mixture Models" (TGMM). The code provided here has been tested with the 64-bit version of Windows 7 and with the 64-bit version of Ubuntu Linux 12.04 LTS, using a variety of CUDA-compatible NVIDIA GPUs.

Table of Contents

1. Contents of the TGMM package

We assume that the user has uncompressed the file "TGMM_NM2013_paperRelease.zip" in a folder of their choice, referred to here as $ROOT_TGMM. The subfolders in $ROOT_TGMM contain the following components:

- *src:* All source code files. This folder also includes a CMakeList.txt file that can be used to generate a Visual Studio solution (using CMake) and/or compile the source code. The latest source code can be obtained from the sourceforge code repository using the following git command: git clone git://git.code.sf.net/p/tgmm/code tgmm-code
- *doc:* Documentation of the TGMM software.
- *build:* A Visual Studio C++ 2010 project generated from src using CMake. This subfolder also contains precompiled binaries, suitable for running the code without the need for re-compiling the source code.
- *data:* Contains a three-dimensional time-lapse data set with 31 time points (corresponding to a cropped sub-region of the *Drosophila* SiMView recording presented in the main text of Amat *et al.*, Nature Methods 2014), for testing the TGMM code and ensuring that the software is running as expected. The test data set can also be download separately from the sourceforge website.

Note: The Visual Studio project will not compile unless the folder "build" is copied to the same absolute path as that used to generate the project. We provide the full project folder primarily as a reference for the final structure of a successful Visual Studio solution.

2. Installation and software requirements

In order to run the precompiled binaries, the following auxiliary software package must be installed as well:

- CUDA Toolkit 5.5: required to run algorithms on an NVIDIA GPU
  Download: https://developer.nvidia.com/cuda-toolkit-archive

  In Ubuntu Linux distribution you can just type from terminal:
  sudo apt-get install nvidia-cuda-toolkit

We provide precompiled binaries for the 64-bit version of Windows 7. The folder with the precompiled binaries also contains all required DLLs, and thus no external software packages other than the CUDA drivers mentioned above need to be installed. The software can effectively be run out-of-the-box, as detailed below in section 3.

For Linux, compilation of the source code is required (see detailed instructions in section 2.1). For compiling the source code, any software version equal to or above the CUDA Toolkit software version listed above should suffice.

For possible common runtime errors and solutions see section 5.

2.1 Source code compilation in Linux

- Make sure CMake is installed (http://www.cmake.org/). For Ubuntu distributions, you can simply use the following command:
  sudo apt-get install cmake cmake-gui
- Go to the folder $ROOT_TGMM and create a new folder called "build", where the binaries will subsequently be generated:
  cd $ROOT_TGMM
  mkdir build
  cd build
- In the build folder, execute the following commands:
  cmake -D CMAKE_BUILD_TYPE=RELEASE ../src/

make

The first command locates all libraries (for example, from the CUDA Toolkit) and generates all necessary makefiles to compile the code. The second command calls these makefiles. After executing the second command, you should see messages in the terminal commenting on the compilation progress. If the progress report reaches 100%, the program has compiled successfully. After successful compilation, the following executables should be present:

$ROOT_TGMM/build/nucleiChSvWshedPBC/ProcessStack
$ROOT_TGMM/build/TGMM

You can use cmake-gui or cmake options to change different parameters in the makefiles (for example, final destination folder or CUDA architecture level).

3. Running the TGMM software

We provide a test data set that allows the user to test the code and familiarize themselves with software configuration before applying the code to their own data sets. Currently, 2D + time and 3D + time datasets with 8-bit or 16-bit unsigned integer TIFF stacks are supported as the input data format. The two-dimensional or three-dimensional image data recorded for each time point should be provided as a single TIFF file.

3.1 Configuration file

The file "$ROOT_TGMM\data\TGMM_configFile.txt" serves as a template for the configuration file and contains all parameters required to run the TGMM code. In principle (and for all results presented in this study), only parameters listed under "main parameters" need to be modified for each new experiment. Access to parameters listed under "advanced parameters" is provided as well and is intended for experienced users who wish to experiment further with the code.

Each parameter is accompanied by a description of its functionality (see section "Overview of advanced framework parameters" below for more details). In order to process a new data set, simply copy the configuration text file and adjust parameters as needed.

Important note: Before applying the TGMM software to the test data set, the variables *debugPathPrefix* and *imgFilePattern* in the configuration file need to be adjusted, so the software can locate the image stacks (*imgFilePattern*) and save the results (*debugPathPrefix*).

3.2 Watershed segmentation with persistence-based agglomeration

Windows

In order to generate the hierarchical segmentation for each time point, follow these three steps:

1. Open a Windows command line terminal (run "cmd.exe").
2. Go to the folder "$ROOT_TGMM\build\nucleiChSvWshedPBC\Release".

3. Execute the command:
   ProcessStackBatchMulticore.exe $ROOT_TGMM\data\TGMM_configFile.txt 0 30 12

The program automatically detects how many processing cores are present in the workstation and parallelizes the image segmentation task accordingly. The second and third arguments are the first time point (0 in this case) and the last time point (30 in this case) of the time-lapse image data set. The last argument is the number of cores to use. This is an optional argument and it is set to the maximum number of cores available in the machine by default.

Once processing is complete, new files "$ROOT_TGMM\data\TM?????_timeFused_blending\SPC0_CM0_CM1_CHN00_CHN01.fusedStack_?????_hierarchicalSegmentation_conn3D74_medFilRad2.bin" should have been generated (one for each time point). These binary files store all information required to restore the hierarchical segmentation for each time point. If the binary files were not created, an error occurred during execution of "ProcessStackBatchMulticore.exe" and a corresponding error message is displayed in the terminal.

Linux

In order to generate the hierarchical segmentation for each time point, follow these three steps:

1. Open a terminal.
2. Go to the folder "$ROOT_TGMM/build/nucleiChSvWshedPBC".
3. Execute the command:
   parallel -j8 ./ProcessStack $ROOT_TGMM\data\TGMM_configFile_linux.txt -- {0..30}

The option -j8 indicates how many cores should be used in parallel (in this case 8). The last option, {0..30}, indicates that the program ProcessStack should be executed for time points 0 to 30.

Important note: The command *parallel* is part of the GNU software (http://www.gnu.org/software/parallel/). The program presents an easy interface to call programs in parallel. If this software is not already installed, it can be downloaded from the GNU website or installed from official repositories. For example, in Ubuntu you can simply use the following command: "*sudo apt-get install moreutils*".

Important note: Make sure to use the configuration file TGMM_configFile_linux.txt instead of TGMM_configFile.txt, since the latter contains Windows end-of-line symbols that will lead to a failure during code parsing in Linux. You can also use the tool dos2unix to ensure that any given text file can be used as a config file.

3.3 Bayesian sequential tracking with Gaussian Mixture Models

In order to track cell nuclei and reconstruct cell lineages, follow these three steps (the same instructions are valid for Windows and Linux):

1. Open a Windows command line terminal (run "cmd.exe" in Windows).
2. Go to the folder "$ROOT_TGMM\build\Release"
3. Execute the command:
   TGMM.exe $ROOT_TGMM\data\TGMM_configFile.txt 0 30

The command line will display notifications about the progress of the tracking and segmentation algorithm. Since the hierarchical segmentation results from step 3.2 are saved separately in the ".bin" files, different tracking parameter settings can be tested without the need for recalculating or changing the segmentation data. The output data format of the tracking module is explained in section 4.

3.4 Verifying successful program execution

In order to simplify the verification of successful TGMM software execution, we provide the output for the test data set in "$ROOT_TGMM\data\TGMMruns_testRunToCheckOutput". The output generated by your execution of the program should be very similar to the contents of this folder.

4. Tracking and segmentation output data format

The folder "*debugPathPrefix*\GMEMtracking3D_%date" contains the output of the TGMM run.

The final result can be found in the subfolder "$debugPathPrefix\GMEMtracking3D_%date\XML_finalResult_lht" or "$debugPathPrefix\GMEMtracking3D_%date\XML_finalResult_lht_bckgRm". The latter directory is used if the user applied the background classifier. The output subfolder contains one XML file and one ".svb" file per time point.

The XML file contains the main tracking and segmentation information. Each object is stored under the tag <GaussianMixtureModel> with the following attributes:

- id [integer]: unique id of the object in this particular time point.
- lineage [integer]: unique id of the cell lineage the object belongs to.
- parent [integer]: id of the linked object at the previous time point. Following the chain of "parent" objects reconstructs the track. A value of -1 indicates the birth of a track.
- splitScore [float]: confidence level for the correct tracking of this particular object. A value of 0 indicates very low confidence and a value of 5 indicates very high confidence. Sorting elements by confidence level can guide the user in the data curation process and facilitate more effective editing of the TGMM results (see main text and **Fig. 4**).
- scale [float[3]]: voxel scaling factors along the *x*-, *y*- and *z*-axis.
- nu, beta, alpha [float]: value of the hyper-parameters for the Bayesian GMM.
- m [float[3]]: mean of the Gaussian Mixture (object centroid, in pixels).
- W [float[3][3]]: precision matrix of the Gaussian Mixture (object shape).
- *Prior: same as before, but for prior values obtained from the previous time point. These values are used during the inference procedure.
- svIdx [integer[]]: list of indices of the super-voxels clustered by this Gaussian. Together with the ".svb" file, this information can be used to obtain precise segmentation regions for each object.

The ".svb" file is a binary file in a custom format that can be read with the constructor "supervoxel::supervoxel(istream& is)". Briefly, it contains information about all super-voxels

generated at a particular time point. Thus, using the "svIdx" attribute, the precise segmentation mask for each object can be recovered.

Users can download a set of Matlab scripts in the zip file "readTGMM_XMLoutput.zip" at http://sourceforge.net/projects/tgmm/files/ in order to import the data to Matlab for further analysis. The zip file also includes a README file explaining how to use these scripts.

5. Troubleshooting common runtime errors

1. Program execution starts and one of the following error messages is displayed in the terminal: “no CUDA- capable device is detected” or “CUDA driver version is insufficient for CUDA runtime version”.

   First, confirm that the workstation is equipped with an NVIDIA CUDA-capable graphics card. This is a hardware requirement for running the software. If such a card is installed, you most likely need to update the driver in order to be compatible with CUDA Toolkit 5.5. Go to https://developer.nvidia.com/cuda-downloads and download the current toolkit. The toolkit will also install an updated NVIDIA driver.
2. When you try to run the program from the terminal, a Windows dialog pops up with the following message “The program can't start because msvcp100.dll is missing from your computer”.

   For some reason, the provided DLL from the Microsoft Visual C++ 2010 SP1 Redistributable Package (x64) is not compatible with your windows version. Delete the DLL from the TGMM software folder and go to http://www.microsoft.com/en-us/download/confirmation.aspx?id=13523 to download and install the appropriate version of the Microsoft Visual C++ 2010 SP1 Redistributable Package.
3. Note that the program needs to be called from a “cmd.exe” terminal in Windows. Cygwin or MinGw terminals cause the program to fail.
4. “ProcessStackBatchMulticore.exe” requires paths to be provided using absolute path names. The use of relative path names also causes the program to fail.
5. Note that the parameter “imgFilePattern” in the configuration file “TGMM_configFile.txt” requires the use of forward slashes in path names (since the image library used to read TIFF files follows the Unix convention), whereas the parameter “debugPathPrefix” in the same file requires the use of double backslashes in path names (since backslashes are special characters that are interpreted by the operating system). On Linux systems, always use forward slashes in both parameters.
6. Program execution starts and one of the following error messages is displayed on the terminal: “invalid device symbol in C:/ROOT_TGMM/src/nucle/iChSvWshedPBC/CUDAmedianFilter2D/medianFilter2D.cu at line 230”

The provided binaries were compiled for CUDA compute capability 2.0 or higher. If your NVIDIA GPU card has a lower CUDA compute capability (this information is available from https://developer.nvidia.com/cuda-gpus), the provided binaries will not work. However, you can recompile the source code, which should allow you to run the software. Before compiling, you need to edit the CMakeLists.txt file and modify the line at the top: set (SETBYUSER_CUDA_ARCH sm_20 CACHE STRING "CUDA architecture"). Adjust the flag sm_20 to the appropriate CUDA compute capability of your NVIDIA GPU (for example, sm_13 for CUDA compute capability 1.3).

6. Description of the parameters in the configuration file

6.1 Key configuration parameters: intensity threshold and persistence agglomeration threshold

Across all computational reconstructions and data sets presented in this study, two parameters were modified: the threshold for persistence-based agglomeration of watershed regions (*persistenceSegmentationTau*) and the intensity threshold for defining the background level in each recording (*backgroundThreshold)*. Both of them refer to image properties and are straight-forward to determine by visual inspection of the image volume at a late time point of the time-lapse recording. In general, inspecting late time points is more useful, since (depending on the experiment) intensity levels are often slightly dimmer and cell densities higher. Measurements in this scenario provide a lower bound constraint for both values.

To determine the background threshold, simply inspect a region in the image volume outside the specimen (for example, by using the open-source software ImageJ) and determine the mean intensity level in this background region. It is preferable to be conservative, i.e. to set a lower value so as not to miss cell nuclei, since false negatives can alter the coherence between time points. Moreover, we compute a local threshold for each super-voxel using Otsu's and, thus, even if background regions are included in the foreground estimate, this will not affect the final shape of the super-voxels. The only drawback of a lower background threshold is a small increase in computation time.

To determine the threshold for persistence-based agglomeration of watershed regions (τ), plot the intensity profile across the line connecting two of the dimmest nuclei centroids in the image stack (for example, by using the open-source software ImageJ). The profile should have two peaks (nuclei centroids) and a valley (nuclei borders). The threshold τ should be set to a value smaller than the difference between the intensity values of the peaks and the valley, such that the corresponding nuclei are not merged into a single super-voxel (under-segmentation). In our experience, a value of τ between 5 and 20 tends to be sufficient to compensate for the watershed over-segmentation of noisy regions, without risking merging of dim cell nuclei.

We note that, although care should be taken to set these parameters appropriately, one can obtain close-to-optimal results for a fairly wide range of parameter values. For images with lower signal-to-noise ratio (SNR), such as confocal microscopy images, the value of τ is more critical than the background intensity because watershed regions fragment the image into smaller regions. In contrast, in images with high SNR, such as most light-sheet microscopy images, the intensity background is more relevant because the watershed algorithm already produces super-voxels that follow nucleus morphologies fairly well.

6.2 Overview of advanced framework parameters

In this section, we provide an overview of all advanced framework parameters. Note that these parameters were not changed across the computational reconstructions and data sets presented in this study. To complement the descriptions below, we also provide the default parameter values in the configuration file "$ROOT_TGMM\data\TGMM_configFile.txt", which is included in "Supplementary_Software_1.zip".

betaPercentageOfN_k

Non-negative floating point scalar. *betaPercentageOfN_k* defines the prior probability for the centroid position of a nucleus based on its position in the previous time point. No motion model is used, unless the optical flow module is activated. Thus, if cells are moving fast this parameter should be set close to zero. If cells are moving very little, and the position at time t is a good prediction for the position at time $t + 1$, this parameter should be set to 1 or greater.

nuPercentageOfN_k

Non-negative floating point scalar. Follows the same concept as *betaPercentageOfN_k*, but for shape variation between consecutive time points. If objects change shape rapidly between two consecutive time points this parameter should be set close to zero. If object shapes change very little between time points this parameter should be set to 1 or greater.

alphaPercentage

Floating point scalar. *alphaPercentage* controls the prior probability of death for a track. The more likely nuclei are to disappear from the image or undergo apoptosis, the lower the value of this parameter should be.

maxIterEM

Integer positive number. *maxIterEM* defines the maximum number of iterations of variational inference allowed each time a Gaussian mixture model is fitted. In general, very few rounds are needed (less than 10), since the model is initialized with the solution from the previous time point. Thus, this parameter is implemented as a precaution. The terminal output of TGMM.exe can be used to obtain an estimate of the typical number of iterations needed and *maxIterEM* can then be set accordingly.

tolLikelihood

Floating point positive number. *tolLikelihood* is used as a stopping criterion for variational inference of the Gaussian mixture model. Optimization is stopped if the relative increase in likelihood between two consecutive iterations is less than the value of this parameter. Thus, the lower the value, the more iterations of variational inference are run.

regularizePrecisionMatrixConstants_lambdaMax

Floating point positive number. This parameter provides the maximum allowed value for any of the eigenvalues (in pixels) of the covariance matrix defining each object in the Gaussian mixture model. Thus, the larger the value of *regularizePrecisionMatrixConstants_lambdaMax*, the larger the ellipsoids fitting each nucleus can grow. This parameter is used during regularization of the variational inference results.

regularizePrecisionMatrixConstants_lambdaMin

Floating point positive number. This parameter provides the minimum allowed value for any of the eigenvalues (in pixels) of the covariance matrix defining each object in the Gaussian mixture

model. Thus, the lower the value of *regularizePrecisionMatrixConstants_lambdaMin*, the smaller the ellipsoids fitting each nucleus can shrink. This parameter is used during regularization of the variational inference results.

regularizePrecisionMatrixConstants_maxExcentricity

Floating point positive number. This parameter provides the maximum eccentricity between any two principal axes of the ellipsoid defining a nucleus. This parameter is used during regularization of the variational inference results.

temporalWindowForLogicalRules

Positive integer number. This parameter provides the radius of the temporal window (total window length is [2 × *temporalWindowForLogicalRules* + 1] time points) used to apply spatio-temporal heuristic rules for fixing tracking errors. The larger the value, the more memory is required, since the program will keep the image data of all concerned time points in memory to be able to calculate the features needed to apply the various heuristics.

thrBackgroundDetectorHigh and *thrBackgroundDetectorLow*

Floating point non-negative numbers. These parameters provide the thresholds applied to the results of the background track detector for removing trajectories representing non-nuclei objects. They control the behavior of a hysteresis filter applied over time to the background probability scores of multiple data points belonging to the same lineage. When the program detects a data point with a background probability above *thrBackgroundDetectorHigh* it proceeds with deleting its descendants until the probability falls below *thrBackgroundDetectorLow*. Thus, the higher the value of *thrBackgroundDetectorHigh*, the fewer objects are removed. If *thrBackgroundDetectorHigh* is above 1, then no background track removal is applied.

SLD_lengthTMthr

Non-negative integer number. Any daughter branch that ends within less than *SLD_lengthTMthr* time points after division is considered a spurious over-segmentation event and is deleted.

radiusMedianFilter

Positive integer number. This parameter provides the radius (in pixels) of the median filter applied before the watershed hierarchical segmentation is performed. The noisier the images, the larger this value should be.

minTau

Non-negative floating point value. This parameter provides the minimum value of τ used for the hierarchical segmentation using persistence-based clustering of watershed regions. The higher *minTau*, the larger the minimum super-voxel size that can be generated at the lower level of the hierarchical segmentation. This value should be kept low so as not to compromise the framework's capability to recover from under-segmentation.

conn3D

Values allowed are 6, 28 and 74. This parameter defines the 3D local neighborhood used to run watershed for generating super-voxels. Values of 6 and 28 define traditional 3D neighborhoods, whereas a value of 74 generates cubes of 5 x 5 x 3 around each point to address the anisotropy of the point-spread-function typically encountered in 3D microscopy images.

estimateOpticalFlow

Values allowed are 0, 1 and 2. This parameter activates/deactivates the use of optical flow calculations between time points for the purpose of compensating for large object displacements. A value of 0 deactivates optical flow calculations. A value of 1 indicates that pre-calculated optical flow files are available and can be used to apply local motion displacements between time points. A value of 2 indicates that the program will calculate optical flow on-the-fly, using a routine provided by the user.

maxDistPartitionNeigh

Floating point positive number. It is only used if *estimateOpticalFlow* is equal to 2 and the routine called is the one described in F. Amat *et al*., "Fast and robust optical flow for time-lapse

microscopy using super-voxels" (*Bioinformatics*, 2013). The parameter provides the maximum allowed distance (in pixels) between super-voxels for them to be considered neighbors in the calculation of the optical flow (coherence constraint).

deathThrOpticalFlow

Integer number. If positive, the optical flow module will be activated automatically when the number of deaths at a specific time point is larger than the value of this parameter. Usually, when large motions occur (larger than one nucleus diameter from one time point to the next), many Gaussians in the model disappear, since the solution from the previous time point is not well-suited for initialization of the current time point. Thus, monitoring deaths can be used as a trigger to activate optical flow only when needed.

minNucleiSize

Positive integer number. If the number of voxels belonging to a super-voxel is less than *minNucleiSize* the super-voxel is deleted. This parameter is useful to delete spurious super-voxels representing background intensity.

maxNucleiSize

Positive integer number. This parameter defines the maximum allowed size (in voxels) of a super-voxel after applying Otsu's threshold. If Otsu's threshold generates an object larger than *maxNucleiSize* the threshold is increased until the objet size falls below *maxNucleiSize*.

maxPercentileTrimSV

Floating point number between 0 and 1. This parameter defines the maximum allowed percentage of voxels in a super-voxel belonging to foreground. If Otsu's threshold generates an object larger than *maxPercentileTrimSV* the threshold is increased until the percentage of foreground voxels falls below below *maxPercentileTrimSV*.

conn3DsvTrim

Values allowed are 6, 28 and 74. The final super-voxel generated after trimming the initial super-voxel partition to detect foreground and background is guaranteed to have this connectivity.

maxNumKNNsupervoxel

Positive integer number. This parameter defines the maximum number of nearest neighbors to consider for each super-voxel when building the spatio-temporal graph for tracking. The shorter the nuclear displacement between time points, the lower the parameter value can be.

maxDistKNNsupervoxel

Floating point positive number. This parameter defines the maximum distance (in pixels) to consider for each super-voxel when building the spatio-temporal graph for tracking. The shorter the nuclear displacement between time points, the lower the parameter value can be.

thrSplitScore

Floating point number. If 3D Haar features are used for cell division classification, this parameter sets the threshold for the machine learning classifier to decide whether a cell is dividing or not. The higher the threshold, the fewer divisions are going to be called by the classifier. In order to activate 3D Haar features, the code needs to be recompiled with preprocessor directive CELL_DIVISION_WITH_GPU_3DHAAR.

thrCellDivisionPlaneDistance

Floating point positive number, defining the threshold of a feature for disregarding cell division false positives. The feature calculates the distance (in pixels) between mother cell and the midplane defined by the two daughter cells. If the value is above *thrCellDivisionPlaneDistance*, the cell division is considered a false positive and the linkage between mother and furthest daughter is removed. The default value of 3.2 was determined empirically from a small training set to maximize precision while maintaining a high recall of true cell divisions. The lower the value, the lower the recall of cell divisions and the higher the precision.

thrCellDivisionWithTemporalWindow

Floating point number between 0 and 1. In order to improve cell division detection accuracy the 3D Haar features have been combined across the temporal window. This parameter sets the threshold for the machine learning classifier to decide whether a cell is dividing or not based on these new set of features. The higher the threshold, the fewer divisions are going to be called by the classifier.