Toward Simple & Scalable 3D Cell Tracking

Mojtaba S. Fazli
Department of Computer Science
The University of Georgia
Athens, GA, USA
mojtaba@uga.edu

Stephen A. Vella
Department of Microbiology
The University of Georgia
Athens, GA, USA
sav28290@uga.edu

Silvia N. J. Moreno
Department of Cellular Biology
The University of Georgia
Athens, GA, USA
smoreno@uga.edu

Gary E. Ward
Dept. of Microbiology and Molecular Genetics
University of Vermont
Burlington, VT, USA
gary.ward@uvm.edu

Shannon P. Quinn
Department of Computer Science
The University of Georgia
Athens, GA, USA
spq@uga.edu

***Abstract*—*Toxoplasma gondii* is an obligate intracellular parasitic that is the causative agent of disseminated toxoplasmosis. Roughly, one third of the worlds population will test positive for *T. gondii*. Its virulence is linked to its lytic cycle and is predicated on its motility and ability to enter and exit nucleated cells; therefore, studies elucidating its mechanism of motility and in particular, its motility patterns in the context of its lytic cycle, are critical to the eventual development of therapeutic strategies. Here, we present an end-to-end computational pipeline for detection and tracking of *T. gondii* cells in 3D videos. Our pipeline consists of four different modules including: *Preprocessing, Sparsification, Cell Detection,* and *Cell Tracking*. By identifying key bottlenecks in object tracking and leveraging domain-specific heuristics, we developed a lightweight and highly parallel tracking procedure. Our results indicate that the parallel version of our pipeline performs remarkably faster than the serial one, and lays the groundwork for a novel, large-scale tracking algorithm.**

Index Terms*—*Cell Detection, Cell Tracking, 3D video Tracking, 3D Microscopic Videos, Large-Scale tracking method, Computer vision, Toxoplasma Gondii

I. Introduction

Toxoplasma gondii is an intracellular parasite that invades host cells, hijacks the cellular machinery and replicates, before egress and subsequent invasion of new host cells. This process of invasion, replication, and egress is the lytic cycle, and is predicated entirely on the parasites motility. *T. gondii* is the causative agent of disseminated toxoplasmosis, which is considered one of the most prolific parasitic infections; more than 1/3 of the worlds population will test positive for toxoplasmosis [1]. For infants, elderly, neonatal fetuses, and individuals with weak or compromised immune systems, *T. gondii* can cause deadly complications [1], [2]. *T. gondii*s virulence and ability to invade host cells is directly linked to its lytic cycle and requires the molecular motility machinery of the parasite; therefore, by studying the motion dynamics of *T. gondii*, we aim to assist in the identification and development of therapeutic countermeasures. By modeling the motion of *T. gondii*, we will be able to answer some underlying questions regarding *T. gondii* biology, including: what are the fundamental motion phenotypes of *T. gondii*?How are they coordinated across the parasites lytic cycle? Are the motion patterns altered under or associated with a particular pharmacological drug pressure? If so, what is their molecular basis? Ultimately, our goal is to assist in identifying targeted therapies to disrupt *T. gondii*s lytic cycle by way of its motility mechanisms.

To study the motion of T. gondii, we need to be able to track and quantify cell motions across the time; only then can we extract and compute the desired statistics of the parasites motility. Previously, we developed a computational pipeline for tracking the cells in 2D space and parameterizing their motion [3]. Building on that tracking and parameterization strategy, we developed an unsupervised method for discovery of latent motion phenotypes of *T. gondii* [4]. However, these studies were conducted with two-dimensional microscopy data; recently, new research has suggested that the motion behavior of *T. gondii* in 3D space differs qualitatively from observed motion in 2D space. Whether this is an artifact of different experimental conditions or a real biological effect is unclear. Nevertheless, using experimental conditions that more closely resemble in vivo circumstances–i.e., parasite motilities in 3D space–is always preferable and may open new horizons to the researchers in this field.

From one point of view, studying the cell motion in 3D space may help researchers avoid some shortcomings in 2D, like the occlusion issue that frequently happens in 2D videos. On the other hand, 3D datasets introduce numerous additional challenges, not the least of which is the significantly larger quantity of data. Not only does the additional data necessitate increased computation, but the degrees of freedom in an object tracking setting also increase, further magnifying computational complexity. Ultimately, it seems that studying 3D video microscopy is more beneficial as it more closely

resembles reality. But to fully leverage this modality, we must investigate new computational approaches that scale appropriately with the data.Here, we propose to build on conceptual advances from our previous work in parallel and distributed analysis of 4D functional MRI brain scans [5], [10].

Previously, researchers used some conventional methods for tracking without significant knowledge of computer vision: *Physical Tracking Method*, which means moving the sample automatically to keep one individual cell in focus [11], [12], is one of those kind of methods. But there are some limitations for using this method, including the requirement of a specific experimental setup, and its restriction on observing multiple cells at a time. In another method [13], the researchers increased the z range and then used image cross-correlations, compared the observed patterns with a library of reference images, and assigned a z position based on the identity of the best match. Both methods use only first-order image features to perform tracking and are thus susceptible to even mild sources of noise or perturbation.

Several methods and papers used 3D object tracking in practice. Some of them relies on Human Computer Intraction concepts [14], and some others work based on localizing dynamic objects and estimating the camera ego-motion in 3D space [15]–[17]. But, it is essential to distinguish between them and *3D cell tracking* that we discuss in this research. The data and the standards on cell tracking videos are entirely different from the videos that are used for 3D object tracking.

In this study, we developed a computational pipeline for tracking *T. gondii* parasites in 3D microscopy videos. Our pipeline consists of 4 different modules including : *Preprocessing, Sparsification, Cell Detection,* and Cell Tracking. Our overall objective was the development of a lightweight and scalable tracking method for rigid objects (such as cells) that is also openly available. We conclude with performance comparisons between serial and parallel versions of our initial pipeline.

II. Method

A. Data

In this study, our dataset consists of two subsets with a total size of $4GB$ as toy datasets for our future analysis. The first data set includes 63 frames in which we have 41 slices of 500x502 images. The second one includes the same number of frames, slices, and image size. This data is stored in raw format as RGB TIFF images indexed by $z-axis$ (depth) and $t-axis$ (time).

B. Data Acquisition

This data is captured using a PlanApo 20x objective $(NA = 0.75)$ on a preheated Nikon Eclipse $TE300$ epifluorescence microscope. Time-lapse stacks were captured using an $iXon$ 885 $EMCCD$ camera (Andor Technology, Belfast, Ireland) cooled to -70^{o}C and driven by NIS Elements software (Nikon Instruments, Melville, NY) as part of related research by Dr. Gary Ward [6]. The camera was set to frame transfer sensor mode, with a vertical pixel shift speed of 1.0 μs, vertical clock voltage amplitude of $+1$, readout speed of 35 MHz, conversion gain of 3.8x, EM gain setting of 3 and 22 binning. The $z-slices$ were imaged with an exposure time of $16ms$; stacks consisted of 41 $z-slices$ spaced 1 μm apart for a total x, y, z, t imaging volume of 402 μm x 401 μm x 40 μm for 67 stacks in $60s$. All experiments were completed within 80 min of harvesting the parasites, and control (i.e., untreated) samples were assayed both at the very beginning and end of each experiment to ensure imaging and parasite conditions remained constant. [6]

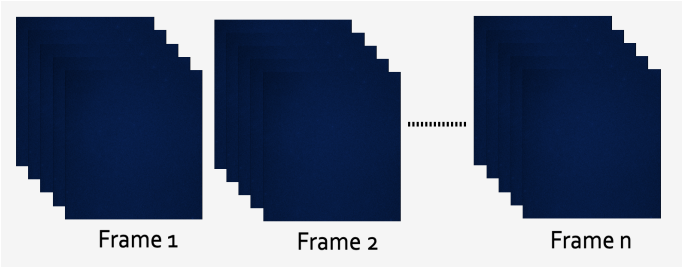

Fig. 1. *Arrangement of the 3D microscopic video slices in different frames across the video.*

C. Software

We implemented our pipeline using Python 3.6 and associated scientific computing libraries (NumPy, SciPy, scikit-learn [18], matplotlib). The core of our detection and tracking algorithm used a combination of tools available in the OpenCV 3.1 [19] computer vision library. For parallelization and multiprocessing, we used the joblib backend and multiprocessing library of python. The full code for our pipeline is available under the MIT open source license at https://github.com/ quinngroup/toxoplasma-3DTracking .

III. Computational Pipeline

For cell detection and tracking, first we created a serial pipeline as a baseline. Due to the large size of our data, we rewrote our code to exploit parallelism in our tracking procedures, recognizing the bottlenecks in our framework and optimizing for so-called embarrassingly parallel submodules. Finally, by parallelizing the bottlenecks, we achieved a fast and scalable platform that can robustly detect and track the cells across a 3D microscopic video.

A. Problem Definition

Recording cell motilities at relevant spatial and temporal scales is a crucial step towards understanding the motion model of *T. gondii*. In 3D microscopic videos, in each frame,

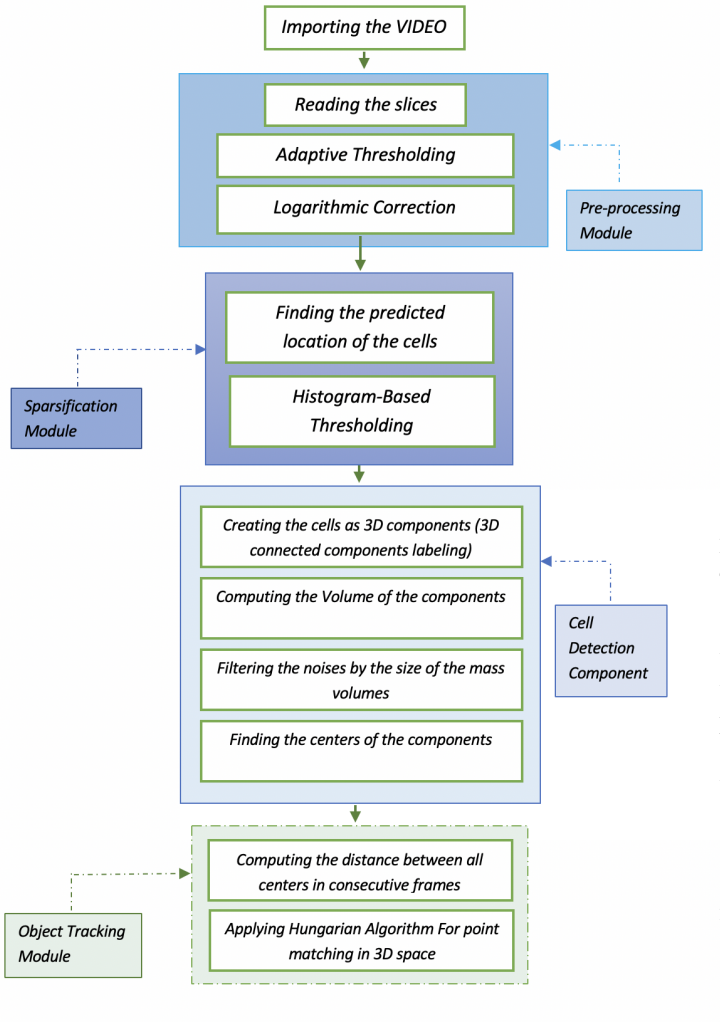

Fig. 2. *Serial version of our 3D computational detection and tracking model.*

we are dealing with a set of slices that determine the depth of the image. The microscopes camera achieves a high-speed acquisition of the slices across different depths, though at the expense of quality, as the images are very noisy. As a consequence, the detection and tracking the cells inside these noisy images is a difficult task.

Figure 1, illustrates the arrangement of the 3D microscopic video slices across time. As it shows, slices are the 2D images that are the cross sections of the depth in 3D video. Each frame contains several slices. Our first step is to detect the partial spatial locations of the cells in each slice of a single frames, then track those particles across time.

B. Model

Our serial computational pipeline is depicted in 2. As it shows, we import all the video slices as TIFF image files,

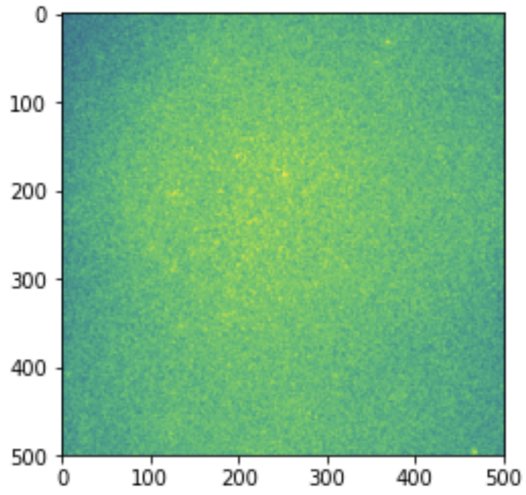

Fig. 3. *A sample slice of the 3D microscopic data set. The artifact of the dark corners is evident in the image.*

then we convert them into numpy arrays. Our next step is to apply some preprocessing on our data. After that, our pipeline makes a sparse matrix, detects the cells in 3D space, and finally tracks the objects by minimizing the cost between the points in consecutive frames.

In the next section, we explain the following four components of our pipeline:

- Pre-processing
- Sparsification
- Cell Detection
- Cell Tracking

C. Pre-Processing Module

In our dataset each frame consists of 41 z-slices, and each slice dimension is 500x502. Our first dataset we contained 63 frames. Thus, in this case we are dealing with a 4D matrix of images with the following dimensions:

$$dim(f(x, y, z, t)) = 63 * 41 * 500 * 502 \quad (1)$$

In our datasets, the last three dimensions of formula (1) are constant, but the number of frames varies among different videos. As it mentioned before, the 3D microscopic images are very noisy. So, the first challenging point is to determine the spatial dimensions of the cells in each frame. A sample slice is illustrated in Figure 3. It contains noise known as salt and pepper. Moreover, the fast imaging speed needed to record at different levels usually creates the artifact of dark corners. This artifact is clear on the Figure 3, where the intensity of the points in the corners of the image is smaller than the intensity of the points in the center of the image.

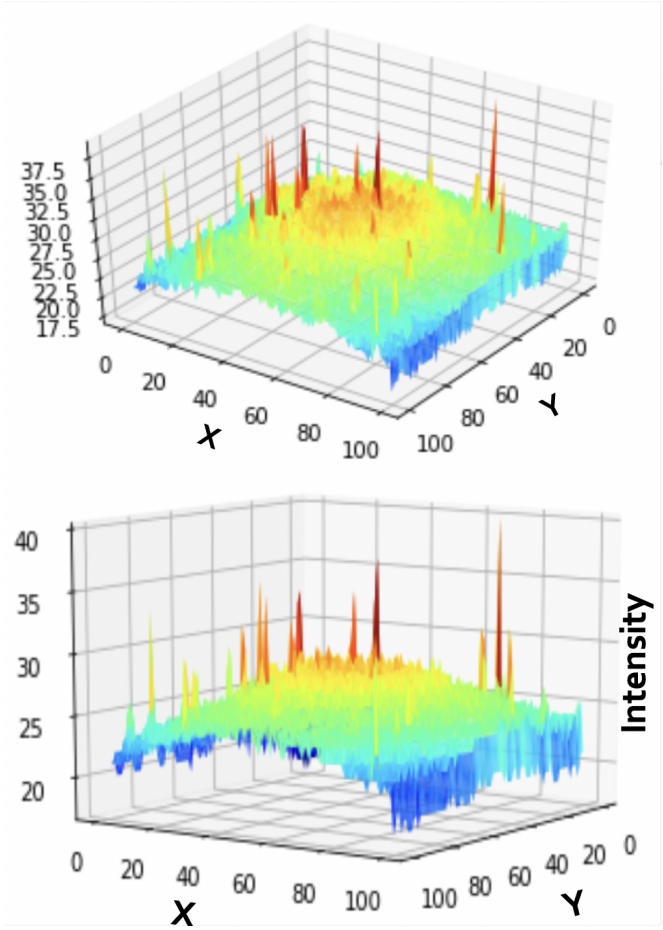

Fig. 4. *3D intensity map of a sample slice image. As the plots indicate, the dark corners make the inverse bowl shape artifact on 3D intensity maps of the images.*

For a better understanding of the effect of the artifacts in our tracking model, we demonstrated the intensity map of the same slice in Figure 4. As it is illustrated, the spikes in intensity maps show the estimated location of cells of that slice. The locations of these spikes can be used for extracting the cell positions. Later, we use this information to threshold and filter the background noise. But the artifact of the dark corners makes the thresholding very challenging.

Simply selecting an inappropriately large threshold value may miss the objects in the corner of each image. Conversely, by choosing a small value for thresholding, we will not eliminate the noise in center of our images that can make the cell detection and tracking very difficult. Thus, we need to have some preprocessing on our images before thresholding. Therefore, we tried different types of preprocessing. Among them, two methods were satisfactory to some extent.

First, we tried smoothing and then sharpening the edges. the result on a sample slice is shown in figure 5.

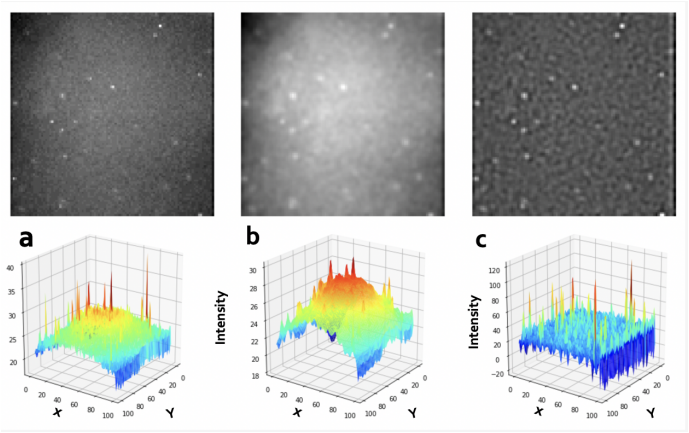

Fig. 5. *First preprocessing method on images to remove the artifact of the dark corners .(a) normal image and its 3D intensity map. (b)The smoothed version of the image and its intensity map. (c) the sharpened version of image in part (b) and it's 3D intensity map.*

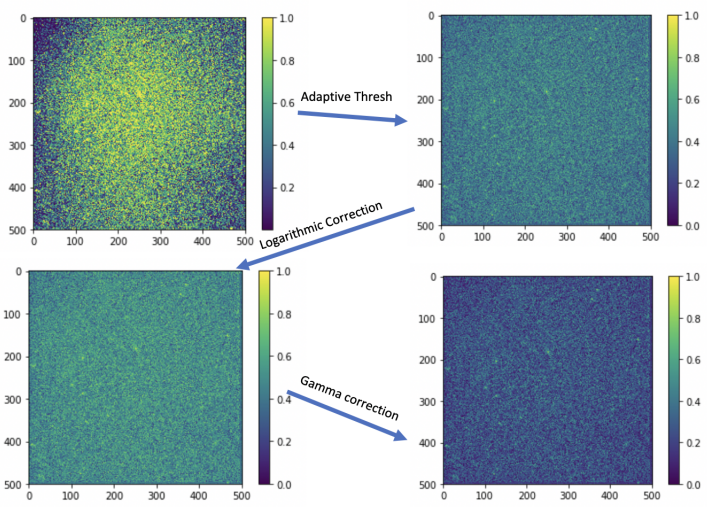

Fig. 6. *Filtering steps of alternate solution to decrease the destructive dark corners artifact.*

This method works well on reducing the effect of the dark corners; however, after checking the results on the thresholding module, we found that we may have missed some objects across the frames, and as a consequence, it may cause some inconsistency in our tracking results. Thus, we reviewed the alternative way of preprocessing as illustrated in Figure 6.

As it is indicated in Figure 6: first we applied the adaptive thresholding on the original image, then we applied the logarithmic correction on that. In the next step, we applied gamma corrections on the resulting image. In turn, we created three images with less pronounced dark corners. To select the best possible result among them, we have plotted their intensity map before and after thresholding. We observed that the final result after applying logarithmic correction works best. The results are illustrated in Figure 7.

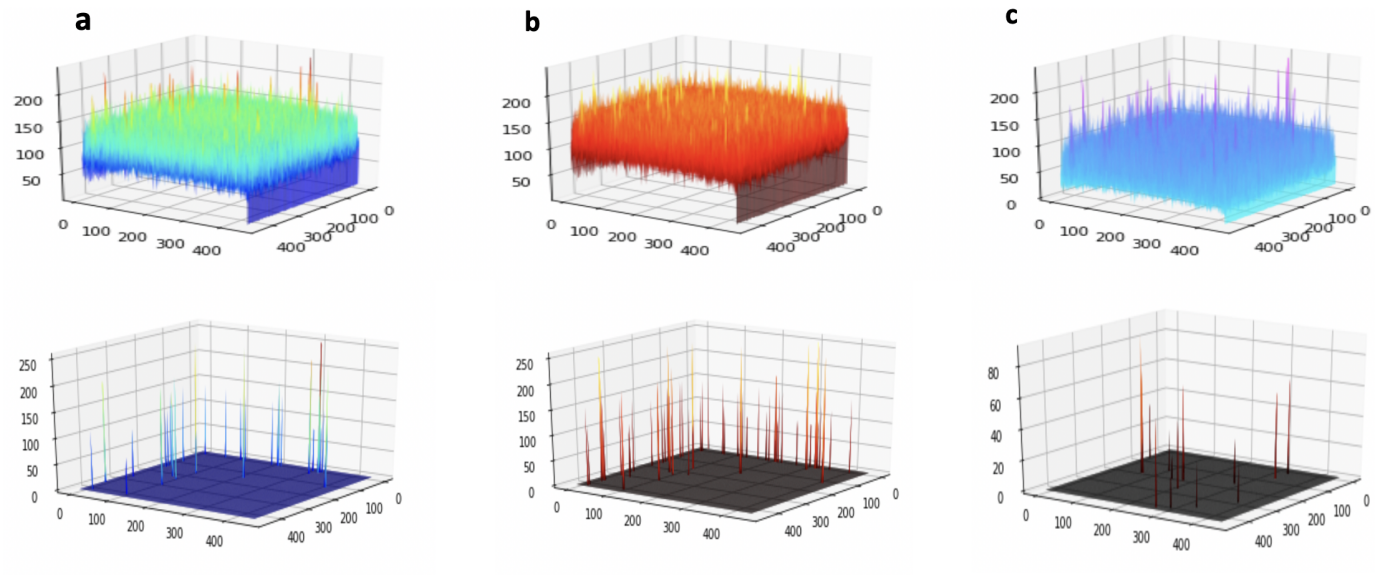

Fig. 7. *(a) top: The result of applying adaptive thresholding on a sample slice image, bottom: the thresholding result of that (b) top :adaptive thresholding + logarithmic correction + gamma enhancement result bottom: the thresholding result of that (c) top :adaptive thresholding + logarithmic correction only. bottom: the thresholding result of that*

D. Sparsification Module

After applying the preprocessing steps, its necessary to find the estimated location of each object in each slice. To do that, first we applied thresholding on the intensity level of the images. As a result, now we have a sparse matrix instead of a huge matrix of data which we illustrated in formula (1). After thresholding, we found that more than 98% of the resulting matrix is sparse where x and y are the spatial locations and the slice number indicates the z coordinate of an object. This way of thresholding creates a location map of potential objects. Figure 8 shows a thresholding result on a sample slice of a frame in our video. Our threshold is purposely kept light, so our false negative rate is minimized or even zeroed out, and we are left with object candidates to be processed further in the next module.

In next step, our pipeline determines the location of the cells pieces across different $z-$slices. Thus, by finding the spikes, we identify the spatial information of each object in a frame, and as a consequence we can reconstruct a 3D representation of the cells. Figure 9 shows a 3D sample frame reconstructed through our pipeline.

E. Cell Detection Module

In previous module after applying preprocessing and thresholding, we shown that how our computational pipeline will extract a meaningful spatial map of the different objects in each separate frame. But these are still a set of discrete points (particles of the cells). To be able to track the cells, first we need to have the unified objects rather than the some independent points in each z-slice. Then we can easily extract the centroids of the objects in the different frames for tracking them across the video.

Therefore, we applied 3D component labeling on our 3D sparse matrix across the video to form the components as cells. As a result, by creating a black and white 3D matrix of the points extracted in our previous step and labeling the closest local points in each frame, we have a labeled matrix (with similar shape of the original matrix) which gives us a component map of the objects in a frame of the video.

In next step we created a 3D representation of all the components across the video (all frames that we have) and compared them with our ground truth. The results are shown in Figure 10. The comparison indicates that our results are a super class of the ground truth. In particular, there are two main differences between our results and the ground truth. First, we need to show the center of each component instead of the whole component. Then, we need to remove the false positives that are a side effect of the thresholding, especially in the middle of our images.

- For finding the center of the components, we assumed a whole component as a mass and computed the center of the mass for each 3D component (cells)
- For removing the false positives, we computed the volume of each mass (components) and if that volume was less than a small threshold we removed that

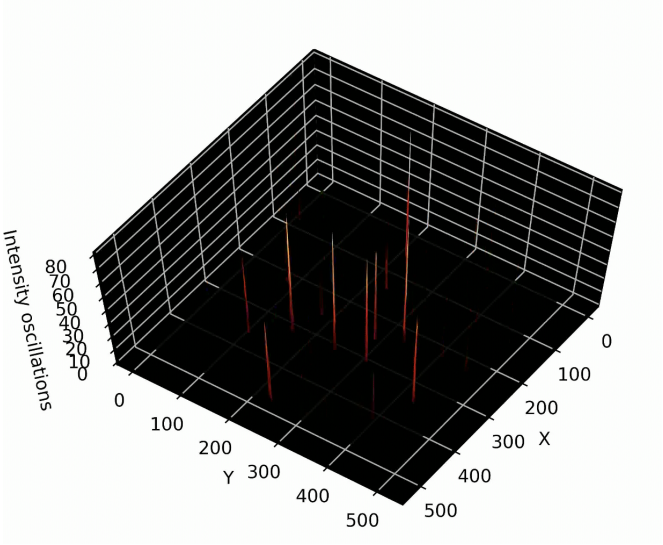

Fig. 8. *First pre-processing method on images to remove the artifact of the dark corners .(a) normal image and its 3D intensity map. (b)The smoothed version of the image and its intensity map. (c) the sharpened version of image in part (b).*

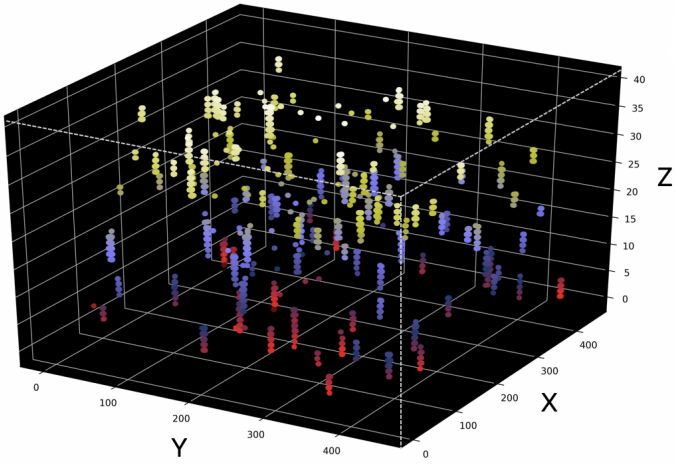

Fig. 9. *3D representation of detected cell particles in different slices of a single frame. The colors show the z axis value (depth).*

component from our list.

In next step, we detected the cell particles, unified them as a 3D mass and computed the centers of the masses as discussed above and the results are shown in Figure 11. Finally, we removed the false positives by filtering the very small particles, and the result is compared with the ground truth in Figure 12. This figure proves the reliability and consistency of our cell detection module.

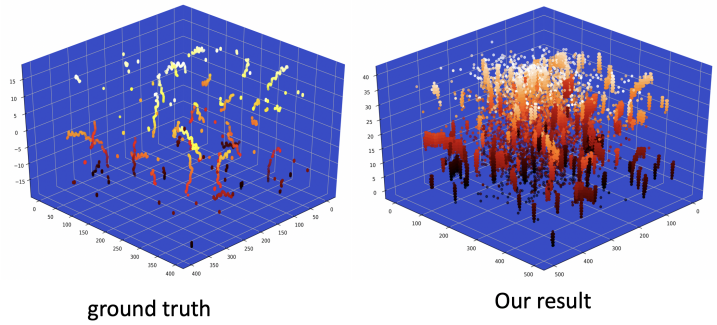

Fig. 10. *Comparing the ground truth(left plot) with the detected components using our pipeline(right) .*

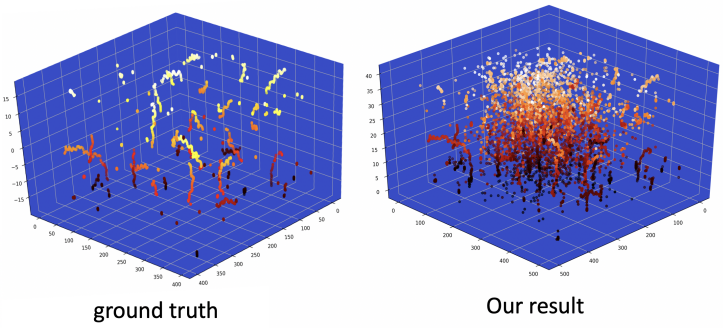

Fig. 11. *Comparing the ground truth(left plot) with the detected components centroids using our pipeline(right) .*

F. Tracking Module

After extracting all the centroids of the masses, we have a jagged list of the centers, meaning that the number of the center points in each frame is slightly varied compared to other frames. This variation is the result of thresholding that makes the tracking part trickier. Consequently, we cannot quickly match the points between the consecutive frames. To do that we need to use the Hungarian algorithm between the points in the current frame and the one in our last selected objects positions.

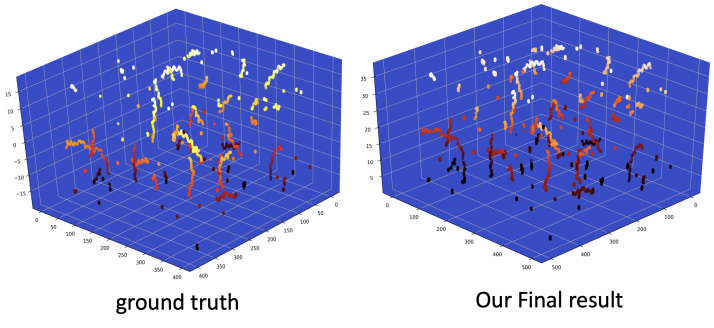

Fig. 12. *Comparing the ground truth (left plot) with the detected components centroids after noise removal process using our pipeline (right) .*

The Hungarian algorithm is a combinatorial optimization algorithm. This method solves the assignment problem in polynomial time and primal-dual methods. The original algorithm is presented by Harold Kuhn in 1955 [7]. We used the sklearn version of this algorithm [9]. This method is also known as Munkers or Kuhn-Munkers algorithm [8]. The original problem is described as a minimum cost matching. Suppose that, there exist a cost matrix which we call it $COST$, in this case, $COST[a, b]$ is the cost of matching worker a to job of b. The goal of the algorithm is to find an assignment map that links the workers to the jobs with minimal possible cost.

Now, lets assume a boolean matrix of $X[a, b]$ that has the value of $True$ or 1 iff the row of a is assigned to the column of b. Then the optimal assignment has the cost of [9] :

$$min \sum_{a} \sum_{b} COST_{a,b} X_{a,b} \tag{2}$$

In our case the cost in formula (2) is the distance between the points. Thus the matching would be optimal iff we have the minimum distance between the points.

1) The problem of local assignment: In our tracking problem, we check the cost between the current frame points and the previous tracked points. The more natural way is to compute the distance between a cell position in the current frame, and all other tracked points at an earlier frames. In this case, we may face an incorrect assignment. For better understanding, consider Figure 13. In this figure, the blue points are the current frame points, and the orange ones belong to the previous frame. Assume that we start from point C, for assigning this point to the best previous matched point we need to compute the distance between that point and all other previously selected ones. As Figure 13 shows, the cost between C and A is smaller than the cost between C and B. Thus the point A will be selected and since A is selected by C the point B will be chosen by D. However, the point A is closer to D rather than C. So, this algorithm fails in locality issues.

To cope with this issue, we are not allowed to only look at the minimum distance between one point and the previously selected points. But also, we have to measure the distance between all the previously selected points with respect to all the cell positions in the current frame, and define it as our cost function. Then using the Hungarian algorithm, our pipeline assigns all the corresponding points to each other and creates the trajectories set. Figure 13-right shows the correct assignments using our pipeline.

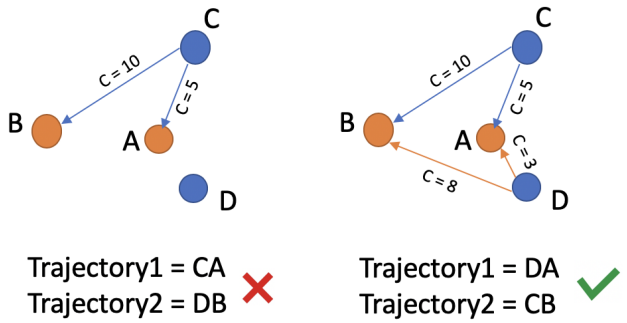

Fig. 13. ***Right)*** *wrong assignment of the points (when we compute the cost function between only one point in the current frame and all other points from previously selected points; and assign the minimum one)* ***Left)*** *The proper assignment by computing the pairwise cost between all the points in the correct frame and all the previously selected points.*

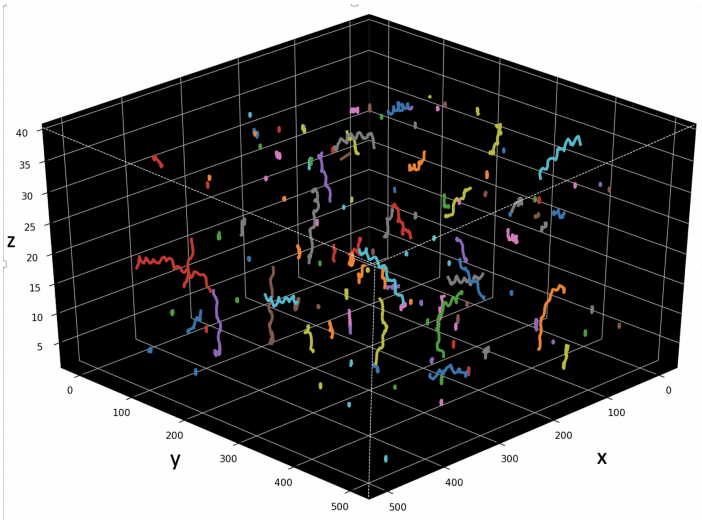

Fig. 14. *3D tracking results . Each trajectory is colored by different color.*

An essential point in our tracking pipeline resides in the fact that we need to initialize the first elements of our objects with the center points of the elements in frame 0. Then we try to assign the points in current frames with the last chosen points in our objects list. Moreover, since our data is noisy, the number of center points are not fixed in all the frames. As a consequence, after assigning some points, some others may remain unassigned. Thus we need to set a distance threshold on the distance between two points not exceeding the defined value in two consecutive frames. Figure 14 demonstrates the tracking results. Furthermore, After successful tracking, we have the trajectories points. As a results, we are able to analyze the single cell dynamics across the time.
Figure 15 shows some separate trajectories extracted from our tracking module.

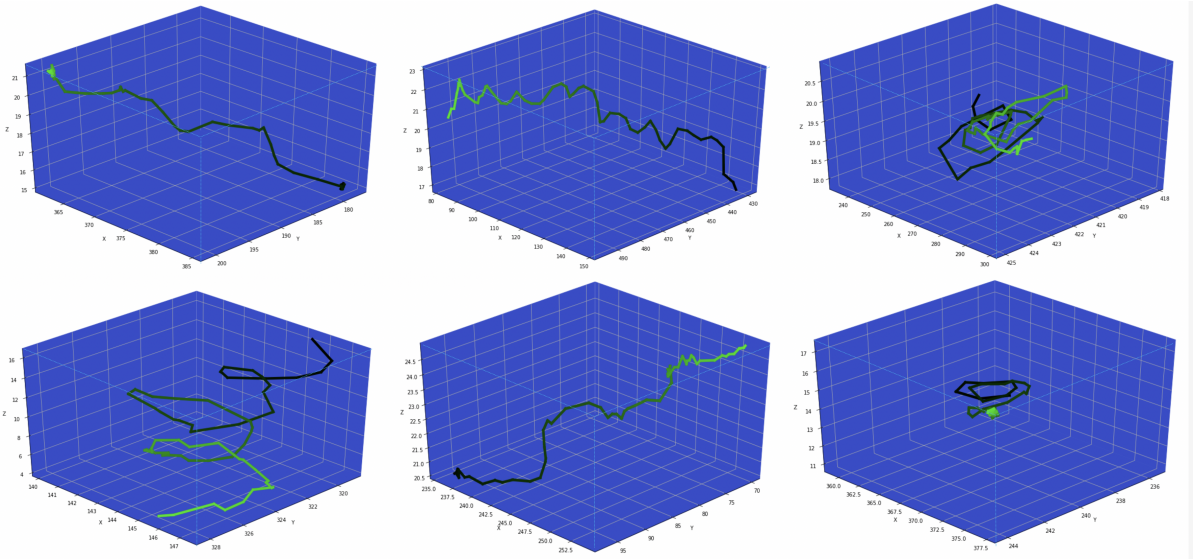

Fig. 15. *4D trajectory representation : the spatial parameters are represented by x, y and z values and the trajectories are colored in such a way that it represents the temporal dimension (frames). The darkest point shows the start point and the lightest one indicates the last frame.*

IV. Extracting the bottlenecks and the parallel model

Although our computational pipeline was designed efficiently, it still suffers from serial bottlenecks, especially because of the size of our data set. Considering that these datasets are only the toy datasets and we will work with the much larger data sets in the future, there is a relevant need of parallelization and distribution of the computations.

In our approach, we repeat most of the steps for multiple slices and multiple frames in different modules. Moreover, most of these computations, before feeding the points into tracking module, are independently computable and can be done separately. To do that, we studied our pipeline components and parallelized the bottlenecks and the results are explained in Table 1. For parallelization, we used the powerful joblib backend and multiprocess library of Python. Generally we assumed the data inside each frame as data chunks and run the parallel processing on them instead of the whole data set. For first step, we used a workstation with Intel $Corei9$ processor with 36 cores and $128GB$ of fast RAM.

To perform multiprocessing in our modules, first, we identified the bottlenecks in our framework components which are listed as follows :

- In preprocessing module, where we efficiently parallelized the preprocessing modules across the frames instead of the whole data set.
- In sparsification module, where we computed the thresholding over the frames instead of the whole data.
- In cell detection component, where we used parallelizing on ”3D connected component” module and used different cores for computing the connected component labeling for each frame
- In cell detection component where we used parallelizing on denoising computation module.
- In cell detection component where we used parallelizing on computing the centers of the masses.

Finally, we developed a parallel version of our code and compared the performance of both parallel and serial versions of the pipeline. The performance improvement is shown in Table 1.

TABLE I
Performance Comparison Table (Serial Vs Parallel)

	Dataset #1	***Dataset #2***
Size	**1.97 GB**	**2.02 GB**
Wall time (serial)	**3 min and 5 sec.**	**3 min and 8 sec.**
Wall time (parallel)	**17.43 Seconds**	**18.19 Seconds**

V. Conclusion and Discussion

In this research, we developed a computational pipeline for cell tracking and detection in 3D microscopic videos. We discussed the challenges unique to 3D microscopic video processing and cell tracking. As we discussed, the most prevalent challenge of our work relies in the large size of our datasets. While our toy dataset for this work was only $4GB$, we are anticipating this framework to be used for

datasets in the terabyte scale next. Thus, we determined the bottlenecks of our pipeline and incorporated parallelism into our implementation. We developed a parallel version of our framework using joblib and multiprocess libraries of Python. Comparison between the ground truth and our final result verified the accuracy of our framework. Moreover, the parallel version shows an order of magnitude improvement in runtime over the serial version. As this is only an initial implementation, we anticipate additional performance improvements we refine our methods.

In the future we plan to concentrate more on two points. First of all, improving the computational performance of our pipeline and its scalability using other distributed systems solutions like dask or Apache Spark, broadening the applicability and generalizability of our framework without sacrificing its ease of use. Second, we hope to use more robust and adaptive thresholding methods to remove sources of noise in the images, such as generative adversarial networks (GANs) to learn noise distributions and perform deconvolutions.

ACKNOWLEDGEMENTS

The authors acknowledge Dr. Kyle Johnsen and the Georgia Informatics Institutes for providing the computational resources for this project. This work is supported in part by the GII Fellowship. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X Pascal GPU used for this research. We acknowledge and thank Google for a generous Research Grant for use on their Compute Platform. We acknowledge partial support from the NSF Advances in Biological Informatics (ABI) under award number 1458766.

REFERENCES

[1] Saadatnia, Geita, and Majid Golkar. "A review on human toxoplasmosis." Scandinavian journal of infectious diseases 44.11 (2012): 805-814.

[2] Dubey, Jitender Prakask. Toxoplasmosis of animals and humans. CRC press, 2016.

[3] Fazli, M.S., Vella, S.A., Moreno, S.N. and Quinn, S., 2017. Computational motility tracking of calcium dynamics in toxoplasma gondii. arXiv preprint arXiv:1708.01871.

[4] Fazli, M.S., Velia, S.A., Moreno, S.N. and Quinn, S., 2018, April. Unsupervised discovery of toxoplasma gondii motility phenotypes. In Biomedical Imaging (ISBI 2018), 2018 IEEE 15th International Symposium on (pp. 981-984). IEEE.

[5] Li, Xiang, Milad Makkie, Binbin Lin, Mojtaba Sedigh Fazli, Ian Davidson, Jieping Ye, Tianming Liu, and Shannon Quinn. "Scalable fast rank-1 dictionary learning for fMRI big data analysis." In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 511-519. ACM, 2016.

[6] Leung, Jacqueline M., et al. "Disruption of TgPHIL1 alters specific parameters of Toxoplasma gondii motility measured in a quantitative, three-dimensional live motility assay." PloS one9.1 (2014): e85763.

[7] Bruff, Derek. "The assignment problem and the hungarian method." Notes for Math 20.29-47 (2005): 5.

[8] Frank, Andrs. "On Kuhn's Hungarian methoda tribute from Hungary." Naval Research Logistics (NRL) 52.1 (2005): 2-5

[9] https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.optimize.linear_sum_assignment.html

[10] Makkie, M., Li, X., Lin, B., Ye, J., Fazli, M.S., Liu, T. and Quinn, S., 2017. Distributed rank-1 dictionary learning: Towards fast and scalable solutions for fMRI big data analytics. arXiv preprint arXiv:1708.02638.

[11] Liu, Bin, et al. "Helical motion of the cell body enhances Caulobacter crescentus motility." Proceedings of the National Academy of Sciences 111.31 (2014): 11252-11256.

[12] Berg, Howard C. "How to track bacteria." Review of Scientific Instruments 42.6 (1971): 868-871.

[13] Taute, K. M., et al. "High-throughput 3D tracking of bacteria on a standard phase contrast microscope." Nature communications 6 (2015): 8776.

[14] Melax, Stan, Leonid Keselman, and Sterling Orsten. "Dynamics based 3D skeletal hand tracking." Proceedings of Graphics Interface 2013. Canadian Information Processing Society,

[15] Shen, Shaojie. "Stereo Vision-based Semantic 3D Object and Ego-motion Tracking for Autonomous Driving." arXiv preprint arXiv:1807.02062 (2018).

[16] Kehl, Wadim, et al. "Deep learning of local RGB-D patches for 3D object detection and 6D pose estimation." European Conference on Computer Vision. Springer, Cham, 2016.

[17] Tejani, Alykhan, et al. "Latent-class hough forests for 3D object detection and pose estimation." European Conference on Computer Vision. Springer, Cham, 2014.

[18] Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." Journal of machine learning research 12.Oct (2011): 2825-2830.

[19] Bradski, Gary. "The opencv library (2000)." Dr. Dobbs Journal of Software Tools (2000).