UiT The Arctic University of Norway

Faculty of Science and Technology
Department of Computer Science

# Presenting CODS (Cell Organelle Dynamic Simulation)

—

**Aaron Celeste**

*INF-3990 Master's Thesis in Computer Science – May 2023*

# Abstract

The interactions between organelles within living cells have an effect on the health of the organism [1]. One way to study this behavior is to develop algorithms which can automatically detect and track objects in microscopy video data. A large amount of data is needed to train algorithms to identify particular behaviors in subcellular machinery. If the case of segmentation is considered, videos of these organelles are difficult to label because of the inability to locate objects with precision in microscopy videos. This thesis introduces CODS (Cell Organelle Dynamic Simulation), a solution for generating as much automatically labeled data as needed for use in several varieties of machine learning. CODS is able to work in parallel to produce dynamic complex simulations of mitochondria performing behaviors such as fission, fusion, and kiss-and-run in various behavior profiles. This lays the groundwork to easily build further complex interactions and to add organelle types to perform further development and study. CODS offers the ability to easily manipulate dozens of parameters relating to the type, number, and behavior of organelles desired. This includes the parameters of the microscope being simulated, providing the opportunity to produce data which works to mimic a variety of situations. I discuss the myriad uses of the CODS simulation engine and show that output from CODS is good enough to train a video segmentation algorithm to detect and track mitochondria through time in real microscopy data.

# Acknowledgements

I have gratitude for my supervisors being there when I needed them during the important moments and giving me enough room to explore and grow on my own as well. Much thanks to Dilip, Krishna, Åsa and Rohit.

Special thanks to Rohit for me helping organize a stronger argument for the weight of the contribution of CODS.

Thanks Suyog for many days of technical council in times of struggle.

Thanks Ida for contributing such great microscopy video.

Thanks to Himanshu, Rohit, Iqra and Nirwan for letting me see that my work was useful by doing new experiments on my data.

# Contents

# List of Figures

# List of Tables

# 1 Introduction

Science is the tedious proving and disproving of theories. Sometimes performing this work requires decades of experience in an exceedingly narrow field because that's what it takes to make sense of collected data. Computers have ushered in an explosion in the pace of science. This is not only due to the instant communication of teams around the world but also because computers can capture some aspects of top class researcher's expertise into routines repeatable by anyone with a computer. Computer science has become an integral part of every field of science, including biology. The recommendation for biologists today is to learn at least one programming language among a myriad of other tools [2] [3] [4]. One of the fundamental tools of the biologist, the microscope, is looking more and more like a high tech computer.

Cells are the building blocks of much of life on earth, including us humans. Microscopes allow us to look deep into the guts of these elementary particles. Making sense of this chaotic world is no easy task. Computers have given humanity the ability to perform complex analyses on this data. Experts in their field can contribute to fine-tuning computer models which perform accurate analytics on new data even when amateurs are the ones collecting the data. Studying the behavior of cells and how they work can produce important insight for medical science. These insights can happen quicker if we can give the power of complex microscopy analysis to anyone with a microscope. Cells contain within them small structures which have an effect on the cells. Studying these can increase our understanding of their effect on health. Observing this subcellular machinery is difficult. If this could be automated, the research it enables

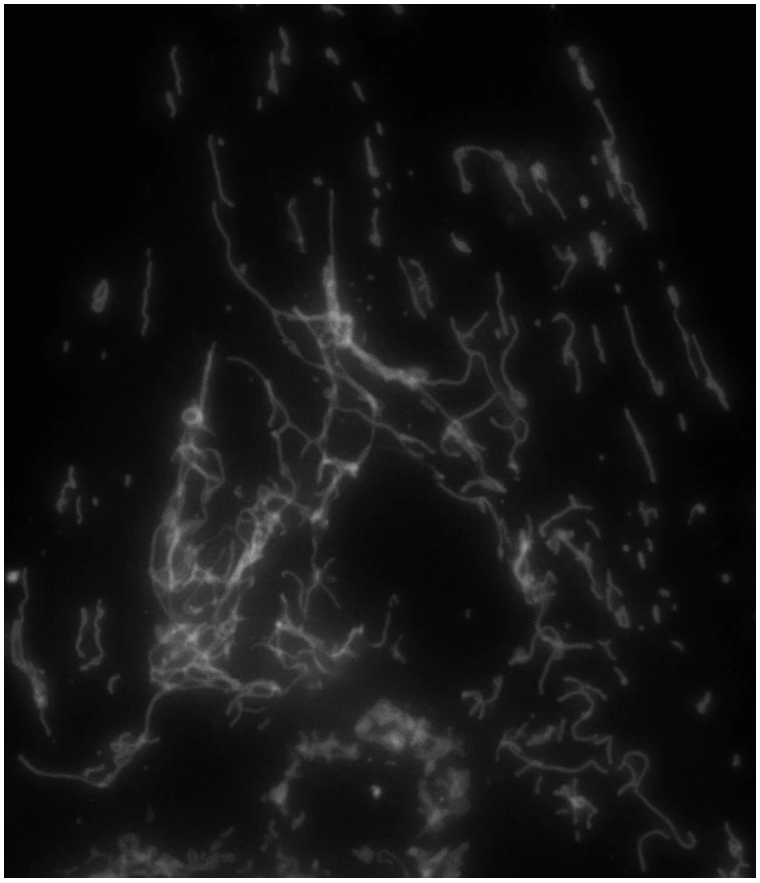

**Figure 1.1:** Common Image of Mitochondria. The complexity seen in this fluorescence microscopy image is the norm in this kind of imagery. This image came from this dataset [5]

might lead to treatments for diseases. Using more powerful microscopes to look deeper and deeper into the building blocks of life comes with side effects such as noise, distortion, and the lack of contrast between different structures. One example of an important player within cells is mitochondria which have been shown to exhibit distinct behavior patterns which seem to be related to the health of the organism [1]. There is a lot to learn about the way the behavior of mitochondria is affected by their environment and how their behavior has an effect on health. One challenge to training a computer model to observe mitochondria is their high numbers and twisting overlapping shapes. There are many challenges to overcome when imaging the building blocks of cells. A few of these challenges will be laid out in the next several paragraphs.

*Complex networks.* Mitochondria often fuse together into complex networks and chains as can be seen in Figure 1.1. When video data of mitochondria is cap-

tured, the picture is blurred depending on the mitochondria's distance from the lens of the microscope, and it is difficult for experts to say with confidence where one mitochondria stops and another begins. This is further made complex if there is a dense region where many tangled mitochondria are present.

*Time consuming.* It is expensive to pay experts to painstakingly review microscope footage and make notes about the mitochondria. One way to automate the process of taking notes about mitochondria videos with a algorithm would be training a machine learning model to do it. This requires lots of training data, so experts will still have to be paid and time will be needed to produce this training data which could include labels like text based, counting events, or segmentation. And for each new type of object, new models will need to be trained to recognize those objects. And for each model, experts will need to be summoned anew to create new annotated data. It's also time consuming to simply collect the microscopy data. Even without the annotating process, real microscopy data is expensive to collect on it's own.

*Bad quality data.* These issues are compounded when considering the low resolution of microscopy data and the noise that comes with it. The blurry nature of microscopy data lends itself poorly to segmentation in particular. If a researcher has decided to attempt to create segmentation labels for machine learning by hand, they will find themselves given the creative freedom to draw lines anywhere within the blurry edges of each entity, if individual entities are even identifiable within a tangled mess of them. This variability in how tightly in toward the mitochondria it's segmentation borders are placed by a human is a problem for machine learning. Machine learning performs best with consistent training data and every pixel of variability causes confusion in the results of a model. This problem is amplified when you consider that to get the large amount of training data needed, likely multiple people will be employed to create these data which introduces even more variability in the segmentation.

Is it possible to build a tool that can address these challenges? This thesis work focuses on answering the question: Can such a tool be shown to have potential to improve the way science is done in this field? I present CODS (Complex Organelle Dynamic Simulator), a simulation generator for use in machine learning that solves the above-mentioned problems. The simulations are of a dynamic complex environment composed of subcellular organelles. The motion of these organelles and their interactions with each other over time are modeled in evolving non deterministic output videos. The output of CODS includes videos and their ground truth counterparts along with various files containing specific attributes of the mitochondria such as their behavior and locations. There are several proposed applications for this output. It can be used for several machine learning applications. The main demonstration

discussed in Results (Chapter 7) is video segmentation while other possibilities are discussed in Applications (Chapter 6) include use in training models that can produce natural language descriptions of microscopy videos, or models that can count the number of certain behaviors occurring in videos.

Work has been done previously toward image segmentation using a method involving simulation supervised machine learning [6]. There has been an effort to simulate the movement of vesicles [7]. These two papers are most related to CODS but there is an ongoing body of work outside my university that was important to this research project. The fission and fusion of mitochondria has been studied [8] [9] as well as the effect fission and fusion have on the networks they create [10]. Mitochondria volume has been approximated [11] and it's morphology has been studied [12] and manipulated [13]. Mitochondria's effect on the host cell [14] [9] and effects on the organism [1] are discussed here as well. In it's current form, CODS produces interesting results only regarding mitochondria. Work has been done to get CODS closer to simulating spherical vesicles as well but it's not there yet.

Focusing now on one of the primary possible applications of CODS, if a deep learning model can be trained on video simulations of subcellular machinery performing specific behaviors that they are observed to do in real microscopy videos, It should be possible to use the model to classify these behaviors automatically in videos of real organelles. The goal of this thesis is to test if CODS can be successfully used for this particular application. CODS data will be tested in the following way. It will be attempted to use simulated video of subcellular organelles performing actions to train an algorithm to classify real videos of mitochondria. Designing and creating a system that algorithmically generates data which is good enough to be used to train a machine learning model that successfully classifies the behavior in real microscopy videos would add a powerful tool in the scientific toolbox which could increase the speed at which researchers are able to gain insight about videos of the inner workings of cells.

The contribution of CODS is the movement of mitochondria in generated simulated videos, the graphical user interface, and the detailed files that can be generated recording their attributes and behavior. These will be discussed in turn in the next paragraphs.

*Mitochondria motion.* The type of movement of mitochondria in the output of CODS is diverse. Movement exhibited in the simulations ranges from actions carried out by a single entity to organized group behaviors involving many mitochondria working together. Movements fall under several categories including perturbation and rotation. One foundational element of movement is the translation. Translating a mitochondria involves applying a mathematical

formula to each of it's position defining points. Each of these several points is modified such that the X coordinate of it's position in the video frame is increased by the X direction of translation multiplied by the speed of translation. The Y coordinate is modified in the same way based on the Y direction of translation and the unique translation speed associated with that particular mitochondria. Dozens of parameters which can be supplied via the graphical user interface fine tune the exact specifications desired including the range that the speeds of translations should be chosen from for each mitochondria. Other examples of parameters include the size and number of mitochondria, number of frames, number of pixels, number of videos, which behaviors will be included etc.

*The graphical user interface.* The graphical user interface is a way to adjust dozens of parameters. This allows researchers to manually fine tune the exact sort of videos they want out of the system. These videos may be different depending on the model that they are trying to train. Sometimes long videos or short videos could be required. And it could be that videos are needed which exclude or include certain behaviors. If a researcher is focusing on a domain with shorter mitochondria or more perturbations, this can all be done and more by adjusting parameters using the graphical user interface.

*Recorded data.* With each batch of data generated using CODS, a summary of these parameters are recorded. This is helpful for double checking exactly what the data contains. Also recorded is a playback of the events that took place in the videos. This can be used to cross reference with the activity seen in the video. Both these groups of collected data are nice to have for sanity checks but there's another category of recorded data that's crucial for potential applications of CODS. Currently there's work being done to adapt data produced by CODS toward natural language processing machine learning and an algorithm to predict the behavior of mitochondria interacting in a video. This data can be information about the position of mitochondria over time or differently formatted data about the behavior of mitochondria inside the simulations.

The text of this research project is organized into several parts. The next chapter, Background (Chapter 2), gives recognition to the works that influenced this research project. It also provides a crash course on some important concepts for reading the rest of this text. Special attention will be given in Preliminaries (Chapter 3) to the image generator that CODS uses. Some strategies implemented in getting CODS up and running will be briefly visited in Methodology (Chapter 4) to give some sense of the mechanics of how it works. The behavioral building blocks of the simulation and their real live analogues will be discussed in Example Scenarios (Chapter 5) and following this potential use cases for CODS in Applications (Chapter 6). An overview of the testing proce-

dures followed and their results lives in Results (Chapter 7). This is also where Discussion (Section 7.4) and Conclusion (Chapter 7.5) are where an attempt to neatly wrap everything up will be found.

# 2

# Background

This thesis project has benefited from work done by researchers in biology and computer science stretching back many decades. Some background will be given about breakthroughs which make up the space where this thesis research finds itself. This includes advances in microscopy techniques, genetic engineering, and the development of computer algorithms which approximate real life phenomena. This chapter will also describe some work being done in the field recently which is related to subcellular organelle research in an attempt to give a brief overview of the state of the art and related works.

## 2.1 Fundamentals of Microscopy

The beginning of this section will focus on defining some terms we'll need later. This is also a convenient way to get caught up on the basics of the field. In a microscope, light is collected from a very small area of space and this light is spread out across a larger space to make objects appear larger. This process is called magnification. This can be seen in Figure 2.1 as the yellow lines converging onto a point near the coverslip. This allows people to see things that normally are too small to be seen. This spreading out or magnification of light takes place in the objective of the microscope which is also labeled in Figure 2.1. The amount of magnification a microscope provides is related to several factors, some of which are listed below.

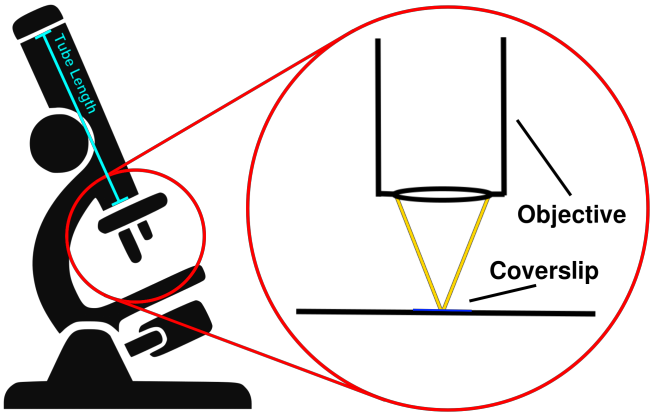


**Figure 2.1:** Microscope. The angle of the photons seen in yellow will change depending on the distance between the objective and coverslip. The black microscope on the left part of this image was downloaded from this website: [15].

- The distance between the objective and the object being observed.
- The diameter of the objective.
- The medium that the light is passing through such as air or oil.

The medium that the light is passing through matters because as light passes from one medium to another, such as when it goes through air and then enters into water, the light changes direction slightly so this effects the angle of travel. This change in the angle of light between different mediums can be determined by a measure called the refractive index which is constant for a particular medium. The distance and size of the objective together determine the angle of travel of the light. In Figure 2.1 if the objective was a closer or further away from the coverslip then the yellow light cone would change angle. These angles are needed to calculate something called the numerical aperture of a microscope. The refractive index of the medium between the object and the objective matters as well as the refractive index of the object itself. Along with the angle of travel of the light and the distance from the object to objective, another factor is the distance that the light travels after the objective, called the tube length. Another thing that effects light traveling from the observed object to the objective is the thickness of the coverslip. A coverslip is a thin rigid transparent device placed on top of the observed object to fix it in place and to provide a flat area for light to change mediums through. This also helps keep the object in a very precise distance from the objective where the microscope is tuned to focus on. The tube length and coverslip are labeled in Figure 2.1.

Cells and the structures inside them can be transparent and therefore have low visible contrast with each other when viewed in the usual manner of bouncing light off them from an external source. In order to capture more clear images of subcellular machinery, it's useful to have a way of viewing the edges where these structures start and stop. What if it was possible to make them glow? Bioluminescence is a phenomenon whereby light emitting particles grown naturally inside organisms allow them to glow. The DNA of glowing Jellyfish has been genetically inserted into cells grown in the lab which causes them to produce a special light emitting dye in subcellular structures like mitochondria and vesicles. This dye is made up of particles called fluorophores and they emit photons of a certain wavelength when an electron drops naturally down from one orbital to a lower more stable orbital. Electrons can be excited up into this high unstable orbital by directing at them photons of a different wavelength. By shining artificial light on these fluorophores, they become visible because the photons produced by these emitters are visible to microscopes. The image of these fluorophores is made clear by introducing a light filter on the microscope which blocks all light except the light which is a wavelength equal to the wavelength expected to be released by the fluorophores. This is called fluorescence microscopy. These emitters switch on and off periodically making them alternate between releasing light and becoming dark. In this review [16], it's described how fluorescing proteins have been inserted in this way which can be customized in various configurations to allow light emitted in almost the entire visible spectrum. This ability to highly customize the light wavelength is useful in experiments when combined with the highly customizability of light filters. This makes it possible to hone in on very specific desired microscopy imaging situations.

Microscopes have trouble capturing images of subcellular machinery with crisp clear edges because these objects are so small that the waves of light used to view them are about as large as the objects themselves. Vesicles and mitochondria can be as thin as a few hundred nanometers or less [17] [12], which is on the low end of the spectrum of visible light. Trying to view these objects directly results in a blurry smudge on the captured image. There are techniques using mathematical and physical tricks to dramatically increase the resolution of these images but they are still blurry. Machine learning works best when the training data is as close as possible to the data which will be encountered in real life. For this reason to train segmentation models for this kind of imagery it's important to use training data which is also blurred. The blurring effecting microscopy data smooths light across pixels according to a PSF. This mathematical function can be used to reverse engineer the blur for use in recreating the same kind of blur artificially in a simulated image. This paper [18] describes a method of using a PSF function to simulate microscopy images of cells. In the paper they discuss how the PSF function is customized according to desired parameters such as the numerical aperture. This is exactly what is done in the

image simulator which CODS uses. This will be discussed in greater detail in the following chapters.

## 2.2 Related Work

Image segmentation is the act of highlighting the pixels in an image which belong to a particular entity. For example to perform image segmentation on a photo of a mitochondria means selecting which pixels make up the mitochondria in the image and not selecting pixels making up the background of the image or other objects shown in the image. This paper [6] presents a static image simulator which has the purpose of training image segmentation models. The simulator creates pairs of images, one black and white ground truth segmented image and one more realistic image which tries to mimic the blurriness and imperfection of microscopy data of real world organelles. This simulator is used to train models to segment mitochondria and vesicles in real world microscopy images. These models were then tested and verified to work on microscopy data which has different wavelengths of emitter photons than the training simulations. It was also discovered that they continued working in data with very densely populated mitochondria networks even though the training data only ever included two mitochondria in one image. The models also performed well under situations with very low contrast between the mitochondria and background. The models also ignored non mitochondria objects in the data correctly even though there were no examples of this in the training data. Transfer learning between microscopes was demonstrated where a model trained with data customized to one type of microscope was able to be successfully generalizable to work with another type of microscope using a small amount of additional data.

Vesicles were included in the still image simulator above and success was seen in using these data also to segment microscopy images of vesicles. Another paper by several of the same authors [7] presents a tool for simulating the motion of vesicles. Inspiration from this paper should be pulled into CODS in future iterations. CODS doesn't currently provide the option to make simulated vesicles move but this is an important next step. The five simulated motions of vesicles from [7] would be a good place to start. It's important that organelles move naturally in training data. This paper uses the same sort of simulation supervised learning method as CODS and the above paper about the image simulator to test their training data.

This paper [11] presents a MitoGraph. A tool for calculating the volume of subcellular structures and organelles, namely mitochondria networks in yeast. MitoGraph is shown to be generalizable to mitochondria in other types of cells

and thought is given to how it might be useful in observations of other types of organelles entirely. This kind of flexibility is good news for the usefulness of solutions of all kinds. This paper discusses how in the past volume of organelles could be estimated by counting the number of pixels in a fluorescence microscope image are brighter than a certain threshold. This was not a good method because different organelles may have different brightnesses, especially those in different parts of a cell or at different depths in the Z dimension toward or away from the microscope lens. MitoGraph attempts to improve this calculation using image segmentation to identify which pixels belong to mitochondria.

This paper [10] explains a study interrogating mitochondria network structure. They observed the structure before and after stimulating fusion and fission in the mitochondria. The work describes the observation of a healthy middle between mitochondria which are chopped into small pieces and mitochondria networks which are too interconnected. Mitochondria are found to be holding themselves near a state of instability and that's how they're able to quickly become dynamic in response to changes. The paper presents a way to add parameters to mitochondria which could help to compare and classify them. The healthiness of the mitochondria network was measured using these parameters. It was noted that the randomness of the network increased when fission or fusion was induced, decreasing the health of the network. When networks were perturbed in this way it was also seen that their ability to fill space was effected. This means mitochondria networks are more able to spread out and occupy space more thoroughly when they are not induced to fuse or have more fissions than the natural amount. The method this work used to observe the structure of mitochondria from a microscopy image was to convert the image to black and white and then draw lines along all areas that were at least one pixel thick. Tools like CODS could be improved by the work introduced in this paper as well. If the parameters introduced here were calculated they could form an important part of the visualization capabilities of CODS by forming a more full picture of the evolving changing state of mitochondria networks.

This paper [14] investigates the effect mitochondria's morphology and motion has on the cell. Distinction is made between cultured cells and native cells, cultured cells being grown in a lab and native cells coming from a live organism being more complex and dynamic. The paper points out how mitochondria can quickly switch from playing a role in normal cell function to inducing cell death. The paper describes many different morphologies of many different cell types and describes the two membrane structure of mitochondria with four distinct regions: the membranes and the spaces between them with the inner membrane being least permeable. They say rod shaped mitochondria appear to bend and change shape while moving, spherical mitochondria do not appear to change shape while moving. The paper presents that mitochondria

morphological changes are observed in the cells of patients with diseases. This highlights how important it is to have access to better tools to help observe these organelles.

This review [8] discusses the proteins that appear to be related to mitochondria fission and fusion and how these proteins and mitochondrial behavior relates to the programmed death of the host cell. It points out that in the early stages of this kind of programmed cell death there can be observed dramatic changes in mitochondria morphology. They suggest that fusing mitochondria may exchange genetic information and use that as a means to repair mutations. The idea is that this would lead to a slower aging process of the mitochondria cells.

This paper [1] proposes how problems with mitochondria behavior could lead to neurodegenerative disease. It does this by exploring how mitochondria undergo fission and move intentionally to certain regions of the host cell. It talks about the specific proteins required for mitochondria fission and fusion. The paper describes how mitochondria fission and fusion seems to be related to content exchanges between mitochondria. It also explains how certain mitochondrial changes are needed for a host cell to undergo programmed cell death more easily, namely the mitochondria membrane must become more porous, and the mitochondria must perform fission resulting in small pieces.

This paper [9] discusses how mitochondria fission and fusion and mitochondria membrane ultrastructural remodelling are linked to the host cell's signaling and metabolism, including signaling related to cell aging, death, division, and ability to change into another type of cell.

This paper [12] describes observations of mitochondria shrinking and sometimes rolling up into a ball in response to stress. It talks about how under this situation it's common for mitochondria to swell up in size as well. A process is detailed whereby a small part of the mitochondria will inflate like a balloon and the rest of the tubular mitochondria will shrink along it's length. This process is described to happen within a short amount of time following a stressor and it's completed within a few minutes. This kind of quick reaction to an event is a good candidate for a behavior to be modeled by CODS in the future.

This paper [13] explains how it's possible to deform mitochondria physically using light to attach motors to the exterior membrane. Deformations stimulate the mitochondria to produce certain chemicals or fission in half. They were able to elongate mitochondria using this light stimulation up to a surprising 25 micrometers long. It was found that the deformation of mitochondria relied upon there being microtubules present. This shows that the modeling of microtubules should be recommended in future versions of CODS.

There are many relevant works being done in fields tangential to microscopy and biology as well. Important research is being done using computer simulations that is directly relatable to CODS. The lessons learned from reading the following papers were important in the development of the research experiment that is this thesis work. The first example that should be touched on is the work done on simulating muscle [19]. In this muscle simulator, a python environment is used with the ability to input a high number of parameters that allow the user to adjust the output. CODS takes inspiration from this path and in the end this last sentence could be used exactly to describe CODS. This paper [19] presents python code which simulates muscle cells for use in interrogating the results of different situations. Digital twins are another way to simulate biology on a computer in a useful way. Digital twins are discussed in this paper [20]. A digital twin, or a simulated version which is updated in real time via physical sensors, of cell cultures can be used to streamline the monitoring of cell cultures. Intracellular transportation is something of interest to biologists [17]. One way to learn more about this phenomenon and how it works is to simulate it and it's observed effects. A framework for doing just that is presented in this paper [21]. It simulates the transport of resources within a cell. Not only the transportation of resources but also chemical reactions and effects of these on the motion of the membranes of the involved molecules. This paper [22] presents a computer model for simulating the effects of different engineered substrates intended to control cell behavior. This model predicts changes in the morphology and movement of the cell and the cell nucleus.

Next, recognition will be given to several projects that are not so closely related to CODS but nevertheless provided critical inspiration and background knowledge for this thesis work. The following lines will recognise these projects. First, a ranking of 36 solutions for generating super resolution images is performed in this paper [23] by comparing their performance. They are tested by analysing simulated datasets and generating 2 dimensional and 3 dimensional images from them. A tool is presented in this paper [24] which is designed to determine the structure of biomolecules by generating the shape which would lead to the observed data. This is a better way than using the computationally intensive procedure involving using noisy tomographic projections to calculate the most likely shape and orientation out of all possible variations. This paper [25] presents a tool which can derive the structure of biomolecules in an unsupervised manner without using the computationally intensive standard methods. It tries to dig deeper into the specific contributions of these various mediums to lipid transport. This paper [26] tests capturing high resolution images of mitochondria derived vesicles in live cell and fixed cell environments. The density of mitochondria derived vesicles was compared in cells undergoing normal growth and cells experiencing metabolic perturbation. This paper finds that there were a higher density of mitochondria derived vesicles in a glucose deficient environment. Computer simulations and microscopy observations are

used here [27] to explore how the membrane of large vesicles change shape in response to small mobile self propelled particles within the vesicles.

# 3

# Preliminaries

CODS, the video simulator which this paper presents, is based on a tool [6] that generates pairs of images. The images are of simulated organelles. One blurry image mimicking microscopy imagery is produced along with a black-and-white ground truth image. This image generator is designed for use in training image segmentation models to segment microscopy images of mitochondria and vesicles. This is known as simulation-supervised training. To make referencing this tool easier, the image simulator presented by this paper [6] will be abbreviated as ImSim (Image Simulator). The team behind ImSim include several people involved in the project group responsible for this thesis including several supervisors. As presented in the paper [6], simulated images of mitochondria were generated using ImSim and they were shown to succeed in training deep learning models to perform segmentation on real microscopy images [6].

One reason this specific image simulator was chosen was it's performance in producing good training data for image segmentation. The fluorophores that help us view vesicles and mitochondria are well known to switch on and off periodically [28] [29] [30] [31]. This means they are sometimes producing photons and sometimes staying dark. This causes some spots on these structures in real life microscopy images to appear dark or dim some of the time as their emitters blink on and off. ImSim simulates this blinking by mimicking the observed probability of a fluorophore being on or off at a given moment. This blinking is not so noticeable to humans in the output of ImSim data because organelles can't be seen in the same spot in multiple frames back to back, so

**Table 3.1:** ImSim Parameters. These parameters exist in a list of configuration files which are meant to be modified to change the configuration of the simulation. The values given in this table are those in the example configuration files. Each parameter is explained right here in Preliminaries (Chapter 3).

| Mitochondria Parameter | Value |
|---|---|
| Number of Mitochondria | 1, 2 |
| Z depth of end A | 100, 600, 800 (nm) |
| Z depth of end B | 600, 800, 1200 (nm) |
| Number of Curve Points | 3, 4 |
| Width Range Low | 200, 300, 400, 500 (nm) |
| Width Range High | 300, 400, 500, 600 (nm) |
| Max Length | 5000 (nm) |

| Vesicle Parameter | Value |
|---|---|
| Number of Vesicles | 1, 2 |
| Z depth Range Low | 0, 300, 600, 800 (nm) |
| Z depth Range High | 300, 600, 800, 1300 (nm) |
| Radius Range Low | 50 (nm) |
| Radius Range High | 300 (nm) |

it's less clear that different areas would be switching between dark or bright. Across large amounts of data a machine learning model is effected by these variations and this training data produces a more resilient model.

One image segmentation technique is to compare the brightness of pixels to a threshold or convert to black and white [10]. The image generated often features bumpy segmentation boundaries which should be smooth in which the mitochondria and other organelles look deformed and they can even be cut into multiple pieces. Because ImSim produces a ground truth version of every frame in a video, it could perform well in training models designed for video that can recognize entities are the same from one frame to the next even though they are changing shape and size and location. ImSim has been shown to have success in training deep learning segmentation models that perform better than the state of the art [6]. Because the simulations produce predictable, uniform shapes, it trains models to expect these shapes and the segmentation produced are less bumpy and fragmented. When strung together in a video this could provide the needed ground truth improvement to make a tracking model find success. This section will explain some relevant inner workings of ImSim which help CODS generate individual frames of videos.

Images of cylindrical mitochondria and spherical vesicles are able to be generated in different shapes and locations by ImSim. The images which will

**Table 3.2:** Microscope Parameters. These are the parameters and values of the microscope that ImSim mimics via the PSF function [32]. The basic concepts explaining these parameters are discussed in Fundamentals of Microscopy (Section 2.1).

| | |
|---|---|
| Number of Rescaled Bessels that Approximate the Phase Function | 100 |
| Number of Pupil Sample Along the Radial Direction | 1000 |
| Magnification | 100,00 |
| Numerical Aperture | 1,4 |
| Coverslip RI Design Value | 1,515 |
| Coverslip RI Experimental Value | 1,515 |
| Immersion Medium RI Design Value | 1,515 |
| Immersion Medium RI Experimental Value | 1,515 |
| Specimen Refractive Index (RI) | 1,33 |
| Working Distance (Immersion Medium Thickness) Design Value | 150 Microns |
| Coverslip Thickness Experimental Value | 170 microns |
| Coverslip Thickness Design Value | 170 microns |
| Microscope Tube Length | 200 millimeters |

be produced are customizable using the user provided parameters which are shown in Figure 3.1 ImSim is only capable of producing the following configurations:

- One Mitochondria.
- Two Mitochondria.
- One Vesicle.
- Two Vesicles.

There is no way to use ImSim to produce an image with both a mitochondria and a vesicle in one image. These images are produced using a list of coordinate points which represent where emitters lie on the surface of simulated organelles. These light emitting particles which lie on the surface of organelles are the visible points of light which need to be simulated. These points are defined by their coordinates in 3 dimensional space. The simulated emitter points are blurred together and into the surrounding pixels of the output image using a PSF function [32] which matches the blurring which occurs in real microscopy images. The microscope configuration simulated by ImSim which the output imagery mimics is seen in Table 3.2. Many of the parameters listed here are explained in Background (Chapter 2.1). The 3 dimensional locations of fluorescent emitters are placed on the surface of the simulated organelles. These emitters are switched on or off in the image using a formula

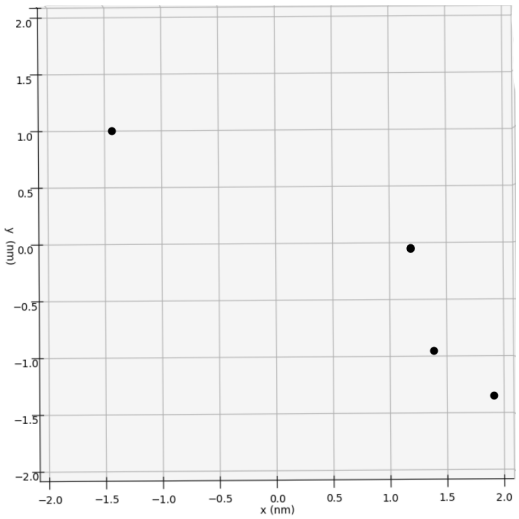


**Figure 3.1:** Curve Points. These points which define the curvature of the mitochondria are placed randomly in the XY plane.

to realistically reflect observed emitter photokinetics. This just means that the same image generated twice will have randomly different patches visible and invisible so the blurry patterns within the bodies of the organelles will be a different fuzzy pattern. This does not effect the black and white ground truth image generated with CODS which always shows every emitter in sharp binary contrast. The way these images are generated starts with plotting the points of these emitters in three dimensional space.

The shape of simulated mitochondria is defined entirely by several curve points. An example of four curve points can be seen in Figure 3.1. The curve points are chosen by randomly choosing coordinate points on the X and Y dimensions which will fall within the visible part of the coordinate space which will become the image. The length of the mitochondria produced from these curve points is checked by generating a spline from these curve points which is a curved line that passes through each curve point. This length is compared to the user supplied parameter dictating the maximum possible length of mitochondria. If the curve points define a mitochondria length which is longer than the maximum length allowed, new curve points are chosen until an acceptable length

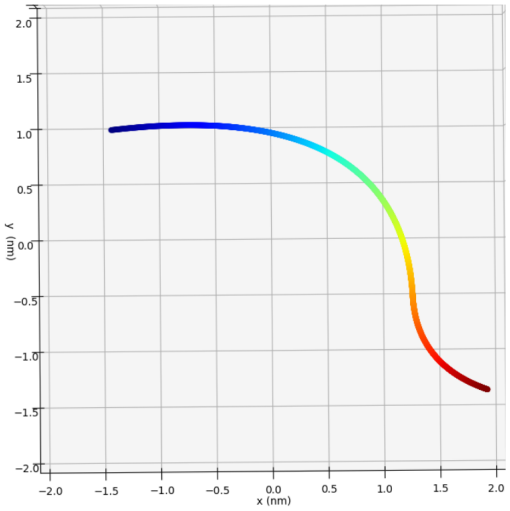


**Figure 3.2:** Center Line Points. These points are so high in number and close together that they resemble a continuous line. Red points have the largest Z dimension value and blue points have the lowest.

is found. The curve points are used to find the skeleton of the mitochondria. A series of points are plotted along this curved spline line connecting the curve points. These mitochondria skeleton points are called the center line points and they can be seen plotted in Figure 3.2. The center line skeleton points are given a z coordinate that stretches from the user provided z high value on one end of the mitochondria to the z low value on the other end of the mitochondria. The z dimension is color coded in these the two dimensional center line plot in Figure 3.2 and the three dimensional plot in Figure 3.3. The difference between the z coordinate value of one skeleton point and the next one is the same for all skeleton points, so the z values stretch linearly from one end of the mitochondria to the other. You can see this in the color coding on the center line figures. The red colored points on one end of the mitochondria are the highest z values and the blue colored points on the other end of the mitochondria are the lowest z values and the points progress gradually from red to blue across the entire length of the mitochondria. Each of these center line points is one nanometer away from it's neighbors in the X Y plane. These center line skeleton points are used to choose surface points which fall upon the surface of

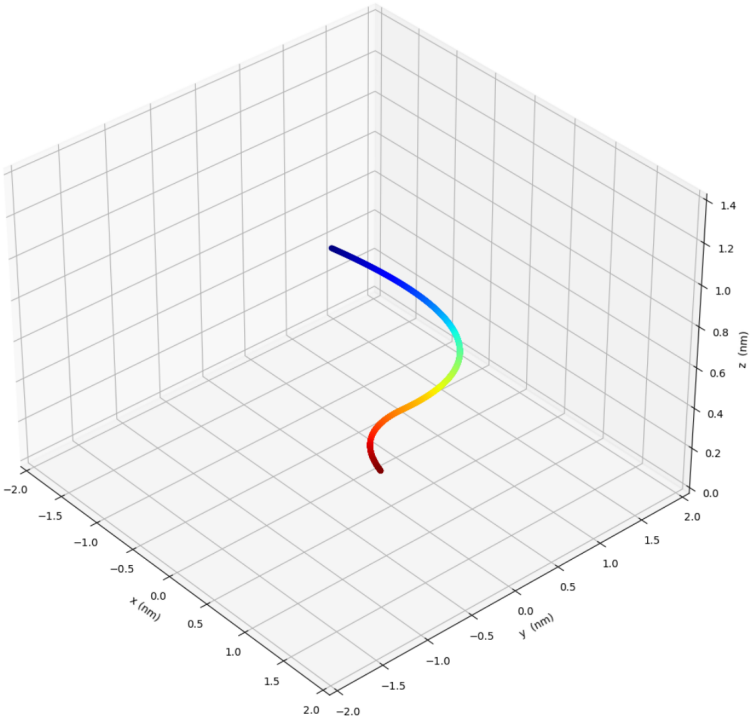


**Figure 3.3:** Center Line in Three Dimensions. These are the same points plotted in the previous two dimensional axis (Figure 3.2). These points are not photon emitters and therefor do not appear in the simulated images.

the simulated mitochondria as seen in Figure 3.4. The color coding meaning is the same in this figure with red points being higher in the z dimension. Surface points represent the light emitting particles on the surface of the simulated mitochondria. The center line points are used as the center of many circles which have a diameter equal to the width of the mitochondria. This mitochondria width is a user supplied parameter. The surface points lie on the edge of these circles. These are chosen at random points along the edges of these circles with a normal distribution. The surface points lie on a curved cylinder wrapping around the center line which leads to a kind of random scattered blanket of points on the surface of the simulated mitochondria.

ImSim has the ability to create images which do not have mitochondria, and instead have either one or two vesicles present. The process of simulating vesicles is somewhat more simple. A point is chosen in the xy plane at a random position as the center point of the sphere that is the simulated vesicle. The emitter points lie on the edge of the sphere which has the diameter of the width of the vesicle. Each vesicles width is chosen at random from between a user

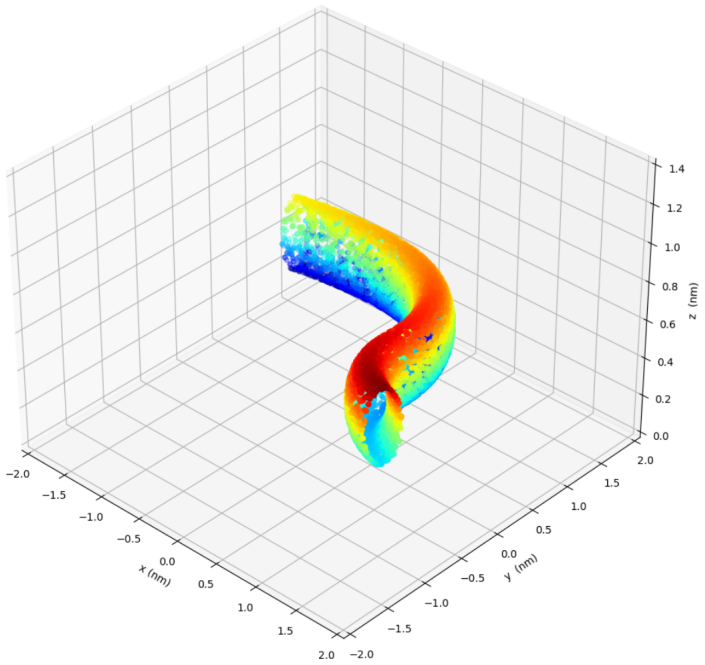


**Figure 3.4:** Surface Points. These photon emitting points are placed on the surface of the simulated mitochondria which is defined by drawing circles around each of the center line skeleton points from the previous plot (Figure 3.3).

defined minimum and maximum range. The vesicle also has a z dimension value that is chosen in the same way from between two user provided minimum and maximum values. The location that an emitter is in the z dimension determines the amount of blurriness given by the PSF function to simulate focus of the microscope. If a point lies between 600 and 800 nanometers then it will be the least blurry. This is the topic discussed in Fundamentals of Microscopy (Section 2.1) where the angle of light is changed by bringing the observed object closer or further away. For this to happen however the cover slip need not change location. A microscopic organelle drifting upward or downward a few hundred nanometers in it's tiny pool of liquid can bring it in and out of focus. In focus or not, simulated photon emitting points of the vesicle or mitochondria have had their coordinates defined and they're ready to take part in the next part of image generation.

The emitter points on the surface of the organelles are put in a list. This list is used to produce the binary ground truth image and the blurry microscopy image. The blurry microscopy image is sometimes refered to as the tif image

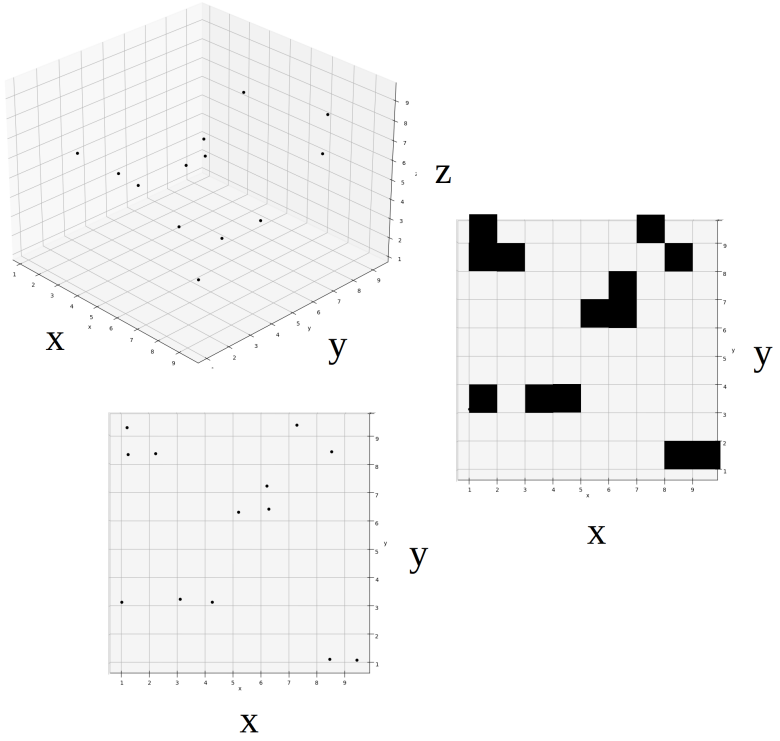


**Figure 3.5:** Ground Truth Points to Pixels. The list of emitter points is translated into a ground truth image by removing the Z dimension and then coloring areas with points white and areas without points black.

since that's the file format CODS produces. In the case of the blurry image, the points are run through a PSF function [32] which takes as parameters the attributes of the microscope which is being simulated from Table 3.2. The PSF function takes the three dimensional points and outputs blurred pixel values. The Z dimension is taken into consideration and important in the calculation for how blurry the light coming from that point should be. This lets CODS simulate the focus of the microscope. These pixel values can easily be turned into an image file. The other thing generated is the binary ground truth image. The original list of emitter coordinate points is translated into 2 dimensions by simply removing the Z dimension value from each point. Then these points are translated into a grid with the same shape as the number of pixels in the desired image. The grid squares with points turn white and those without points are black background pixels. Figure 3.5 is an illustration of this basic process with some random points for illustration.

From generating pixel values from point coordinates to simulating microscope focus, everything that has been described in this chapter has been adapted for

**Table 3.3:** Features Compared. The niche that CODS fills is in simultaneous complex dynamic interactions between organelles (for now only mitochondria). Noise should be added as one of the next features of CODS in the future.

| **ImSim** | **VeMoS** | **CODS** |
|---|---|---|
| Only 1-2 Mitochondria or 1-2 Vesicles | Limited Number of only Mitochondria | Unlimited Mitochondria and Vesicles |
| Noise | Noise | No Noise |
| Static | Vesicle Motion | Mitochondria Motion |
| Static | No Interaction | Diverse Interaction Between Mitochondria |
| Sequential | Sequential | Parallel |

use inside CODS. CODS builds on the capabilities of ImSim because of this. Naturally, the research closest in scope and methods to CODS has been done by those in the same project group at the same institution where the CODS system was conceived. A simulator presented by the paper [7] discussed in Background (Chapter 2) will be abbreviated VeMoS here (Vesicle Motion Simulator). Table 3.3 shows a comparison of the features offered by CODS, ImSim [6], and VeMoS. These systems are compared because they attempt to achieve the most similar goals to each other. As can be seen from this comparison, the focus of CODS is on the motion of mitochondria. But despite the limited scope of the current CODS system, it offers a sturdy foundation to build up other features in the future. ImSim is an important piece of the puzzle and to understand CODS you must first understand ImSim.

As is seen in this chapter, ImSim takes several parameters configurable by the user and generates an array of data. This data is quite narrow on scope, being only single images of a fixed size and fixed number of organelles, but it performs well as an image segmentation training data generator. In the next chapter, CODS will be given a similar treatment of being broken down to it's components and examined. CODS takes a much larger number of parameters and the diversity of it's fine tuning capabilities are of a similar order of magnitude more plentiful.

# 4

# Methodology

Figure 4.1 shows a preview of what will be discussed in this chapter. The problem that CODS solves is the need for motion to make a video, which is a series of images through time each separated by a uniform duration. As seen in Figure 4.2, the problem scenario to solve is the movement of an organelle between the individual image frames that make up a video. One key thing that makes CODS unique is the creation of a temporal difference between output images. As seen throughout the literature [1] [10] [14] mitochondria change their physical location within the cell containing them over time. The goal is to train models to expect the common features of real life dynamic objects and the way to do that is to use data that more closely mimics more features of those objects. In a video, what gives objects motion is their spacial location changing over time. Different points in time are represented in video data by the frame number in an ordered group of frames. In microscopy data, organelles taking part in various behaviors defined by changes in their shape and position relative to the edges of the video and other organelles. These behaviors take time to carry out and this changing of shape and location can be observed smoothly frame after frame. CODS is able to simulate this spatial temporal modification of mitochondria. To simulate motion in an organelle it's position must change from one frame to another. Observation of these organelles happens via the collection of photons coming from the position of photon emitting fluorescent particles. As the simulated body of the organelle moves and deforms from one frame to the next, the positions of these emitters change. The location where photons are being emitted from changes and this is visible in the simulated output video. This is because the emitter points move along with the organelle in the sim-

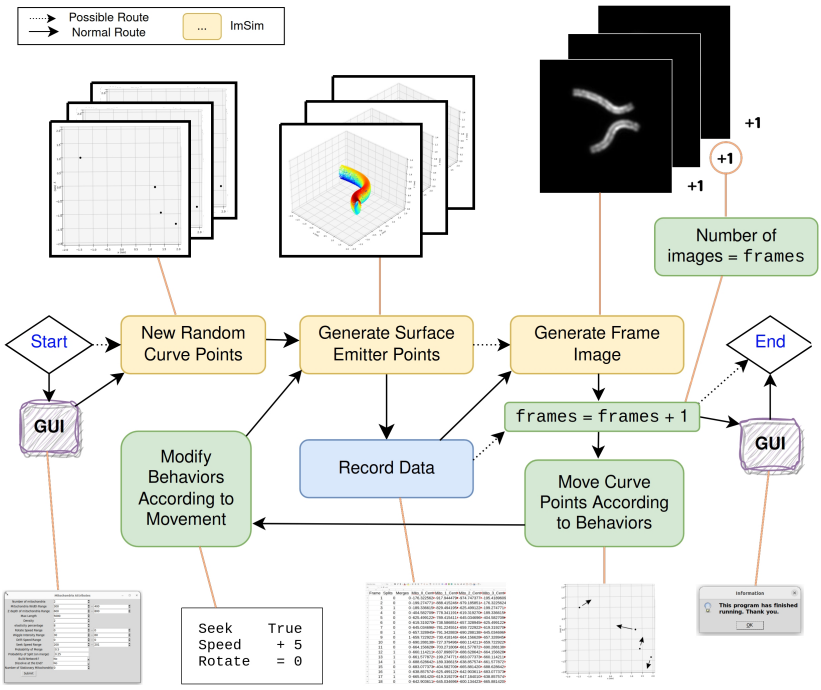


**Figure 4.1:** Detailed Pipeline. Dotted lines are paths that skip optional steps.

ulation and their emitted light is seen to be moving around from one frame to the next. Making the leap from image data to video data has the potential to provide important benefits. With image data observations are possible such as the number of mitochondria in an area and what kind of networks they're organized into. But with videos more information can be captured like how the number of certain behaviors change over time in various situations. With the ability to analyse video data there comes the ability to talk about the same system of organelles reacting to different stressors over time or simply get a better understanding of the ebb and flow of behaviors under normal conditions. There are many ways to analyse video data and some roads to get there are discussed in the next paragraph.

The training data produced by this thesis project work has many promising uses. One highlighted example use case is training video segmentation models. Video segmentation is more complex than segmenting an image over and over again for each frame. In order to segment pixels that make up one particular blurry microscopic entity in a video with the highest accuracy throughout multiple frames, this entity could be identified and tracked. ImSim is the name used in this thesis work to refer to the image generator introduced in Preliminaries (Chapter 3). ImSim generates images for use in segmentation models and per-

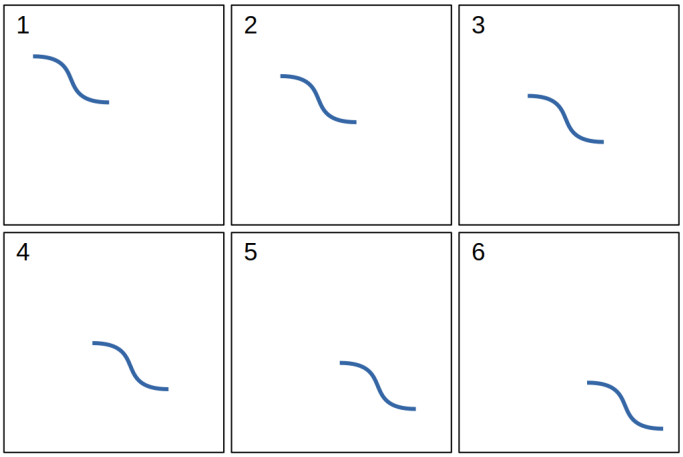


**Figure 4.2:** Problem Scenario. CODS is developed to solve the problem of needing smooth motion simulated over time for use in training various machine learning models.

forms very well on still images [6] but does not attempt to cater to video data. As can be seen in the sequence of images at the top of Figure 4.4, ImSim alone is incapable of producing anything resembling a video for use in video segmentation. ImSim cannot consider the position of the last image it generated when generating the next one, therefore they are all randomly placed and you cannot make a video with the images it produces. This should be compared with the output of CODS in the sequence of images on the bottom of Figure 4.4 which is clearly simulating motion where mitochondria shift gradually from one frame to another. As seen when viewing these frames in order from left to right, top to bottom, CODS successfully solves the problem scenario. Mitochondria make incremental changes from one frame to the next resulting in a smooth intentional movement across time. Among other use cases described in Applications (Chapter 6), CODS also provides a convenient solution for this gap illustrated by the problem scenario in Figure 4.2. CODS could be used to harness the power of ImSim combined with the potential for tracking and behavior recognition. CODS uses ImSim but it could use any image simulator that works by defining organelle locations using curve points in a similar way. CODS has the option to allow the user to completely disable the image generation functionality. This is useful if data is needed only about the locations the simulated mitochondria are in and their behavior and interactions with each other. As can be seen in the simple pipeline diagram in Figure 4.3, CODS makes use of ImSim to simulate the mitochondria within individual frames, but the work of simulating the movement between frames which makes up greater behaviors over time

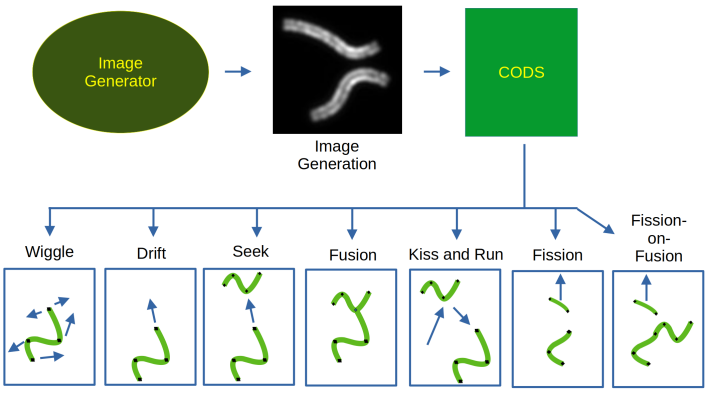


**Figure 4.3:** Simple CODS Pipeline. This figure shows ImSim as an input and various dynamic behaviors as an output.

is accomplished by CODS. In the next section, some mechanics of how this is done are presented.

## 4.1 Execution of CODS Overview

The user has the option to modify various parameters to fine tune the output video(s). These parameters are received before the video is generated. One option for inputting these parameters is via a graphical user interface. Table 4.1 shows the parameters included in one of these windows. There are two more graphical user interface windows and their parameters are shown in Tables 4.2 and 4.3. Screenshots of these windows can be found in Coding Documentation in Appendix A. A fourth window is included in CODS graphical user interface and that includes options to modify all values of the microscope parameters shown in Table 3.2 discussed in Preliminaries (Chapter 3). If a user changes nothing in the graphical user interface, the default parameter values will result in some simple data being generated. The graphical user interface is optional as can be seen in Figure 4.1 and CODS can be run on a headless server. Some examples of these parameters are the number of videos and the number of frames in each video that will be generated. Sample is the term used to mean an individual video, so if a user indicates 10 should be the input for the parameter which controls the number of samples and they enable the parallel samples parameter, then CODS will work on ten different videos at the same time. Many of the parameters offer the option of selecting a minimum and maximum range of values. In Table 4.2, it can be seen that this is true about many parameters related to the physical attributes of mitochondria such as their width and their behavioral attributes such as their wiggle intensity. For these parameters that allow distinct minimum and maximum values, a random value N is chosen for

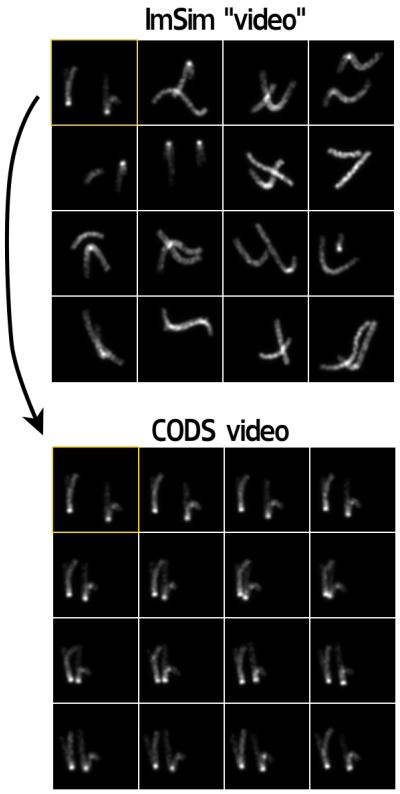


**Figure 4.4:** Unordered ImSim "Video" and Ordered CODS Video. The top sequence features the closest thing to a video that ImSim can produce using user provided parameters. The bottom sequence are the frames of a CODS video. In this video, the mitochondria on the right advances toward the mitochondria on the left and performs a kiss and run [33] which sends it drifting away to the right.

that parameter which falls between the inputted upper and lower boundary limits such that minimum < N < maximum. This is done separately for each video so they all have a unique configuration of parameters. CODS can take some time to produce videos. These chosen parameters are immediately visible to the user while CODS is running and they are also always recorded for future reference. The graphical user interface allows more visual manipulation of the parameters. Many of the parameters have only two options and these can be toggled using dropdown menus from within the graphical user interface.

When generating a simulation video with CODS, there are several steps that must be completed before the final video is ready to be produced. Before work begins on the first frame of a video, values are selected for the parameters which have upper and lower ranges. For example one such parameter is the default

**Table 4.1:** CODS System Parameters. The PSF parameter is explained below in Parallel PSF (Section 4.14). The record coordinates parameter is explained in Applications (Chapter 6.6). The print Tiff and GT parameters are what allow disabling the image generation functionality. The RAM usage reduction parameter and mode parameter are explained below in Reduced Memory Approaches (Section 4.13. Also explained in the Reduced Memory Approaches section is the parameter labeled PSF for each mitochondria in one frame. The parameter labeled Random Behavior + Record Centerpoints is explained in Random Behavior Profile (Section 6.2).

| Parameter | Value |
|---|---|
| Canvas Width and Height | 500-50.000 (nm) |
| Number of Frames | 1-∞ |
| Number of Samples | 1-∞ |
| PSF Parallel? | Yes, No |
| Samples Parallel? | Yes, No |
| Record Coordinates? | Yes, No |
| Generate Images? | Yes, No |
| Generate Ground Truth? | Yes, No |
| Low Memory Mode? | Yes, No |
| PSF Multiple Entities/Frame | Together, One at a Time |
| Fewer Emitters (Testing Mode)? | Yes, No |

seek speed of a mitochondria. This comes into play when a mitochondria needs to target another mitochondria and approach it with the possibility of fusing with it. The seek speed is the speed that a mitochondria will move toward another mitochondria in this situation. The user can provide a minimum and maximum seek speed for a batch of videos. A different seek speed will be chosen for each mitochondria in each video such that the values will be different in different videos. This allows for variability in the videos so the data is not too homogeneous. The random value N is chosen between the minimum seek speed MinSS and the maximum seek speed MaxSS such that MinSS < N <MaxSS. If the maximum and minimum seek speed are exactly equal, then all seek speeds are be set to be equal to the minimum seek speed value.

A user defined parameter controls the width and the height of the canvas. This determines the number of nanometers high and the number of nanometers wide that the generated video will be. CODS runs ImSim [6] over and over to generate one frame after another until a number of frames is reached equal to the user inputted frame number parameter. The position and shape of a mitochondria is entirely determined by the location of several points called curve points and the value which defines the mitochondria's width. Vesicles can be similarly reconstructed using only their radius and the coordinates of their

**Table 4.2:** Mitochondria Parameters. Several of the topmost parameters are the same parameters used in the ImSim image simulator explained in Preliminaries (Chapter 3). The elasticity parameter is explained below in Wiggle Behavior (Section 4.9). The rotate, wiggle and drift ranges are explained below in their respective sections (Sections 4.12, 4.9, 4.3). The seek speed range parameter is explained here in Execution of CODS Overview (Section 4.1). The probability of merge parameter refers to the probability of a Fusion behavior occurring and that is explained below in Fusion Behavior (Section 4.6). The probability of split (on merge) parameter refers to the probability of a fission behavior being triggered by a fusion behavior and this is explained in Fission on Fusion (Section 4.8). The network and dissolve parameters are explained in Network Example Behavior Profile and Dissolve Example Behavior Profile (Section 6.7 and 6.8). And finally, the stationary mitochondria parameter is explained in Stationary Mitochondria (Section 4.16)

| **Parameter** | **Value** |
|---|---|
| Number of Mitochondria | 0-∞ |
| Mitochondria Width Min | 50-1.000 (nm) |
| Mitochondria Width Max | 50-1.000 (nm) |
| Z Depth of End A | 100-2.000 (nm) |
| Z Depth of End B | 100-2.000 (nm) |
| Maximum Length | 50-∞ (nm) |
| Elasticity | 1%-∞% |
| Rotation Min | 0-360 (Degrees per Frame) |
| Rotation Max | 0-360 (Degrees per Frame) |
| Wiggle Intensity Min | 0-1.000 (nm) |
| Wiggle Intensity Max | 0-1.000 (nm) |
| Drift Speed Min | 0-1.000 (nm) |
| Drift Speed Max | 0-1.000 (nm) |
| Seek Speed Min | 0-1.000 (nm) |
| Seek Speed Max | 0-1.000 (nm) |
| Probability of Fusion | 0-1 |
| Probability of Fission-on-Fusion | 0-1 |
| Random Behaviors and Record Center Points | Yes, No |
| Build Network Behavior Profile | Yes, No |
| Dissolve at the End Behavior Profile | Yes, No |
| Number of Stationary Mitochondria | 0-1000 |

**Table 4.3:** Vesicle Parameters. Adding vesicles to videos generated with CODS is an available feature, however they don't yet move at all or interact with other organelles.

| Parameter | Value |
|---|---|
| Number of Vesicles Min | 0-∞ |
| Number of Vesicles Max | 0-∞ |
| Vesicle Radius Min | 1-10.000 (nm) |
| Vesicle Radius Max | 1-10.000 (nm) |

center point. The coordinates of these four parameters are all that is needed to remember between frames and with that data the same organelles are able to be rebuilt after tweaking them a small amount to simulate movement. There is a set of behaviors that a mitochondria can be involved in that effect it's motion over the length of a video. On a given frame, each mitochondria has individual behaviors switched on or off. The curve points which were recorded can be modified in various ways according to these behaviors. These shifts in position between frames is what creates the effect of a mitochondria in motion when the frames are played together as a video. The position of a mitochondria being changed is notable because the position of a mitochondria can have an effect on which behaviors it is taking part in. For example in a fusion behavior the initial position a mitochondria is in when it begins a seek behavior will determine which part of it is closest to which part of the target mitochondria. That in turn will determine whether it fuses on some middle point or an end point. As we'll see later, if the fusion happens on a middle point, this has the potential to trigger a fission behavior which further adds complexity to the system by introducing an additional mitochondria into the mix. Depending on what the user wants, several more actions can take place. Various point coordinates associated with the mitochondria can be recorded for use in various applications. Another thing that effects the events triggered on a given frame is the behavior profiles which are active in a batch of videos. A behavior profile is a set of instructions that introduces additional activity into a video. If data is needed about mitochondria joining together into networks then the network behavior profile can be switched on. If data is desired with examples of mitochondria performing a fission resulting in two new mitochondria then the dissolve behavior profile can be activated. More detail about behavior profiles is given at the end of this chapter.

## 4.2 Choosing Curve Points

As can be seen in Figure 4.1, the first few steps in the CODS pipeline are basically ImSim. However, important changes needed to take place to enable the wider capabilities of CODS. The modification to ImSim relevant for this section relates to the pixel width and height desired in CODS videos. To accommodate more than two mitochondria in the same video, there must be more than 128 pixels square, which is the only option in ImSim. Increasing the canvas on which organelles are drawn has an impact on the process of choosing curve points. This will be explained first.

On the first frame a random starting shape and location is chosen for each mitochondria. These attributes are entirely defined by four curve points. To choose the curve points, CODS chooses random numbers for the X and Y values of the coordinates for each curve point within the user specified canvas size. The maximum length of the mitochondria is a user chosen value. If the curve points would result in a mitochondria which is longer than the maximum allowed length, then CODS discards the curve points it chose and picks four entirely new curve points again and again until it discovers a set of points that result in a mitochondria whose length is less than the maximum length allowed. So far, this is just like ImSim but this is where the descriptions of these two systems starts to diverge. CODS allows the user to chose any size canvas they want and if the maximum mitochondria length is substantially shorter than the width of the canvas then it could take a while to find an acceptable set of curve points because most combinations of points would result in a mitochondria which is too long. To arrive at an acceptable length faster, the coordinate space which these random points can be generated within is reduced in size strategically. This is done by only generating potential points within a bounding box which has a width and height equal to the maximum mitochondria length. To be sure the mitochondria don't always appear in the center of the canvas, they are moved a random direction by a random amount within the canvas. The XY coordinates of a possible curve point are shown in the following equation where i is a random initial coordinate and o is the random offset that the point is moved away from the center:

$$\begin{gathered} x = i_x + o_x \\ y = i_y + o_y \\ \text{where,} \\ 0 < i_x, i_y < l \\ \frac{-c}{2} < o_x, o_y < \frac{c}{2} \end{gathered} \tag{4.1}$$

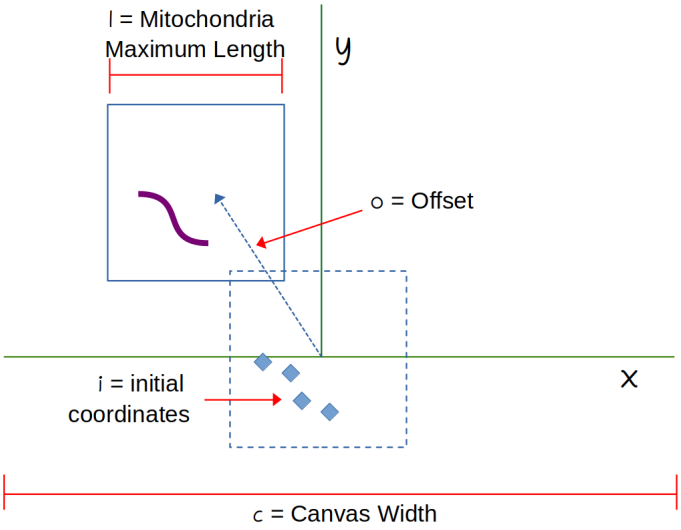


**Figure 4.5:** Curve Point Selection. This figure illustrates the meaning of the letter variables i, o, l, and c from Equation 4.1. Mitochondria are placed in a random position within the small bounding box and then moved in a random direction by a random offset within the canvas. This makes finding a mitochondria shape and size which is the appropriate length quicker.

Where c is the length of the canvas and l is the maximum mitochondria length. The origin is in the center of the video canvas. This configuration of the bounding box and offset can be seen in the Figure 4.5. On later frames after the first frame, CODS skips the step of generating random locations for curve points and instead uses the curve points which it remembers from the previous frame. Once the coordinates of the curve points are found, ImSim takes care of producing an image and ground truth for that frame of the video. To produce behaviors, motion is needed. And to produce motion between frames, CODS just needs to adjust the remembered positions of the four curve points from the previous frame.

CODS enables several classes of behavior of mitochondria. These behaviors are produced by changing the position and shape of the mitochondria consistently between frames which creates a smooth intentional motion. Each mitochondria is assigned a set of behaviors that it is currently supposed to be actively participating in. Between the generation of one frame and the generation of the next frame, the coordinates of the curve points of each mitochondria is adjusted based on each of it's assigned behaviors. Then when the next frame is generated, it appears that each mitochondria has moved. The behaviors created for CODS are not an exhaustive list of the activities at the disposal of mitochondria in the real world. This set of possible actions can and should be expanded

in the future to become more realistic. One excellent example of a behavior which could be modeled next is the flip maneuver.

## 4.3 Drift Behavior

These next several sections will discuss several behaviors available to be selected within the CODS simulation engine. Notable examples within the behavior suite and illustrations of these are in Figure 4.3. Each mitochondria has a drift direction and a drift speed associated with it. If the user assigns a maximum drift speed greater than zero and CODS selects a value for this parameter greater than zero or if the minimum drift speed is also greater than zero then a drift maneuver is executed. When this happens a mitochondria's curve points translate across the screen. When the surface points are generated during the next frame, they will fall around the new modified curve points. During a drift, the curve points of a mitochondria are translated by the drift amount in the mitochondria's assigned drift direction in the two dimensional X Y plane. This is done by adding the x values of each curve point's coordinates to some x drift value and adding the y coordinate values to some y drift value. These x and y drift values are found using the following equation.

$$x_drift = \frac{dx * d}{\sqrt{(dx * dy) + (dx * dy)}} \tag{4.2}$$

$$y_drift = \frac{dy * d}{\sqrt{(dx * dy) + (dx * dy)}} \tag{4.3}$$

Where dx and dy are the direction coordinates and d is the distance parameter. These direction coordinates are chosen at random for each mitochondria before the first frame is generated. The drift direction is defined by it's component x and y coordinate values. One value for the relative translation that must happen in the X direction and one value for the relative translation that must happen in the Y direction. The only thing that matters about these two values is their values relative to each other. The sizes of these values do not effect anything except for the angle they produce when taken together. For example an X direction value of 5 and a Y direction value of 10 for mitochondria 2 is the exact same as an X direction value of 2 and a Y direction value of 4 because it results in the same angle. The speed of drift is managed by the drift speed parameter which is the length of the translation that occurs between each frame. The drift behavior is a critical part of the seek behavior.

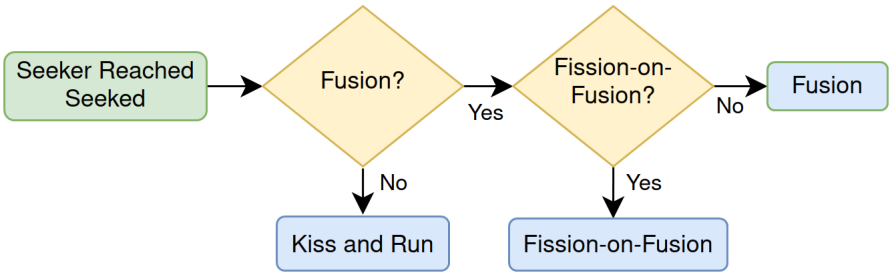


**Figure 4.6:** Seek Flowchart. Illustrations of seek, kiss and run [33], fusion, and fission-on-fusion can be seen in Figure 4.3.

## 4.4 Seek Behavior

In order for two mitochondria to fuse together into one bigger mitochondria, first one of them must seek out another. First one mitochondria must calculate a direction to move in that will bring it closer to another mitochondria, and then it must drift in this direction by an amount equal to the seek speed variable which is provided by the user. The seeker mitochondria translates it's position in the direction that will bring it closer to the seeked mitochondria. After the seeker arrives at the seeked mitochondria it can trigger several possible behaviors after that. This is illustrated by the flowchart in Figure 4.6. Every time a seek has been successful by way of the seeker mitochondria arriving to overlap one of it's curve points with the seeked mitochondria, there are three possible outcomes as shown in this flowchart. Depending on the values of several user provided parameters the relative probabilities of these outcomes can change. These parameters are the probability of fusion and the probability of fission-on-fusion. When a mitochondria is set to be in the process of executing a seek behavior, the drift behavior is ignored. This is to prevent the two behaviors from coming into conflict with one another which could prevent the seek from successfully being carried out. When a seeker mitochondria arrives at the seeked mitochondria it will either trigger a fusion whereby the two mitochondria become linked forming one big mitochondria, or it will trigger a kiss and run behavior. A kiss and run is when a seeker mitochondria arrives at a seeked mitochondria and drifts away from it. The kiss and run behavior has been observed in other subcellular machinery as well [33] and it sometimes is believed to be useful for content exchange between organelles. Every mitochondria has a seek speed associated with it. When CODS begins generating a video before it has started on the first frame, the mitochondria are not set to be seeking any other mitochondria. If a mitochondria is later set to seek another mitochondria, then the closest two curve points are identified in the seeker and the seeked. These two points are identified by comparing the distance of each point in the seeker to each point in the seeked. When the shortest distance

is found, then this point in the seeker and this point in the seeked mitochondrias are used. The seeker will drift by an amount defined by the seek speed associated with the seeker in such a direction that it will bring these two curve points closer together. This direction is defined by the difference between the X and Y values of the coordinates of these two curve points. If a seek behavior is being performed and these closest curve points are close enough already that they are within range to be united in just one frame then the seeker will only move by the amount needed to make these curve points overlap, and in this way it does not overshoot it's goal. Then a value called probability of fusion comes into play. This was user provided and it determines the probability that a fusion will take place upon successful uniting of the two curve points. If the probability of fusion is 50 percent, then 50 percent of the time the seeker and seeked will become fused together, and the rest of the time a kiss and run will be performed. When the two fusion points overlap, the seeker mitochondria is set to no longer be seeking anymore.

## 4.5 Kiss-and-Run Behavior

When a seeker mitochondria reaches the seeked mitochondria and a kiss and run has been triggered, the seeker mitochondria moves away from the seeked mitochondria at a constant speed. The direction of travel is set by adding a new random drift direction to be associated with the former seeker mitochondria. This is added at the moment the kiss and run is triggered. The drift speed is set so that the mitochondria continues along it's new path in future frames. The drift speed is chosen at the moment the kiss and run is triggered and falls between the user set minimum and maximum drift speeds which were provided by the user. The other option after a seek is of course a fusion of the two mitochondria

## 4.6 Fusion Behavior

When a seeker mitochondria has one of it's fusion points overlapping with a curve point of the seeked mitochondria, a fusion can be triggered. During and after a fusion, these overlapping curve points can be referred to as fusion points. This entails the seeker mitochondria taking on the drift direction, drift speed, rotation, and wiggle intensity of the seeked mitochondria so that they begin moving together across subsequent frames. A record of this fusion is created and CODS will try to keep these two fusion points overlapping in subsequent frames. The fusion also means that some behaviors will be executed by the fused mitochondria as one all together. For example, this is true about the

rotation behavior because now the fused mitochondria must rotate around a shared central axis. If the user sets some higher than zero probability of fission-on-fusion, then there's a chance a fission will take place at the moment a fusion is completed. In this case, if the fusion point on the seeker or the seeked mitochondria is one of the two center curve points, then the mitochondria may perform a fission behavior at that fusion point. This will happen or not according to the value of the probability of fission-on-fusion parameter. But before fission on fusion can be discussed, an explanation of the simple fission is needed.

## 4.7 Fission Behavior

A mitochondria can be triggered to perform a fission resulting in two separated pieces. This fission happens at the location of a curve point belonging on the original mitochondria which is to conduct the fission. If the curve point which is to be the fission point is an end point of the mitochondria then a fission will not take place. Instead, any mitochondria which are fused to this end point will be unfused with the mitochondria which is executing the fission behavior. The record of the fusion of any mitochondria to this fission point will be erased, causing them to be able to move independently of each other. If the fission point is one of the two middle curve points, then one of the two resulting mitochondria will be left with three of the original mitochondria's curve points and the other one will be left with two. The resulting mitochondria with fewer leftover curve points with the same coordinates of the curve points in the original larger mitochondria will be referred to as the smaller mitochondria here for simplicity. This smaller mitochondria may actually be longer depending on how close together some of the curve points of the original mitochondria conducting the fission were. The drift direction of the smaller mitochondria is set to a new random direction. The larger mitochondria will be given any records of fused mitochondria who were fused to the fission point of the original mitochondria executing the fission behavior. One curve point will be added between the second and third curve points of the larger mitochondria. The smaller mitochondria will have two new curve points added between it's curve points so that it will not stay in the boring shape of a straight line. After the fission, the new length is recorded and this is an important value due to mitochondria elasticity which will be discussed down below in the section about the wiggle behavior.

## 4.8 Fission-On-Fusion Behavior

One user provided parameter that is available is the probability of a behavior called fission-on-fusion. This is the name for a fission behavior which has been triggered by a fusion behavior. Whenever there is a fusion behavior triggered, CODS checks this probability and if the probability is greater than zero then there's a chance of triggering a fission behavior. The fission behavior will be triggered on the fusion point. Fission behaviors can have a different effect depending on which curve point is the location of the fission. If a fission occurs on an endpoint, fused mitochondria will be broken off and if a fission occurs on another curve point which is not an endpoint, then the mitochondria is cut into two new ones on that fission point.

## 4.9 Wiggle Behavior

If the user provided parameter of minimum wiggle intensity is greater than zero, then every frame features a wiggle behavior by all mitochondria. This mimics the shaking or perturbation motion observed in microscopy videos [34] [35].. This is where it appears that the mitochondria is shivering or jerking around on a small scale. During a wiggle, the curve points of the wiggling mitochondria are moved in a random direction on the X and Y axes by a random amount between zero and this mitochondria's assigned wiggle intensity. The Z dimension is not modified. This is not realistic and in the future it is recommended that this be modified to include some motion in the Z dimension. This produces a shivering like effect. The wiggle observed in real microscopy video happens in live cells featuring mitochondria as can be seen clearly in the videos provided in Example Scenarios (Chapter 5). The length of the mitochondria needs to stay somewhat fixed over time so the length of the mitochondria doesn't shift dramatically over time. This somewhat constant length is observed in microscopy videos and this is why CODS works hard to keep mitochondria a consistent length over time. This random shift in the curve points from their starting position during a wiggle event is repeated until a configuration is found which does not increase or decrease the length of the mitochondria more than a particular percentage chosen by the user. This percentage is called the mitochondria elasticity. When a mitochondria is first generated, the length is recorded and this original length is the value which is compared against during subsequent wiggles so that the length doesn't drift.

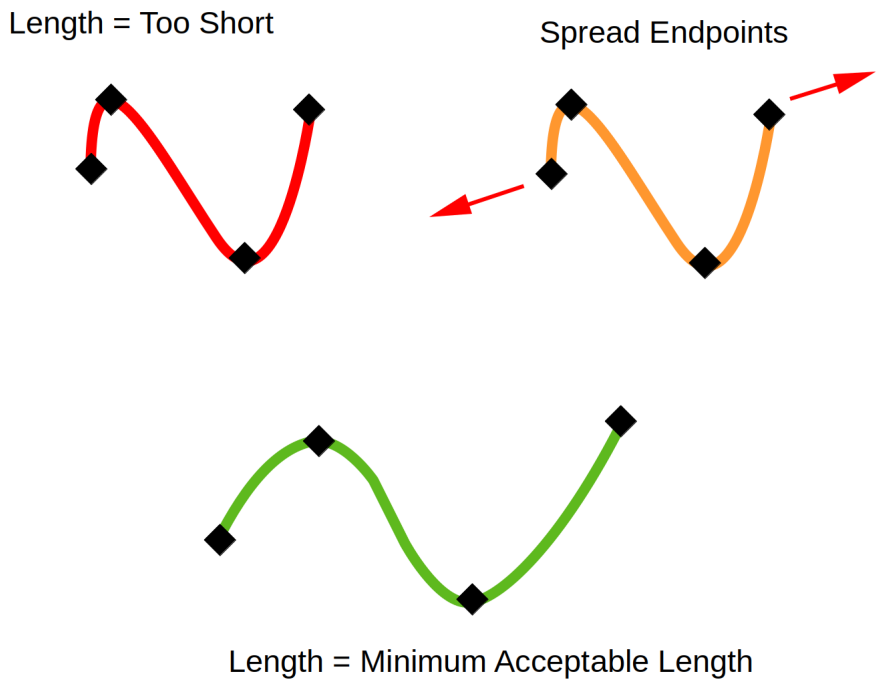


**Figure 4.7:** Spread Endpoints Behavior. When a fission occures which produces a mitochondria which is too short, the end points are translated away from each other an amount that corrects the total length of the mitochondria.

## 4.10 Spread Endpoints Behavior

If a mitochondria undergoes fission and becomes so short that it is less than 280 nanometers long, then it is lengthened to reach 280 nanometers. A mitochondria is lengthened simply by spreading it's two curve points on either end directly away from each other as shown in Figure 4.7. This is done so mitochondria don't get too small. When a mitochondria is around 280 nanometers, it's width and length are similar and it appears like a little sphere. These tiny blobs of mitochondria can be likened to what are called mitochondria derived vesicles which can bee seen in real microscopy videos of live mitochondria. The spread endpoints behavior is a limitation of the system CODS. The reason the mitochondria cannot become shorter than 280 nanometers is because of the elasticity requirement. When CODS is calculating the curve point movement during a wiggle behavior, it tries many different variations in the new locations of curve points until it finds a configuration which doesn't change the length by more than a certain percentage of the total length of the mitochondria. When a mitochondria's length becomes tiny, this presents a problem because the wiggle intensity could be quite large in comparison to the length of the mitochondria.

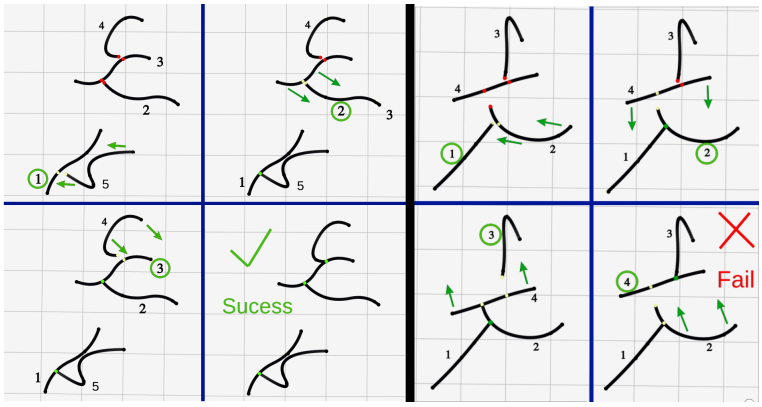


**Figure 4.8:** Connect Fusion Points. The mitochondria which has the green circle is having each fused mitochondria moved in toward it to reconnect the points which should be fused together. With the order being as shown, the configuration on the right will never accomplish all fuse points overlapping at the same time. However, what would happen if mitochondria 3 and 4 swapped numbers?

So the amount that curve points will drift in random directions during a wiggle behavior may be so large that an acceptable configuration may not be found. One elegant solution for this problem would be to detect when these ratios are reaching dangerous levels and to adjust the value of the wiggle intensity accordingly. This approach was not taken due to time constraints. Another approach which was not implemented but is simple enough to be implemented in the short term is to check if a mitochondria fission will lead to a mitochondria being created which is too short, and then simply abort the fission behavior. This limits the agency of the researcher to fine tune behavior patterns featured in video data.

## 4.11 Connect Fusion Points Behavior

A record of a fusion means that one curve point in one mitochondria is set to be a fusion point in a fusion point pair with a curve point in another mitochondria. This means that CODS will attempt to keep these two fusion points at the same location in the X and the Y dimension. When mitochondria wiggle, their fusion points drift randomly away from each other. Because of this, fused mitochondria need to be reconnected to each other. This random movement in one fusion point during a wiggle is almost always different from the random movement made by the fusion point in the other mitochondria it's fused to. Any mitochondria which is fused with a mitochondria will be referred to as it's fusee. Each mitochondria is reunited with all of it's fusees one after another.

Each mitochondria who is fused to other mitochondria tries to reconnect with each of them. This process involves a mitochondria pulling in all mitochondria who it's fused with, and this happens one at a time to each mitochondria in a loop until all are reconnected. For example if the order is 1, 2, then 3, mitochondria 1 is reunited with all it's fusees by way of each of them making a translation toward it. After that mitochondria 2 is reunited with it's fusees and so on and so forth. A successful reconnection is illustrated on the left side of Figure 4.8. In each movement shown, the mitochondria whose turn it is can be seen as the center of a pulling motion whereby it's fusees are drawn inward toward it.

In the case of simple networks of mitochondria where there are only two or three mitochondria fused together, this process is simple to imagine, and it works well in practice. One after another, each mitochondria pulls all mitochondria who should be fused with it toward itself the precise amount needed to make their fusion point pairs overlap. The hope is that eventually all fusion point pairs of all mitochondria are successfully overlapping at the same time. This is checked for and if true, this behavior is considered finished for the current frame and CODS moves on to the next task. Depending on the structure of the network, after each mitochondria has successfully pulled in it's fusees it can be that the network does not have all fusion point pairs overlapping. This happens because bringing one fusion point pair together can be undone by bringing another fusion point pair together. In other words, a subsequent mitochondria bringing it's fused mitochondria toward it can sabotage other mitochondria who previously tried to bring together it's fusion points. For this reason, CODS loops over the action of bringing together the fusion points several times. In some cases it can be that a loop forms where no matter how many times this is tried, there is always some fusion points separated in space. An example of this is shown on the right side of Figure 4.8. The numbers in the figure show which mitochondria is currently bringing all it's fused mitochondria toward it. The order that this happens stays the same and if this process repeats it can be seen that it will never be completed.

The solution is to change up the order in which the mitochondria bring in their fusees. The order with which CODS goes through each mitochondria to connect their fusion points is changed up to break down these loops which cause the process to fail. By changing the order that CODS looks at each mitochondria, there is more often success in reuniting all fusion point pairs. If a loop forms where mitochondria never fall into fusion point harmony, changing the order that they bring them together can cause them to successfully fall into place. In the broken example on the right side of Figure 4.8 if the order of the mitochondria was changed then it would fix the loop which causes them to never fully connect. In this broken example the order is bottom-left, bottom-right, top, middle. But if the order was changed to bottom-left, bottom-right, middle,

top, then they would all be fully connected and the process would be quickly completed.

The reason this solution was chosen where curve points are wiggled randomly and then fusion points are brought together by drifting entire sets of fusion points is one of simplicity. Another idea would be to move only the fusion point pairs together following each wiggle behavior. This would stretch the length of the mitochondria involved. The length of both mitochondria in a pair which have been fused would need to be calculated and held within the allowed elasticity. This could be done in a similar way to how the elasticity is respected currently. The way this is done is by repeatedly choosing a random configuration of curve points which have been wiggled some random distance from their original location between zero and the predefined wiggle intensity for that mitochondria. The wiggle intensity is chosen at random from within a range of parameter values set by the user. Different sets of wiggled curve points are chosen until a configuration is discovered which doesn't lead to a length change greater than the amount allowed by the elasticity. If two fused mitochondria needed to be wiggled at the same time in order to keep their fusion points together, this would require many more wiggle attempts than already required by the current elasticity requirements. This is because if a suitable configuration is found for one mitochondria, the other one might have wiggled too much and another attempt must be made. Not to mention if there were a third mitochondria attached to one of them then the wiggle would separate them from that third mitochondria. This would result in the need for wiggling all mitochondria in a network all at the same time making sure to not change their lengths too much. Testing the viability of this strategy is a possibility for future work.

## 4.12 Rotation Behavior

If a rotation speed minimum and maximum are set by the user, then a rotation speed will be chosen for each mitochondria and each frame they will rotate in the X Y plane around the center point of the mitochondria. A rotation direction is also chosen for each mitochondria, this decides if the rotation will be clockwise or counterclockwise. The rotation is decided by picking a random whole number between zero and one inclusive. If it's zero, then the rotation value assigned to that mitochondria is multiplied by -1. The coordinates of the axis of rotation are calculated as the average of the coordinates of the 4 curve points. Each curve point is simply rotated around the axis by some distance proportional to the rotation value assigned to the mitochondria. When a mitochondria fuses with another, their rotations are effected. If the seeked mitochondria is not already fused with any other mitochondria, then the seeker takes on the

rotation of the seeked. But If the seeked mitochondria is already fused with another mitochondria, then all mitochondria are set to have zero rotation. This is because a network of mitochondria all rotating together didn't seem realistic. If there are two mitochondria fused together and they have a rotation value assigned to them, then their eight curve points are averaged together for the axis of rotation so that they rotate as one large mitochondria.

## 4.13 Reduced Memory Approaches

CODS can be run in low memory usage mode for the purpose of running on a personal machine with limited memory. The reason this is needed is because of the PSF function [32] in ImSim [6] introduced in Preliminaries (Chapter 3). The PSF function is the way CODS blurs the points of light from the emitters to make the simulation more like blurry microscopy data. This PSF function holds very large variables in memory which when combined are often too large for the resources on a personal machine to handle. This low memory usage mode stores these large variables to disk while calculating the PSF function. This slows down video generation while dramatically reducing the memory needed. When one sample is being generated in low memory usage mode, a file is created to store temporary PSF function variables. This file is written to and read from during the execution of the PSF function.

There's another parameter that can be employed which instructs CODS to reach in and adjust the PSF function for the purpose of reducing the total amount of memory used at any moment in time. This parameter is called PSF together. The PSF function calculates the value of each pixel in the image which is determined by considering the light coming from the mitochondria and vesicles. The PSF together parameter can be set to the off value which makes CODS break up the PSF calculation into smaller chunks. This means the PSF function is only given the coordinates of the emitters of one single entity at a time. Each entity is calculated separately and a blurry image is produced for each one. These output images are added together to make one complete image containing all entities. It's less memory intensive to calculate the PSF for an image with fewer entities in it, so doing them one at a time spreads the memory usage out across more time. This is important when running CODS on a computer with limited memory.

During testing and troubleshooting of new CODS features, it's helpful to generate quick image data to get an overview of the behavior of mitochondria without needing crisp microscopy mimicking imagery. For this a parameter called mode is useful. It allows dramatically reducing the number of emitter points simulated. This offers very quick image generation with very little mem-

**Table 4.4:** Parallel Restrictions. Running the PSF function in parallel for the different entities in a frame has several disadvantages. Write disk is another name for the parameter sometimes called low memory mode where PSF variables are written to disk to use the least amount of memory at one time. Samples must be run sequentially because parallel processes cannot spawn more parallel processes.

| **If Parallel PSF is On:** | **If Parallel PSF is Off:** |
|---|---|
| PSF Together Must be Off | PSF Together Can be On or Off |
| Write Disk Must be Off | Write Disk Can be On or Off |
| Parallel Samples must be Off | Parallel Samples Can be On or Off |

ory required.

## 4.14 Parallel PSF

Setting the PSF together parameter to the off value and breaking the PSF calculations into chunks provides the opportunity for parallelization. If there are several mitochondria and/or vesicles in a video and if the PSF together parameter is set to off, then each of these organelles will have their own temporary PSF image data calculated and when they are all finished they are added together to make the full image which includes all the organelles together. Because the calculations for any given organelle doesn't require the result of any other, they can happen simultaneously by different processes quicker than if they are calculated one at a time. Enabling the Parallel PSF parameter activates this behavior and the time savings could be dramatic depending on how many organelles are in the frame.

## 4.15 Parallel Restrictions

If parallel PSF is turned on, this file is accessed by multiple parallel processes and this causes conflicts. For this reason, if CODS detects that parallel PSF and low memory usage mode are both turned on then CODS will turn off parallel PSF. However, this file is located within the specific directory for the video which is being generated, therefore there will not be conflicts if parallel samples is turned on as there will be a unique PSF variable file for each video. There are several factors that reduce the combinations of options available for the different modes. In particular there are certain conditions which prevent CODS from being run in parallel. These are shown in Table 4.4. If parallel PSF is switched off then the other parameters like write disk and PSF together can

be used in any combination.

## 4.16 Stationary Mitochondria

It can be useful when testing training data on a machine learning model to have known parameters that don't change dramatically. For this reason CODS includes a user provided parameter which allows for the creation of one or more stationary mitochondria. Stationary mitochondria don't drift, rotate, or interact with other mitochondria. This means they cannot be seeked, fission, or engage in seeking or fusion behaviors.

## 4.17 Example Behavior Profiles

Because it is possible to trigger a fission or a seek, custom behavior profiles can be created by modifying CODS. Three examples have been created which showcase some options available for a researcher to customize different behavior sequences. This could be helpful if someone wants to create specific kind of simulation data in order to train a model to recognize specific behavior in real microscopy video. The first example behavior profile is called network. If this is chosen by the user, then the mitochondria will have a series of eight seek behaviors triggered between frames 30 and 180. Depending on what values the user chose for probability of fusion and probability of fission-on-fusion, this will probably result in a video that shows mitochondria slowly linking up with each other, generally forming one or more networks of tangled mitochondria hanging off each other. The next example behavior profile is called dissolve. This involves 17 fission triggers between frames 170 and 330. This will result in a video which shows mitochondria being snipped into multiple pieces many times until there is a high number of quite short stumpy mitochondria. This is designed so it can be used together with the network behavior profile so that you can see a network form and then after that the network starts dissolving into little fragments of mitochondria. The third behavior profile takes a less deterministic approach. This behavior profile triggers fissions and seeks randomly and records specific data including midpoint coordinates of each mitochondria on every frame. Instead of triggering a fission in the same mitochondria on the same frame number on every video, it triggers seek and fission behaviors randomly. On average, there will be one fission and one seek behavior triggered every five frames. This is kept true no matter how many mitochondria there are, so when many fissions have taken place and the number of mitochondria have doubled, then the likelihood that any one mitochondria will undergo fission on a given frame is tuned down accordingly. This is accomplished according

to the following method. Every frame, CODS decides whether or not to trigger a fission or seek behavior on each mitochondria. The probability of triggering is calculated separately for each of these two behaviors, and it's equal to the following:

$$1/f/m \tag{4.4}$$

Where m is the total number of mitochondria which are not assigned to be stationary and f is the number of frames between each desired event. So if f is equal to 5, then the goal is to have one of each behavior triggered on one of the mitochondria every five frames on average.

## 4.18 Recording Video Attributes

While CODS is doing the calculations necessary to produce a video, it keeps the user updated about it's status. Several datapoints are included in this stream of information like how much time it took to generate each frame. While this is done to keep the user updated as video generation progresses, many more details are written to a text file for later reference to document these details more permanently. This includes all user inputted values and the random values generated for each mitochondria between the user inputted ranges for each video produced in a batch. This also includes mitochondria length changes, changes in the fusion status of the mitochondria, and the triggering of any behaviors and which mitochondria are involved in those behaviors.

## 4.19 Photokinetics

ImSim [6] simulates photokinetics by simulating the switching on and off of the fluorescent emitter points in the image using a formula to realistically reflects observed emitter photokinetics. When translated into video these effects effectively produce a striking resemblance to the flickering on and off of emitters on real subcellular organelles from one frame of a video to the next. This can be seen in any video produced by CODS but is less clear in the output of ImSim. This is because in CODS the video plays many generated images of mitochondria in the same location for easy comparison but in ImSim each image delivers mitochondria of different length and dramatically different shapes such that the differences in brightness in various regions of a mitochondria between images is more obscured.

## 4.20 Ensuring Randomness

When generating random numbers while experimenting with parallelization as part of this thesis work it became important to rule out the effect of the random number generator seed when troubleshooting CODS. A seed is a number which is used as the base for a random number generator to start from when choosing a random number. To reduce the chance of getting the same number when generating two random numbers it's best to use a seed that has the highest chances of being different. This was done in CODS by getting the number of miliseconds that have passed since the year 1990 and adding the processor number to that. This guarantees that a seed generated by two different parallel processes is different as well as guaranteeing that seeds generated seperated in time will be different.

# 5

# Example Scenarios

The purpose of CODS is to produce dynamic complex scenarios that mimic real microscopy data. This chapter will draw links between the output video data from CODS and the behavior exhibited by real mitochondria shown in microscopy data. The several behaviors that CODS is capable of simulating presented in Methodology (Chapter 4) will be shown here along with snippets of video from a fluorescence microscope. The video [5] that these snippets are taken from is from the draft version of a dataset which should be published with DataverseNO a few weeks after this thesis.

*Flickering.* The first notable thing about CODS videos is the flickering effect visible in the output of CODS. This flickering of the emitter points in CODS is due to photokinetics calculations which simulates the photon emitting particles switching off and on. These calculations are done by ImSim and they were discussed in Preliminaries (Chapter 3). This switching on and off of fluorophores is well documented [28] [29] [30] [31]. These emitters producing light only part of the time makes it so between frames different parts of the mitochondria will be lit up and different parts will be dark. This can also be seen in the microscopy data provided in the link below.

This video (Appendix C.1) shows real microscopy data in which a mitochondria undergoes fission splitting in half into two pieces and shortly thereafter fuses with a different mitochondria to form one bigger one.

This video (Appendix C.2) is CODS output video where a kiss and run has been

simulated. In a kiss and run, the seeker mitochondria arrives at the seeked mitochondria and then immediately retreats, drifting away at constant speed. The seek behavior is triggered after the start of the video. After the kiss and run, the seeker mitochondria drifts off screen. This showcases the ability of CODS to produce realistic video where organelles can be partially cut off from view as they would in real microscopy data and even disappear entirely.

*Wiggling.* The wiggle behavior produced by CODS bears a strong resemblance to mitochondria behavior. These papers discuss wiggling and shaking as one commonly observed behavior [34] [35]. In the links below you can see that the random directions and distances chosen by CODS to adjust the curve points in each frame leads to remarkably similar looking deformations happening in an erratic uncoordinated way.

In this microscopy video (Appendix C.3) the short mitochondria in the center can be seen suddenly changing shape to become a tight curve and then abruptly straightening back into a line while it drifts.

This CODS simulated mitochondria (Appendix C.4) exhibits the same propensity for experiencing perturbations which deform it back and forth between curvy and straight.

*Drift.* The drift is a fundamental building block of behavior in CODS. It is used to carry out more complex motions and by itself is a powerful tool for training models to expect movement in video. In the videos linked below the simplicity of the movement is clear. This makes it a good candidate for simulation. The behavior consists of a simple translation of the entire entity together in a direction by some distance from one frame to the next. The cumulative effect of a constant drift is a smooth motion from one location to another.

This video (Appendix C.5) of real world mitochondria as seen through a microscope contains multiple mitochondria drifting at different rates. This can happen when the medium that they are immersed in begins moving. But the drift motion is also an important building block of other behaviors such as seek. For this reason, it's wise to capture this motion by itself to train a model to expect it.

In CODS, the drift speed of each mitochondria is chosen at random from within a user provided range as explained in Methodology (Chapter 4). This allows for quite different drift speeds to be chosen for different mitochondria if a wide range is provided by the user. This can be seen in this video (Appendix C.6). This makes it very easy to provide variability into the training data which should allow a model to learn to recognize a wider range of situations in microscopy video.

*Seek.* The seek behavior brings a seeker mitochondria toward a seeked mitochondria. The videos below showcase various situations where this motion is applicable. In all examples it's the same basic movement. This behavior is vital to bring mitochondria into contact with each other which is important for content exchanges between them [1] [8]. Mitochondria perform directed motion with the goal of moving from one point to another within a cell [1].

In this video (Appendix C.7) a mitochondria can be seen in a mocroscopy video seeking out another mitochondria. It very intentionally moves in a direction directly toward the other mitochondria and after performing a kiss and run it eventually moves back to the seeked mitochondria and fuses with it.

This simulated kiss and run (Appendix C.8) showcases a seek behavior which features the seeker mitochondria presenting a point which is not an endpoint to the seeked mitochondria. An endpoint is the very tip of a mitochondria. Due to the way that curve points define a mitochondria's shape, the endpoint of a mitochondria is always the first or last curve point. This kiss and run happening at a middle curve point rather than an endpoint is intended to present a different scenario from the most common. This is important for preparing a machine learning model which is being trained on this data to handle a wide range of situations.

*Fusion.* Fusion is the next behavior shown in this section. To carry out a fusion behavior, a seeker mitochondria must first seek another mitochondria. Once they are united then a fusion can take place where both mitochondria merge into one large one. The example chosen for the microscopy clip here results in an obvious fusion where the resulting larger mitochondria is smooth and begins behaving as one single entity after the fusion. As explained in Background (Chapter 2), mitochondria can be stimulated to undergo both fusion and fission [10] and this behavior is critical to regulate their structure and maintain their even distribution throughout the host cell.

The mitochondria seen fusing in this video (Appendix C.9) is a bit more difficult to locate amid the busy scene. The mitochondria which performs the seek and fusion behaviors is a short quick moving one near the bottom of the video. It begins by moving to the right and coming close to the seeked mitochondria. It navigates downward toward the end of the seeked mitochondria and near the end of the video they fuse together and the longer resulting mitochondria advances upward.

The simulation shown in this video (Appendix C.10) is a basic seek and fusion behavior where the seeker mitochondria joins up with the seeked one. CODS doesn't consider the orientation of mitochondria when they seek or translate in any way so this video is unique in that it looks slightly more natural due to

the seeker mitochondria advancing along the direction of it's length instead of awkwardly moving sideways.

The fusion in this simulation video (Appendix C.11) ends with an interesting morphological change occurring. The seeker mitochondria curls up into a ball which looks surprisingly realistic given certain observations of real life mitochondria experiencing certain types of stress [12]. This feature of CODS producing surprisingly life like simulations has become a common occurrence throughout this research project.

*Kiss and Run.* The kiss and run behavior is another option after a successful seek has been carried out. If two mitochondria do not fuse together they may also simply not do that. This is described in this paper [36] as an interaction between the mitochondria where the outer layers are intermingled temporarily before they withdraw from eachother. The withdrawl happens before their structural integrity is violated, and so the behavior occurs while maintaining the original shape of the participants. In microscopy videos this is sometimes preceded by another kiss and run or a fusion behavior. Kiss and run behaviors have been observed in other types of organelles [33] as a way to exchange content, possibly for the purpose of moving resources within the cell. This type of behavior can be seen in CODS output videos when there is a quick succession of seek behaviors.

This microscopy video (Appendix C.12) is an example of mitochondria eventually performing a fusion behavior with it's seeked mitochondria. The mitochondria in question are located in the center of the video. The seeker mitochondria performs several kiss and runs before finally fusion takes place.

This CODS output video (Appendix C.13) shows a simulation of the seeker repeatedly performing kiss and runs. These can be triggered arbitrarily depending on the desired behavior profile in training data so this kind of repetition is not difficult to produce using CODS.

*Fission.* This paper [8] discusses the relevance of certain resources within the cell which play a role in fission and fusion. This is discussed in greater detail in Background (Chapter 2). This is another case where the output of CODS looks surprisingly close to the observed behavior of living mitochondria.

In this example (Appendix C.14) not only does a mitochondria undergo fission resulting in two smaller mitochondria, but it appears to happen multiple times producing a very small fragment that drifts away.

Due to the nature of how CODS orchestrates fission, this small fragment being produced (Appendix C.15) is common when two fission events happen in one

mitochondria.

*Fission-on-Fusion.* Fission on fusion is a fission behavior being effected causally by a fusion event. When a mitochondria fuses with another mitochondria it can be that there is a fission as a response to that.

This behavior (Appendix C.16) is a bit more difficult to identify. It's difficult to say for sure if one event was triggered by another event in videos like these but here are two possible examples of this happening. In this first video the action happens in the center of the screen. In this video (Appendix C.17) of a possible fission triggered by a fusion event, the area of interest is the bottom part where several shorter mitochondria take part in a brief dance.

This is a simulated video (Appendix C.18) from CODS showing a fission on fusion event. The probability of these occurring can be adjusted when generating videos with CODS. This event won't take place at all if a fusion happens on the endpoint where no other mitochondria are fused.

# 6 Applications

The broad goal of CODS is to become a useful tool which is easy for researchers to use that can help them more easily observe mitochondria and their behavior. The machine learning models described below have the potential, when trained on CODS data, to give researchers a much easier time counting and tracking mitochondria, telling them apart from each other, and generally observing their behavior. The power of this opportunity lies in the potential for use in a small specific area of interest and also throughout an entire cell. This could be a game changer for a researcher looking at a short moment in the life of a cell when a complex dynamic interaction occurs or it could be an important shortcut in summarizing the actions taken by mitochondria over a long duration microscopy video. Just this implementation presented in this thesis work could already make a difference because of the known impact mitochondria on the health of the host organism [1].

CODS has the ability to make several different kinds of records of the activity going on in the videos it generates. It also has several different options for example behavior profiles that are meant to showcase the kinds of videos that could be created using CODS.

Even in the early stages of CODS when it has begun to produce it's first test videos, other researchers in the department have requested access to this new data. This has been helpful already in several different areas explained in the sections below. Data that CODS produces has been an area of high interest within the project group already and the data is currently being used to launch

new projects using the data to train prediction tools, classifiers, and other such machine learning models.

## 6.1 Video Segmentation

Video segmentation is a more accurate version of image segmentation. Image segmentation is where each pixel in an image is categorized into one of a select number of classes. The point is to put the pixels that make up an object in the image into one class and leave out the background of the image. In video segmentation this is the same, but the method of getting to this end result can be produce more accurate results if you make use of the fact that it's a video instead of an image. This is notable because video data contains much more information about which specific pixels should belong to an object. The idea is to train a machine learning model to do what a human does when it looks at a video. This is to use neighboring video frames to gain important context about objects in the video which help classify them as being part of certain objects. Data that can help this includes for example which parts move together in the video and which edges disappear behind which other edges. The way to train a machine learning model to do video segmentation is to feed it training data which includes segmentations. CODS by default produces a perfectly segmented version of each frame of every video. This is because the purpose of CODS is to produce data for training machine learning models and one of the expected applications is video segmentation.

## 6.2 Triggering Behaviors Randomly

One behavior profile involves setting a probability of fission on each frame and a probability of seek on each frame. If the probability of fission is set to 50 percent and there are two mitochondria in the video, then each mitochondria will have a 25 percent chance of engaging in a fission behavior on any given frame. This makes it so there is on average a 50 percent chance that some mitochondria will engage in a fission behavior on a given frame. This percentage is adjusted down for each mitochondria during every fission behavior to keep up with the rising number of mitochondria. This behavior profile has been particularly useful in generating data for training video segmentation models. It was unexpectedly useful in creating randomized data in which behaviors were triggered at random times. The original purpose of constructing this profile was actually to help one colleague in the project group capture data from CODS that would help train a model on CODS video meant to count the number of actions taken on each frame.

## 6.3 Counting Behaviors

If a model watches the behavior of mitochondria in a video, it should be able to identify when an action takes place. One approach to achieve this is to train the model on the number of behaviors that were taken at each frame number. The goal to land on the correct number of actions occurring on a frame, the model will arrive at a high proficiency at detecting behaviors by observing different aspects of a video that contains different numbers of behaviors in certain frames. This model tries to observe the number of events in a video that happen on each frame and then counts the number of them as the main output of this solution. To help accomplish this goal, this profile in CODS also creates a CSV file and every frame it notes down the coordinates of the center line center point which lies in the center of each circle of emitter points in each mitochondria. It also records the total number of fission behaviors and seeks triggered during each frame. The coordinate data is saved in a particular format. Normally during a fission, one of the mitochondria resulting from the fission is given the old id number of the mitochondria which fission into two parts and the other mitochondria is given an id number equal to one plus the current highest mitochondria id number. But the data saved by this profile uses a different id number scheme where during a fission both resulting mitochondria get assigned new id numbers, and the old id number is left being assigned to no mitochondria. This data was used to test if CODS can create viable data to use to train an action detection machine learning algorithm that can classify events like fission behaviors and fusion behaviors in mitochondria.

## 6.4 Natural Language

Natural language information is highly desirable. Imagine a model that can print plain English text that describes the events in a complex video. If this was possible, researchers wouldn't need training to interpret the output of such a model. This makes a big difference not only in cutting the cost of extra training but also this ease of use offers an improvement in the experience of anyone using the tool. Use of CODS for such a purpose has been requested by a colleague in the project group. CODS now includes the ability to generate natural language data for the purpose of running some initial tests for this work. This is not a polished functionality yet and the parameter that activates this ability is not yet available in the graphical user interface but it will be soon. However, as this natural language processing project is still internal, initial data generation and testing is possible and this has already begun.

## 6.5 Entity Tracking

Tracking is a key part of any of these potential models that could benefit from CODS. Any segmentation model or natural language model would need to keep track of which mitochondria are doing what on the screen in the background. This requires some understanding that the entity in one frame is the same as the slightly distorted and translated entity in nearby frames. Some models specialize in getting the tracking part right as their main deliverable. This is currently being worked on by a different colleague within the project group as an idea to solve one part of this machine learning pipeline. An experiment is ongoing to train a machine learning model to add a bounding box around mitochondria as they move around in video. As part of this training experiment, a variety of data is being generated with CODS. To achieve a diverse dataset, CODS parameters are being adjusted between different batches of data produced. To create different scenarios, parameters like the following are being modified:

- The number of mitochondria.
- Including and excluding different combinations of seek, fission and fusion behaviors.
- Drift speed and seek speed.
- Wiggle intensity.
- Position in the Z dimension (amount in or out of focus).

## 6.6 Recording Skeleton and Curve Points

A profile called record coordinates will write the coordinates of all curve points and all center line skeleton points to a CSV file. This is a large amount of data. Writing this much data to disk every frame with the file growing larger and larger each frame becomes more and more time consuming. For this reason, for this profile and the random actions and center points profile CODS writes data to disk in batches of several hundred frames. This means CODS needs to edit these files much less frequently and simply saves the intermediate values to memory in the meantime. If the number of frames is not divisible by the number it waits for to save data, then it saves the last chunk of data after generating the last frame.

## 6.7 Network Example Behavior Profile

The network profile checks if the current frame being generated is equal to a list of numbers and if yes then it triggers a seek behavior involving arbitrary mitochondria. This results in a video that shows mitochondria begin periodically seeking eachother, and assuming the user provided value probability-of-fusion is somewhat high, the mitochondria start forming network(s). This often looks like a tangled mess of worms but it can be interesting when combined with the dissolve example behavior profile.

## 6.8 Dissolve Example Behavior Profile

When Dissolve is switched on, the current frame is compared with a list of frames that begins in the hundreds and if there's a match, it triggers a fission behavior in an arbitrary mitochondria. This results in a video where after some time mitochondria begin to be chopped up into little bits which drift out into space. When combined with the Network profile, the video showcases the networking of mitochondria and then the network dissolves into small chunks before your eyes. This is not an immediately useful data, however it does a good job showcasing to a researcher wondering how to create his own behavior profile which generates very specific behavior sequences for use in training a particular model how to do this. The code written to build these example profiles can be learned from quickly to see how behaviors can be triggered to target specific mitochondria and specific curve points on those mitochondria.

## 6.9 Recording Video Attributes

Every time CODS runs, it produces a text file called attributes. It produces one of these files for each sample. This includes the values of all attributes including the randomly generated ones for each mitochondria. This way you can go back and see exactly which parameters were used in generating the output of a run of CODS. This file also includes a record of every time the length of a mitochondria is changed due to a fission behavior. This record includes the frame number where this occurred as well as which mitochondria this happened to and the length value itself. This record keeping also happens with the data regarding which mitochondria are fused with which other ones, this way you can see this variable be updated whenever a fusion is triggered or terminated. Whenever a seek behavior is triggered or terminated this is noted as well keeping careful note of the frame number and the mitochondria

id numbers involved. Attributes.txt also includes the amount of time taken to generate each frame along with the timestamp as well as the percentage of this time which was devoted to only the PSF function because this is a very time consuming function and often takes over 90 percent of the run time of a frame.

# 7

# Results

## 7.1 Experiment Background

The highlighted use case for the purpose of this thesis research project is video segmentation. The stated goal of CODS is to produce data for use in machine learning. To prove this data is useful for training a model, it seems appropriate to select a variety which is similar to image segmentation which was what ImSim was proved on. MMTracking [37] is the tool chosen to provide the video segmentation functionality for testing CODS data. MMTracking allows video instance segmentation via a machine learning model called MaskTrack RCNN. In MMTracking there is training and testing functionality for data of the same format as the youtube VIS dataset [38]. This means that in order to accept data into the model, it must be organized into a specific directory and file structure. Training a model involves a set of parameter values called weights which get adjusted continuously throughout a training process in response to the kind of data fed to a model. For computer vision models such as this one, that means the computer makes tiny adjustments to a huge set of these weights depending on the pixel values in the frames of a video. This is a complex process and takes into account an object identification system that tries to keep in mind the identities of objects throughout the frames of a video, as well as the exact pixels which make up an object in the segmentation part of the model, and the number of pixels these objects inhabit called the area, and bounding boxes in the form of rectangle coordinates which surrounds these objects. This training process involves teaching the model that for a particular input, a certain answer is correct. Sometimes this answer is a simple class like "this is a dog" or "this is a

cat" such as in classification algorithms, and sometimes a more complex answer type is aimed for such as in segmentation where a binary mask consisting of an entire video of pixel values is the correct answer to be aimed for by the model. The binary mask referred to here is exactly the black and white image types included in the output frames of CODS videos. In MMTracking, the default behavior of the video instance segmentation pipeline is to use a pretrained model downloaded from openmmlab.com. This model has had it's weights adjusted already in response to a dataset of third party videos. The testing pipeline in MMTracking involves using your trained model with all of it's weights set in place to observe new data and make inferences about it. This is where the model guesses what the correct answer is for a particular piece of data. In this case the model looks at a blurry microscopy video and tries to do video segmentation on it. This means generating an output video which contains a binary mask overtop the original video such that the objects of interest are included in the foreground of the mask, and everything else is part of the background. In the case of CODS masks, the black pixels are the background, and the white pixels which are over the organelles are the foreground objects of interest.

To test CODS data by training a model with MMTracking, certain extra data is needed to be compiled about CODS data to assist MMTracking in training. This is part of that particular data directory and file structure needed to be matched exactly, of the same format as that dataset from youtube VIS [38]. JSON format type files are needed which encode the ground truth binary masks into uncompressed RLE representations in the form of long lists of numbers. These numbers encode the number of one color of pixel in a row before a change in color. In a binary video frame image where there are only two colors of pixel this approach is a useful way to describe an entire image using a list of numbers. Also required in the JSOS files is the definition of a bounding box for each object in the video at each frame and the area of the object which is just one number that counts the number of pixels in the foreground of the mask. MMTracking expects the JSON files to have a specific organization. In the case of a video containing more than one object of interest, the masks for each object need to treat the pixel locations of all other objects of interest as background pixels. This is unfortunate due to the type of ground truth data available from CODS. To capture the power of CODS simulations in these data submitted to the model, videos of multiple mitochondria interacting would be necessary. In it's current form, CODS ground truth mask videos include white pixels for all organelles in a frame of the video in the same mask image for that frame. Because sometimes mitochondria overlap while interacting, the overlapping zone is indistinguishable from non overlapping zones in the CODS binary mask output. This is because areas which include an organelle are pure white pixels, so two overlapping organelles also simply mean pure white pixels. This would be an easy fix to change how CODS creates output data if not

for the time constraint of this thesis. What could be done is to save a ground truth mask image for each organelle where only that one organelle is painted in white. The only way to use the binary masks from CODS data without further modification to create these JSON files while being faithful to the exact locations of mitochondria in the videos is to encode the RLE values in the JSON file that consider all white pixels to be part of the same object. This would teach the machine learning model that a video containing five mitochondria moving around actually contains one object of interest. This is not conducive to individual identity tracking or behavior such as fission and fusion detection. It's also harder to imagine it working successfully as a video segmentation strategy as this switch from treating a group of objects as individuals to treating them as one entity would be a major departure from the convention. Therefore the path chosen is to only use videos containing one mitochondria in the training phase of this experiment.

A few concepts are needed before continuing, namely transfer learning, batch sizes, and epochs. Transfer learning is the act of training a machine learning model on one type of data, and then using those weights achieved by that training to further train a model on more data. This effectively resumes training from the saved checkpoint arrived at after training in the first dataset and it can work as a kind of fine tuning where the weights are adjusted less than in the first round of training. This is a useful concept when considering that large datasets exist already which can get your model to a point of some understanding of a domain like using vision to recognize the edges and shapes of objects for example. One way that transfer learning was used in ImSim [6] was to first train a model on data mimicking one type of microscope configuration, and then using somewhat less data mimicking a different microscope to continue training the model to fine tune it. The result of this was that the model could successfully recognize mitochondria and vesicles in a range of different microscopy videos of different types. The next concept we will visit is epochs and batch size. Training works by adjusting the weights of the model over and over again for each piece of data, and it goes over the entire dataset more than once to attempt to arrive at the best results. Each time the model runs through the entire dataset, this is called an epoch and this can be adjusted when launching a training algorithm. Running through the entire dataset many times can take time, and to do this faster, more than one piece of data is processed at the same time and the number of these pieces processed in batches is called the batch size. Adjusting the batch size is one of the many ways to fine tune a training regime and like all the others it has effects on the performance that can be hard to predict. Batch size has an effect on something called overfitting which is when a model is only good at guessing the answer to a very specific kind of data and might not perform well on different kinds of data.

## 7.2 Experiment Configuration

To get the best results, eight configurations of settings are tested as different training profiles on CODS data. The idea is to look at the results from each of these eight configurations and choose the one that exhibits the best performance. The performance will be judged by examining the output which will consist of videos with the inferred segmentation masks overlaid atop them. The data which the inference is done on is not included in any data used for training. This testing data is four simulation videos generated by CODS of various varieties including videos which include multiple organelles, as well as eleven clips of video from real microscopy data. This gives a good idea of whether the model can segment and track mitochondria across frames in video data. The training dataset consists of 48 videos which each have one hundred microscopy mimicking frames. Included of course is the segmented ground truth copy of each frame. This data is generated by CODS. In addition to the videos, JSON files are fed into the MMTracking system as well which include uncompressed RLE representations of each binary segmentation of each frame of each video as discussed in Experiment Background (Section 7.1). Also as discussed in Experiment Background, the training data only includes one mitochondria in each video. Two categories of tests are performed, the first category is a training regime where the default pretrained model is trained on a huge dataset of three thousand videos from the youtube VIS dataset [38], and then this trained model is fine tuned on CODS data. The second category is a regime where the pretrained model is trained directly on CODS data without first training on these three thousand videos from the internet. These two groups are divided into two more groups each, one group using the default batch size automatically selected by the system based on the inputted data, and the other group using a batch size of one. These four groups are divided once more into groups that are trained on CODS data across three epochs, and ten epochs.

## 7.3 Experiment Results

The results of the inferences are binary masks overlaid onto the original videos. An example of what is meant by this can be found in Figure 7.1 where the colorful shapes superimposed over top the images are what is referred to here as masks. I was unable to get the evaluation tool working within MMTracking for generating numerical performance evaluations such as the IoU. This could be related to the lack of segmentations for all of the the testing data which included mostly unsegmented real microscopy data. In lieu of the most objective form of evaluation metrics, this section will instead present observations of the segmentation inference output consisting of videos with these masks overlaid on top of them. Inferences are performed on four videos of simulated CODS data

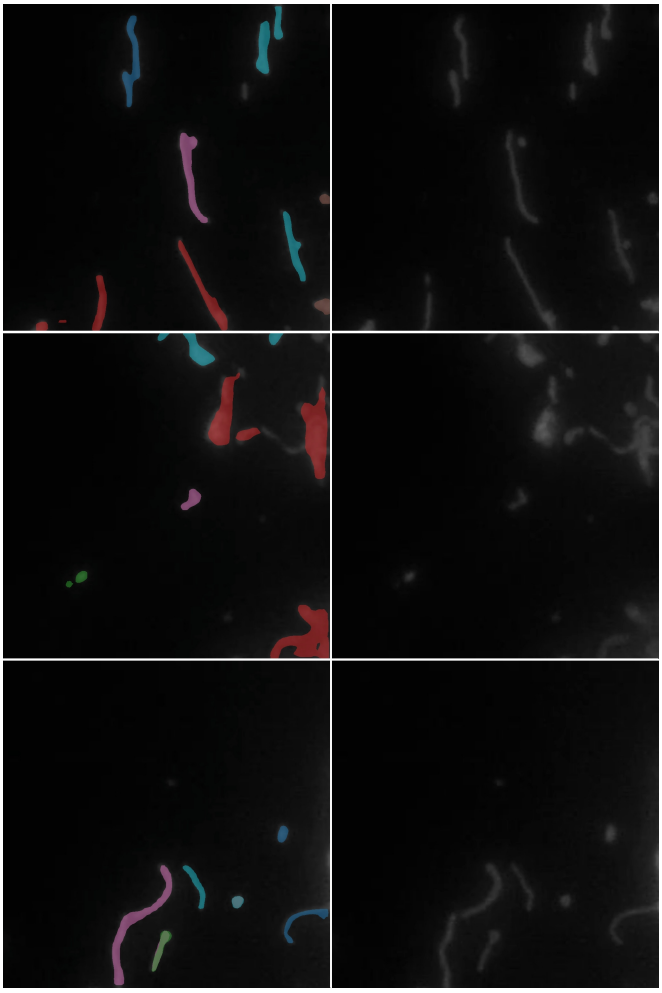

**Figure 7.1:** Example Frames From Inferences. Inferences are the left side and original frames on the right. Different colors are unique individual objects of interest.

(Appendix Sections C.18, C.11, C.13, and C.8) and seven snippets of fluorescence microscopy videos of mitochondria (Appendix Sections C.5, C.1, C.14, C.12, C.9, C.7, and C.3). These seven snippets are obtained by zooming in and cropping this video [5]. The performance criteria for judging the best model are color, dark time, and shape.

*Color.* Continuity in color is examined to check for tracking ability. This means watching for sudden changes in the color of the mask on an organelle. The color of the mask signals the id of the organelle. So if an organelle has a blue mask for part of the video and then it changes to a green mask, then the model believes the first organelle has disappeared and a new one has replaced it. A

video which features an organelle with a mask whose color flickers between colors will be considered to have worse performance than a video where the mask of an organelle stays a constant color throughout the video.

*Dark Time.* If a mask vanishes from the video even though the organelle is still there, this is not a good sign. This means the model believes the organelle has been hidden from view and is no longer present in the video. An example of dark time can be if an organelle never has a mask added to it. If there are many organelles that the model doesn't apply masks to then the model doesn't recognize them as objects of interest at all and this would be an example of an opportunity for improvement. Another example is if an organelle's mask vanishes for one or more frames. This metric, like the others, is relative. It's used as a tool for comparison. If a video features an organelle's mask flickering on and off slightly more than an organelle in a different video, then the first video can be said to have worse dark time.

*Shape.* The shapes of the masks are observed and checked against the actual shapes of the organelles. If a large clump of individual mitochondria overlapping each other is segmented as one large continuous blob instead of a more accurate tangle of elongated shapes, then this would be a point counting against that model configuration. A video can be said to have better shapes if the masks fit tightly to the edges of organelles. Masks that bridge the gap between organelles and classify multiple seperated organelles as one identity are scored worse than masks which apply a different color to each organelle.

The following list includes links to the output inference videos and their real microscopy data counterparts.

- Drift with masks: C.19, without masks: C.5
- Fission and Fusion with masks: C.20, without masks: C.1
- Fission Multiple with masks: C.21, without masks: C.14
- Kiss and Run and Fusion with masks: C.22, without masks: C.12
- Seek and Fusion with masks: C.23, without masks: C.9
- Seek and Kiss and Run and Fusion with masks: C.24, without masks: C.7
- Wiggle with masks: C.29, without masks: C.3
- Simulated Fission on Fusion with masks: C.25, without masks: C.18

**Table 7.1:** Results. Top compares a batch size of one (1) vs the default batch size (D). Middle compares high (High) vs low (Low) number of epochs. Bottom compares with (3K) and without (No3K) 3000 pretraining videos from youtube vis dataset [38].

| | **High 3K D vs High 3K 1** | **Low 3K D vs Low 3K 1** | **High No3K D vs High No3K 1** | **Low No3K D vs Low No3K 1** |
|---|---|---|---|---|
| **Drift** | Identical | Identical | Identical | Default is much better at color. Default has better shapes. |
| **Fission and Fusion** | Identical | Identical | 1 has better shapes. | Default has less dark time. 1 is better at color. |
| **Fission Multiple** | Identical | Identical | Default is less consistent between frames. 1 has less dark time. | Default is slightly better at color. |
| **Kiss and Run and Fusion** | Identical | Identical | Default is less consistent between frames. 1 has less dark time. | Default is slightly better with dark time and color |
| **Seek and Fusion** | Identical | Identical | Identical | Identical |
| **Seek and Kiss and Run and Fusion** | Identical | Identical | Identical | Identical |
| **Wiggle** | Identical | Different but pros and cons. batch size Default is better at a short mito, and batch size 1 is better at a tiny mito. | 1 has better color. 1 has less dark time. | Default has less dark time. |
| **Simulated Fission on Fusion** | Identical | Identical | Identical | Identical |
| **Simulated fusion** | Identical | Identical | Identical | Identical |
| **Simulated Kiss and Run Multiple** | Identical | Identical | Identical | Default has better color. |
| Simulated Kiss and Run NonEndpoint | Identical | Identical | 1 has slightly less dark time | 1 has better color |
| **Winner** | **High 3K D** | **Low 3K D** | **High No3K 1** | **Low No3K D** |

| | **High 3K D vs Low 3K D** | **High No3K 1 vs Low No3K D** |
|---|---|---|
| **Drift** | Identical | Low No3K has better color. Low No3K has less dark time |
| **Fission and Fusion** | Low 3K slightly better at shape. High 3K slightly less dark time. | Low No3K has less dark time. |
| **Fission Multiple** | Identical | Low No3K is better at color |
| **Kiss and Run and Fusion** | High 3K is better with less dark time. | Low No3K has less dark time |
| **Seek and Fusion** | Low 3K has more dark time | Low No3K has less dark time |
| **Seek and Kiss and Run and Fusion** | High 3K has slightly better shapes. | Low No3K has less dark time |
| **Wiggle** | Low 3K has much less dark time | Low No3K has less dark time. High No3K has better color |
| **Simulated Fission on Fusion** | High 3K has slightly better shapes. High 3K has slightly better color. | High No3K has twice as much dark time |
| **Simulated fusion** | Identical | Identical |
| **Simulated Kiss and Run Multiple** | Identical | Low No3K is slightly better at color |
| **Simulated Kiss and Run NonEndpoint** | Identical | High No3K is better at color |
| **Winner** | **High 3K D** | **Low No3K D** |

| | **High 3K D vs Low No3K D** |
|---|---|
| **Drift** | Low No3K has less dark time (But High 3K detected a small mitochondria that Low No3K did not. And High 3K detected the network cluster as an object more than Low No3K) |
| **Fission and Fusion** | Low No3K has less dark time and slightly better color. |
| **Fission Multiple** | Low No3K has better shapes. High 3K has slightly better color. High 3K has less dark time on complex shapes but it just lays a blanket mask over them. |
| **Kiss and Run and Fusion** | Low No3K has better shapes. Both have better dark time on different mitochondria. High 3K has less dark time on networks but just uses blanket mask over it. |
| **Seek and Fusion** | Low No3K has better color. Low No3K has better shapes. Low No3K has less dark time on small mito. High 3K has better network coverage. |
| **Seek and Kiss and Run and Fusion** | Low No3K has better shapes. High 3K has less dark time on one mito. High 3K is better at putting masks on super long mitos. |
| **Wiggle** | High 3K has better color. High 3K has more dark time |
| **Simulated Fission on Fusion** | High 3K has less dark time. Low No3K has slightly better color. |
| **Simulated fusion** | Low No3K has slightly worse color. Low No3K has better shapes. |
| **Simulated Kiss and Run Multiple** | Low No3K has better color |
| **Simulated Kiss and Run NonEndpoint** | High 3K has slightly better color |
| **Winner** | **Low No3K** |

- Simulated Fusion Curve into Doughnut with masks: C.26, without masks: C.11
- Simulated Kiss and Run Multiple with masks: C.27, without masks: C.13
- Simulated Kiss and Run Non-Endpoint with masks: C.28, without masks: C.8

As can be seen in Table 7, the configuration which has the best performance after evaluation is the one which used the lowest number of epochs, and the default batch number, and which was not pretrained on the three thousand videos from the youtube vis dataset [38].

## 7.4 Discussion

In the Drift Inference video (Appendix C.19), the Kiss and Run and Fusion Inference video (Appendix C.22), the Seek and Fusion Inference video (Appendix C.23), and to a certain extent the Seek and Kiss and Run and Fusion Inference video (Appendix C.24) the model can be seen making attempts to segment complex mitochondria networks, but it does a poor job of this. It creates large patches of color and doesn't really know what it's looking for. And in the Fission Multiple Inference video (Appendix C.21) the model doesn't even make such an attempt. It simply leaves the network dark and focuses on segmenting the small individual mitochondria. The model also struggles with this rather simple Simulated Fission on Fusion Inference video (Appendix C.25). In this video, two simulated mitochondria form a knot with each other as they fuse and a part of one of them is fissioned off and drifts away. The model gets so confused that it drops the masks entirely for many frames, leaving them dark. The issues brought up here are certainly due to the training data only including videos of single mitochondria. Solving the road blocks associated with using multiple mitochondria in the same video as testing data as discussed in 7.1 would be an obvious next step in the evaluation of CODS in the future.

On the other hand, given a very small sample of extremely limited training data, the model has done some rather extraordinary things. Take the Simulated Fusion Curve into Doughnut Inference video (Appendix C.26) for instance. Several frames of this video can be seen in Figure 7.2. This video shows a mitochondria approaching another mitochondria. Initially they are identified as separate entities but when they fuse they suddenly have the same color. This shows a very high potential for this data to train diverse behavior detection that Isn't included in the training data. This accurate segmentation is common in the inference data in Appendix C.

**Figure 7.2:** Frames from the Simulated Fusion Curve into Doughnut Inference video (Appendix C.26). This video shows high accuracy on the part of the machine learning model. It also shows the ability to correctly track organelle identities across frames and then make adjustments when changes occur such as this fusion behavior. This is despite the model not being trained on any such behaviors or remotely related situations.

If CODS data with multiple mitochondria in one video is used to train a machine learning model, this could provide different results. An easy extension of these results could be to train a model which has the ability to classify behaviors seen in video, this would be an easy next step to test CODS data on and see what kind of configuration parameters lead to the best results. However, some work would need to be done finding a model that accepts this kind of classification information for behaviors. This might come in the form of event detection where a frame number or range of frame numbers would be specified and the behavior would be defined. This kind of information would need to be put into the correct format to feed into the model and that would take some work as well. This is a prime example of something for future work which will be discussed more in the next section (Section 7.6).

MMTracking [37] is chosen to be used for testing CODS data for several reasons. Because video segmentation is chosen as the use case to be tested in particular for the purposes of this research project, a model needed to be found which fulfilled several criteria. The machine learning model which would be trained

on CODS data needed to be capable of doing this kind of computer vision in the first place: video segmentation. This model also needs to be easy to install and use for a beginner without experience in the practical use of machine learning tools. Another important requirement used in the consideration of many tools to do the job is that it must be freely available to download, modify and use for research purposes. This ability for modification became necessary in the end when several bugs were discovered and some editing of the source code required to access the functionality of MMTracking for this project.

In this thesis, a heavy amount of work is put into developing the CODS simulation engine. The high effort put into debugging interesting behavioral issues with fission and fusion has come a long way in providing a platform that can be relied upon to generate data with minimal glitching or bugs. Several bugs that have evaded efforts to fix them until now can be found in Appendix B. More resources in this thesis could have been directed toward the testing and evaluation process. Training more machine learning models could have been prioritized more as well as running more testing regimes on all of them. One quirk of the use of MMTracking in this thesis is it's hiccups related to evaluation in the video segmentation module that is used as part of this thesis. Several of the features related to providing evaluation results and validation are not functioning either because of simple bugs causing runtime errors, or sometimes because the dataset it was designed for youtube VIS [38] simply didn't include segmented masks in the validation dataset, and so it didn't include the feature of validating results after each epoch. However, the test results show that the answer to the research question at the beginning of this thesis is a resounding yes. Such a tool can be built which can show concrete proof that it can make a difference in the field of bioinformatics. This thesis is a proof of concept that physics based simulation supervised machine learning training and testing with video data works on the microscopic level.

One step that can be taken to make models trained with CODS data more versatile can be to train them with multiple different configurations of the microscope parameters discussed in Execution of CODS Overview (Section 4.1) and shown in Table 3.2. This could be done via transfer learning and it might provide the ability to recognize mitochondria in microscopy videos using microscopes with different configurations or even different types of microscopes all together.

Mitochondria are important for cell function, as discussed in Background (Chapter 2). Models trained on carefully designed CODS data which make use of it's wide range of customizability have the potential to provide real help to researchers studying mitochondria. If this tool is picked up by researchers with an expertise in machine learning methods, they can use CODS to produce a high amount of data for only the cost of running the computers required to

generate it, and it will be a simple task to implement this trained model into a tool that can provide quick inferences on real data in real time.

There has been a lot of research done trying to peer deep into living cells. There have been developed several ways to add photon emitting particles to subcellular organelles and this technology stretches back many years [16]. Using computer simulations to better observe these structures is a younger area of interest and using video simulations to train machine learning models is in it's infancy. CODS provides some of the building blocks to begin to better understand the potential of this field. This is the goal of this thesis work. CODS and the research it represents can be easily developed further. CODS is designed to be easily taken further and there are many goals still yet unfulfilled that are outside the scope of this work but that nonetheless be an improvement. This will be discussed in greater detail in Future Work (Section 7.6).

CODS has been presented in this thesis as a solution to answer the question: Can a tool be built that shows promise in providing benefit to the field of subcellular organelle observation? CODS has shown results in training a tool that can observe these tiny machines. MMTracking [37] provided the means to test the data and ImSim [6] provided the image generation functionality. CODS expanded the concept of still images into motion video arena. This work builds on the work of related research taking inspiration from other simulators. Absorbing concepts from other research work in the area plays a critical role in the development of CODS and this project couldn't exist without the contributions of researchers in biology, computer vision and microscopy. The CODS system succeeds in providing a wide array of complex dynamic mitochondria behaviors to fine tune into more usable data in ways that cannot be appreciated until ever more creative ideas are applied to combine the power of CODS data with new machine learning techniques.

## 7.5 Conclusion

With the advances recently in machine learning methods and analysis tools, excitement has been growing around what this can mean for the traditionally hard to crack microscopic realm. Many tools are being created as discussed in Background 2. Simulations are one area of recent advancement that has shown viability for reducing down complex real life data into human understandable summaries that are useful to research. The development of CODS adds a datapoint along this journey of the creation of better and better tools to help visualize microscopy data and it has the potential to spark additional inspiration in the field. This kind of data processing is crucial in this age of massive data availability. To harness this data it's important that we have tools

like this to extract a variety of meaningful interpretations that can be used to assist in the ever growing knowledge seeking missions around the world.

CODS is able to generate high quality simple simulations with three or four mitochondria on a small personal laptop due to the work done designing low memory modes of running. It can be run on a large server network in parallel and produce high amounts of larger complexity data extremely quickly due to the work done on parallelization and optimization. This is discussed in more detail in the Appendix A. CODS shows promise in it's goal of providing high quality training data for machine learning methods like segmentation and it has produced interest from several researchers in the project group where it is designed.

There is a lot of work to be done both on testing CODS and also in using those results to do rounds of fine tuning that can help it excel where it shows weakness. This first round presented in this thesis shows that CODS output needs adjusting to fit a wider range of use cases. By modifying the output data format in several key ways it could provide much higher levels of ability for scrutiny via testing procedures involving machine learning models. If the binary masks produced by CODS were available in several different formats anticipating a variety of machine learning model dependencies, that would be influential. Currently the masks include all organelles in one frame image, whereas they could be separated as well into several image frames with only one organelle's mask in each. This would allow training to happen on CODS data including more than one organelle per video and that would quickly open up the evaluation regime to behavior detection such as fission and fusion.

Initial tests on CODS data have shown that it can be used as as a training data producer. Not only that, but these results conclusively prove the hypothesis of this thesis work: that the challenges in analysing microscopy data can indeed be tackled using dynamic complex simulations of the objects of interest. This is discussed in more detail in Discussion (Section 7.4). This training data is useful in this demonstrated case as simulation supervised training. This combines the power of physics based simulation with the adaptability of deep learning. Several other use cases have been outlined and can be found in Applications (Chapter 6). These have a direct path using CODS data to help prospective researchers run experiments using these methods to accomplish training on machine learning models that have different particular goals but use the same simulation engine and machine learning concepts to get there.

CODS in it's current form is very powerful in it's ability to produce endless dynamic complex situations between mitochondria that can be highly customized. CODS is a powerful tool that should be further tested to see just how accurate models can become from CODS training data. CODS establishes a foundation

for this area of research to jump off from. It's exciting to have the opportunity to watch this field take further strides in years to come. CODS in particular has a long way to go, and several points are highlighted for short term improvement in the next section (Section 7.6).

## 7.6 Future Work

CODS can be expanded to include other organelles besides mitochondria, and this would be a very interesting direction to go in with this project. Currently vesicles do not have the ability to move between frames. This can be added by copying many of the same functions that exist for mitochondria. For example the drift function and seek functions would look very similar. There are many plans for the vesicles to interact with mitochondria and induce behaviors within them such as fission behaviors for example. The vesicle motion patterns described CVPR paper [7] should be implemented in the next version of CODS.

One behavior that should be captured in this model is the flip manoeuvre. This should be built in the near future as it is among the original feature aspirations for CODS. There are not many videos showing this behavior but an attempt could be made to model it and see if it's produces results in classifying this behavior. One way this could be done based on one video of a mitochondria performing this action would be to move one of the endpoints in an arc which begins by moving away from the neighboring curve point. The rest of the curve points could follow along in a train behind this leader curve point and this would produce the sliding motion you see in real microscopy video.

One feature that should be built is the ability to choose which color the vesicles and mitochondria should be in the blurry PSF images. It is standard to capture different colors for these organelles in microscopy images. Many more feature improvements can be found in Bugs (Appendix B). There is interest in pursuing CODS further in the future and it will be interesting to see the direction it goes and the progress it makes going forward.

# Bibliography


[1] S. A. Detmer and D. C. Chan, "Functions and dysfunctions of mitochondrial dynamics," *Nature Reviews Molecular Cell Biology*, vol. 8, pp. 870–879, Nov 2007.

[2] E. Roesch, J. G. Greener, A. L. MacLean, H. Nassar, C. Rackauckas, T. E. Holy, and M. P. H. Stumpf, "Julia for biologists," *Nature Methods*, Apr 2023.

[3] A. R. McDonald, R. Roberts, J. R. Koeppe, and B. L. Hall, "Undergraduate structural biology education: A shift from users to developers of computation and simulation tools," *Current Opinion in Structural Biology*, vol. 72, pp. 39–45, 2022.

[4] A. M. Wright, R. S. Schwartz, J. R. Oaks, C. E. Newman, and S. P. Flanagan, "The why, when, and how of computing in biology classrooms," *F1000Research*, vol. 8, 2019.

[5] I. Opstad, "Fluorescence microscopy videos of mitochondria in H9c2 cardiomyoblasts," 2023. Filename: 20210611_H9c2-dTag_CCCP10uM_1518_conTL_002_ALX_PRJ doi: 10.18710/11LLTW url: https://doi.org/10.18710/11LLTW.

[6] A. A. Sekh, I. S. Opstad, G. Godtliebsen, Å. B. Birgisdottir, B. S. Ahluwalia, K. Agarwal, and D. K. Prasad, "Physics-based machine learning for subcellular segmentation in living cells," *Nature Machine Intelligence*, vol. 3, pp. 1071–1080, Dec 2021.

[7] A. A. Sekh, I. S. Opstad, A. B. Birgisdottir, T. Myrmel, B. S. Ahluwalia, K. Agarwal, and D. K. Prasad, "Learning nanoscale motion patterns of vesicles in living cells," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[8] M. Karbowski and R. J. Youle, "Dynamics of mitochondrial morphology in healthy cells and during apoptosis," *Cell Death & Differentiation*, vol. 10, pp. 870–880, Aug 2003.

[9] M. Giacomello, A. Pyakurel, C. Glytsou, and L. Scorrano, "The cell biology of mitochondrial membrane dynamics," *Nature Reviews Molecular Cell Biology*, vol. 21, pp. 204–224, Apr 2020.

[10] N. Zamponi, E. Zamponi, S. A. Cannas, O. V. Billoni, P. R. Helguera, and D. R. Chialvo, "Mitochondrial network complexity emerges from fission/-fusion dynamics," *Scientific Reports*, vol. 8, p. 363, Jan 2018.

[11] M. P. Viana, S. Lim, and S. M. Rafelski, "Chapter 6 - quantifying mitochondrial content in living cells," in *Biophysical Methods in Cell Biology* (E. K. Paluch, ed.), vol. 125 of *Methods in Cell Biology*, pp. 77–93, Academic Press, 2015.

[12] Y. Miyazono, S. Hirashima, N. Ishihara, J. Kusukawa, K.-i. Nakamura, and K. Ohta, "Uncoupled mitochondria quickly shorten along their long axis to form indented spheroids, instead of rings, in a fission-independent manner," *Scientific reports*, vol. 8, no. 1, pp. 1–14, 2018.

[13] Y. Song, P. Huang, X. Liu, Z. Zhao, Y. Wang, B. Cui, and L. Duan, "Light-inducible deformation of mitochondria in live cells," *Cell Chemical Biology*, vol. 29, no. 1, pp. 109–119.e3, 2022.

[14] J. G. McCarron, C. Wilson, M. E. Sandison, M. L. Olson, J. M. Girkin, C. Saunter, and S. Chalmers, "From structure to function: mitochondrial morphology, motion and shaping in vascular smooth muscle," *Journal of vascular research*, vol. 50, pp. 357–371, 2013.

[15] "Microscope." `https://purepng.com/photo/20614/clipart-microscope`, Jun 2018.

[16] S. Jakobs, "High resolution imaging of live mitochondria," *Biochimica et Biophysica Acta (BBA) - Molecular Cell Research*, vol. 1763, no. 5, pp. 561–575, 2006. Mitochondrial Dynamics in Cell Life and Death.

[17] C. J. Stefan, W. S. Trimble, S. Grinstein, G. Drin, K. Reinisch, P. De Camilli, S. Cohen, A. M. Valm, J. Lippincott-Schwartz, T. P. Levine, D. B. Iaea, F. R. Maxfield, C. E. Futter, E. R. Eden, D. Judith, A. R. van Vliet, P. Agostinis, S. A. Tooze, A. Sugiura, and H. M. McBride, "Membrane dynamics and organelle biogenesis—lipid pipelines and vesicular carriers," *BMC Biology*, vol. 15, p. 102, Oct 2017.

[18] A. Lehmussola, J. Selinummi, P. Ruusuvuori, A. Niemisto, and O. Yli-Harja, "Simulating fluorescent microscope images of cell populations," in *2005 IEEE Engineering in Medicine and Biology 27th Annual Conference*, pp. 3153–

3156, 2005.

[19] J. Lamsfuss and S. Bargmann, "Python codes to generate skeletal muscle models on each hierarchical level," *Software Impacts*, vol. 14, p. 100437, 2022.

[20] S.-Y. Park, C.-H. Park, D.-H. Choi, J. K. Hong, and D.-Y. Lee, "Bioprocess digital twins of mammalian cell culture for advanced biomanufacturing," *Current Opinion in Chemical Engineering*, vol. 33, p. 100702, 2021.

[21] B. M. Slepchenko, J. C. Schaff, I. Macara, and L. M. Loew, "Quantitative cell biology with the virtual cell," *Trends in Cell Biology*, vol. 13, no. 11, pp. 570–576, 2003.

[22] T. Heydari, M. Heidari, O. Mashinchian, M. Wojcik, K. Xu, M. J. Dalby, M. Mahmoudi, and M. R. Ejtehadi, "Development of a virtual cell model to predict cell response to substrate topography," *ACS Nano*, vol. 11, pp. 9084–9092, Sep 2017.

[23] D. Sage, T.-A. Pham, H. Babcock, T. Lukes, T. Pengo, J. Chao, R. Velmurugan, A. Herbert, A. Agrawal, S. Colabrese, A. Wheeler, A. Archetti, B. Rieger, R. Ober, G. M. Hagen, J.-B. Sibarita, J. Ries, R. Henriques, M. Unser, and S. Holden, "Super-resolution fight club: assessment of 2d and 3d single-molecule localization microscopy software," *Nature Methods*, vol. 16, pp. 387–395, May 2019.

[24] H. Gupta, M. T. McCann, L. Donati, and M. Unser, "Cryogan: A new reconstruction paradigm for single-particle cryo-em via deep adversarial learning," *IEEE Transactions on Computational Imaging*, vol. 7, pp. 759–774, 2021.

[25] H. Gupta, T. H. Phan, J. Yoo, and M. Unser, "Multi-cryogan: Reconstruction of continuous conformations in cryo-em using generative adversarial networks," in *Computer Vision – ECCV 2020 Workshops* (A. Bartoli and A. Fusiello, eds.), (Cham), pp. 429–444, Springer International Publishing, 2020.

[26] I. S. Opstad, G. Godtliebsen, B. S. Ahluwalia, T. Myrmel, K. Agarwal, and B. Birgisdottir, "Mitochondrial dynamics and quantification of mitochondria-derived vesicles in cardiomyoblasts using structured illumination microscopy," *Journal of Biophotonics*, vol. 15, no. 2, p. e202100305, 2022.

[27] H. R. Vutukuri, M. Hoore, C. Abaurrea-Velasco, L. van Buren, A. Dutto, T. Auth, D. A. Fedosov, G. Gompper, and J. Vermant, "Active particles induce

large shape deformations in giant lipid vesicles," *Nature*, vol. 586, pp. 52–56, Oct 2020.

[28] B. Mahler, P. Spinicelli, S. Buil, X. Quelin, J.-P. Hermier, and B. Dubertret, "Towards non-blinking colloidal quantum dots," *Nature Materials*, vol. 7, pp. 659–664, Aug 2008.

[29] C. H. Bohrer, X. Yang, S. Thakur, X. Weng, B. Tenner, R. McQuillen, B. Ross, M. Wooten, X. Chen, J. Zhang, E. Roberts, M. Lakadamyali, and J. Xiao, "A pairwise distance distribution correction (ddc) algorithm to eliminate blinking-caused artifacts in smlm," *Nature Methods*, vol. 18, pp. 669–677, Jun 2021.

[30] Y. Fu, J. Zhang, and J. R. Lakowicz, "Reduced blinking and long-lasting fluorescence of single fluorophores coupling to silver nanoparticles," *Langmuir*, vol. 24, pp. 3429–3433, Apr 2008.

[31] R. Platzer, B. K. Rossboth, M. C. Schneider, E. Sevcsik, F. Baumgart, H. Stockinger, G. J. Schütz, J. B. Huppa, and M. Brameshuber, "Unscrambling fluorophore blinking for comprehensive cluster detection via photoactivated localization microscopy," *Nature Communications*, vol. 11, p. 4993, Oct 2020.

[32] J. Li, F. Xue, and T. Blu, "Fast and accurate three-dimensional point spread function computation for fluorescence microscopy," *J. Opt. Soc. Am. A*, vol. 34, pp. 1029–1034, Jun 2017.

[33] L. Jahreiss, F. M. Menzies, and D. C. Rubinsztein, "The itinerary of autophagosomes: from peripheral formation to kiss-and-run fusion with lysosomes," *Traffic*, vol. 9, pp. 574–587, Jan. 2008.

[34] M. Zheng, M. Beck, J. Müller, T. Chen, X. Wang, F. Wang, Q. Wang, Y. Wang, F. Baluška, D. C. Logan, *et al.*, "Actin turnover is required for myosin-dependent mitochondrial movements in arabidopsis root hairs," *PLoS One*, vol. 4, no. 6, p. e5961, 2009.

[35] T. Mirzapoiazova, H. Li, A. Nathan, S. Srivstava, M. W. Nasser, F. Lennon, B. Armstrong, I. Mambetsariev, P. G. Chu, S. Achuthan, S. K. Batra, P. Kulkarni, and R. Salgia, "Monitoring and determining mitochondrial network parameters in live lung cancer cells," *Journal of Clinical Medicine*, vol. 8, no. 10, 2019.

[36] X. Liu, D. Weaver, O. Shirihai, and G. Hajnóczky, "Mitochondrial 'kiss-and-run': interplay between mitochondrial motility and fusion–fission

dynamics," *The EMBO Journal*, vol. 28, no. 20, pp. 3074–3089, 2009.

[37] M. Contributors, "MMTracking: OpenMMLab video perception toolbox and benchmark." `https://github.com/open-mmlab/mmtracking`, 2020.

[38] L. Yang, Y. Fan, and N. Xu, "Video instance segmentation," in *ICCV*, 2019.

# A

# Coding Documentation

The following will document the coding work I have done as part of this project.

## A.1 Debugging

When I first ran the code, it generated ground truth images as it should but the .tif images were all black. I fixed the .tif image generation code in print_mito() in generator_batch_parallel.py. I changed it so it converts all the elements in the image array to uint8 instead of uint16.

There was some text printed to the command line when running the original version of the code that said "FutureWarning: ‘rcond‘ parameter will change...To use the future default and silence this warning we advise to pass ‘rcond=None‘, to keep using the old, explicitly pass ‘rcond=-1‘." I chose to include rcond=-1 in the lstsq function to continue using the old default functionality.

The command_center() function (previously named parallel_run()) was originally written to happen in parallel, one instance per sample. In initial testing of this old code, it would usually crash and fail to produce mitochondria images but when I removed the parallelization, it ran smoothly and successfully produced all expected images.

## A.2 Cleaning

I have removed all the functions that are never called in the code.

I have removed all the code that is commented out.

The outputted data to files on disk that the old code produced is different from what I'll need to output in my project, so I removed the code for creating those files in favor of my own that I could customize. In addition, many directories were created with the old code on each run that never had any data deposited inside them, so I removed all code related to creating excess directories and made everything clean and simple.

I also streamlined a few obvious things, for example I noticed that the old code was reading wlow and whigh from the config file in generate_save_mito_in_batch() in generator_batch_parallel.py and then generating a random number between those two values and then passing that random value to generate_save_mito() just to do the entire process again inside that function in helper_generator_v5.py. I changed it to only read wlow and whigh in generate_save_mito() and then generate all the random values needed there.

I compared my solution for managing the coordinate unit relationship between pixels and nanometers with the old code solution and realized the old code simply lies about the max_xy value, and when I make it truthful, it actually ends up with the correct values for units. In the old code there is a variable max_xy which is set to 2000 and then in the function which actually prints the images

## A.3 Human Understandability

I changed the names of the functions and variables to be more human understandable. Often in the old code the function names and variable names were nonsensical and confusing. One example of something that's important to know that was obfuscated for some reason is that inside the batch csv files curve points are referred to as control points.

generate_mitochondria() -> render_mito() This function needs to be clearly differentiated from the function with a similar name "generate_save_mito". The purpose of this function is to take the list of points that make up mitochondria and print it into .tif and .png files. I moved this function render_mito() to the helper_generator_v5.py file because the generator_batch_parallel.py file because it seems more related to the generate_save_mito() function since these

two functions are the two main ones used in command_center()

parallel_run() -> command_center() Originally in the old code this function did nothing but call the two functions required to generate and print the mitochondria. The main purpose of command_center() now is looping through each desired frame of the output video. Once per frame, I call the same two functions from the old code which generate and render the mitochondria for that frame. This function is also where the initial values of all of the global parameters are defined and where all the action happens in terms of directing the dynamic behavior of the mitochondria.

generate_save_mito_in_batch() -> loop_through_samples() It's true that the function of the entire program is to generate and save the mitos in the specified batch. However, a more accurate description of this particular function is that it is looping through each sample.

get_mitochondria_2D_points() -> get_mito_center_line_pts() This function does return the two-dimensional curve points of the mitochondria. However, it also returns the three dimensional center line points of the mitochondria which is used for generating the points plotted onto the surface of the mitochondria later on in a different function and these points are the important ones which are used to print the final product images that are the whole purpose of this program.

get_mitochondria_3D_points() -> get_mito_surface_points() The 3D points referred to here are on the surface of the mitochondria. The center line points returned by the previous function are also 3D so this name doesn't help in differentiating the function of these two functions.

In generate_save_mito_in_batch(): total_sample -> samples This variable appears many times in this function, so I made it shorter without sacrificing specificity.

In main(): number_of_sample -> samples This was changed to be consistent with the preceding function for the purpose of human understandability.

get_mito_center_line_pts() dist -> mitoLength Dist is not as descriptive as mitoLength in communicating that the variable is the mitochondria's length.

curve_point -> num_of_curve_pts This variable dictates the number of total curve points used to define the position of this mitochondria.

p1, p2 -> curve_point_xes, curve_point_ys This name describes what this variable represents. It is an array containing the X values of the curve points of the

mitochondria.

zhigh1 -> zhighTemp This variable is a temporary copy of zhigh. So, this name is more descriptive and understandable to me.

x1, x2, x3 -> centerXPoints, centerYPoints, centerZPoints These are arrays of the X, Y, and Z values for a large number of points connecting the curve points in a gentle curve, I call these the center points because they lie in the center of the mitochondria forming a sort of spine.

get_mito_surface_points(): x, y, z -> centLineX, centLineY, centLineZ These are the same values described in the last set variables centerXPoints, centerYPoints, and centerZPoints.

generate_save_mito(): number -> sampleNum What kind of number? The number of the sample that is currently being worked on.

## A.4 Global Variables

I have added some global variables that will stay the same across multiple frame generations. This is important so a mitochondria keeps heading in the same direction as it was earlier. This is also important to remember the locations of the mitochondria, so I don't generate them in totally new random positions each frame. This will also prove helpful for reminding a mitochondria what it's doing for example executing a kiss and run.

curve_pointXs and curve_pointYs Lists that define the various mitochondria curve points, so the location and shape of the mitochondria are preserved from one frame to another. The first element in the curve_pointXs list is an array which contains all of the X values of the two-dimensional curve points which define the position of the first mitochondria. The length of this array is the same as the number of mitochondria.

rotate_speed and rotate_speed_range This is a list which contains a value for each mitochondria. The value is the angle out of 360 total degrees that each mitochondria will spin between each frame. rotate_speed_range is a list containing the low value and the high value for possible rotation speeds. When the program selects a random rotation speed for a mitochondria it will select a value between these two numbers.

wiggle_intensity and wiggle_intensity_range This is a list which contains a value for each mitochondria. The value is the number of coordinate points

that each point in the curve points of the mitochondria will move between frames. Each curve point moves in a random direction independent of the directions of other curve points in the mitochondria. wiggle_intensity_range is a list containing the low and high values for possible wiggle intensities. These are used when generating a random wiggle intensity for a mitochondria.

Page 6 of 23drift_speed and drift_speed_range This is a list which contains a value for each mitochondria. The value is the number of coordinate points that each point in the curve points of the mitochondria will move between frames. Each curve point in one mitochondria will move in the same direction. At the start of the program, a random drift direction is chosen for each mitochondria and that stays the same between frame generations. drift_speed_range is a list containing the high and low values for possible random drift speeds when being chosen for a new mitochondria.

seek_speed and seek_speed_range This is a list which contains a value for each mitochondria. The value is the number of coordinate points that each point in the curve points of the mitochondria will move between frames. Each curve point in one mitochondria will move in the same direction. The direction the mitochondria will move in is decided by finding one point in the mitochondria which is the shortest distance to one other point in a different mitochondria and calculating the direction the mitochondria would need to go to shorten that distance. seek_speed_range is a list containing the high and low values for possible random seek speeds when they need to be generated.

seek_target This is a list which contains a value for each mitochondria. The value is a number which represents a different mitochondria which this mitochondria is meant to be seeking.

merged This is a list which contains a list for each mitochondria. A list for a given mitochondria contains a list, each element of which represents a different mitochondria which this mitochondria is merged with. Each element representing a mitochondria that is merged to this given mitochondria contains a list containing three numbers. The first number is the id representing the other mitochondria. The second number is the id of the curve point on this mitochondria which is the merge point. The third number is the id of the curve point / merge point on the other mitochondria.

drift_directionX and drift_directionY This is a variable to keep track of the drift direction of each mitochondria. It does not take speed into account, so these values may be large or small, the only thing that matters is that the x and y gives the correct angle of the direction that the mitochondria should be drifting in. As usual these are lists with each item corresponding to each mitochondria in the same order as they are defined in the curve_pointXs list.

original_length This is a variable that defines the length of each mitochondria when they were first randomly generated. When mitochondrias wiggle their length changes. If the new length after the wiggle diverges from the value in original_length more than the set elasticity percentage difference from the original_length allowed, then the new wiggled position is abandoned and a new one is calculated.

## A.5 Argument Parser

I have added an argument parser so you can input variables into the program via command line upon running the program.

batch Each batch in the old code is given different config parameters stored in a corresponding file in the batch directory. During development I have kept this value as 2.

samples The samples var in the old code determined how many different images the code would produce each with different randomly positioned mitochondria. I have made the new code so that each sample is it's own video with differently positioned starting mitochondria. For each sample, instead of one image being generated, now a directory is generated which contains images which represent each frame of a video with the mitochondria's position moving slightly between images.

frames This is a new variable I have created which dictates the number of frames that should be produced. It is effectively determining the length of time of the video. The program simulates the movement movement changes that the mitochondria would undergo given their last known position until the desired number of frames is reached and then the program stops.

mitopop Number of mitochondria you want to generate

centermult and surfaceptsdivisor These are two variables that should be set to 1 if you want the mitochondria to appear as they should. The default for these is set to some number which dramatically reduces the total number of points that make up the mitochondria. This dramatically reduces the amount of time it takes to run the PSF function when rendering the .tif images for these mitochondria.

canvas This variable is used to adjust the size of the environment which contains the mitochondria. When the program randomly generates the mitochondria for the first frame they can appear anywhere in this area, but when the mito-

chondria move around in later frames they may drift outside this field of view. The units which this variable is measuring is nanometers.

onlygt This can be set to zero or one. If it's set to one, then only the ground truth images will be generated, and the time intensive PSF function will be skipped and no .tif images will be generated at all. This is nice for development when I just need to see the position of the mitochondria and not the exact perfect mimicked microscope blurring.

psftogether If this is set to one, then the program will use the method from the old code where all points in the images including all mitochondria are sent off to be processed by the PSF function. This uses a large amount of RAM. If the psftogether variable is set to zero then the program will separate the list of points into smaller chunks, one for the points that make up each mitochondria. Each mitochondria will be sent separately to the PSF function. If run sequentially, this consumes much less RAM. After the separate images of each mitochondria in the frame have been rendered, they are all simply added together into one image.

parallel If gsftogether is set to zero, these PSF function calls can happen in parallel because the images created for each mitochondria don't effect the other images, they can be added together normally when all are complete. Setting this parallel variable to one will make that happen. This dramatically increases the speed of the program. 90-95 percent of the run time is consumed by the PSF function. Running it parallel speeds up the run time of the PSF function by a factor equal to the number of mitochondria in the frame.

writedisk If this is set to zero, the PSF function will have it's largest variables written to disk in a way that makes the amount of RAM being used at any given moment lower. This dramatically increases run time. This is nice for testing on my machine when I need more mitochondria points and a larger canvas.

## A.6 command_center()

Because command center can run in parallel when there are multiple samples, it's important that all the samples don't fail when there's an exception in one of the samples. Therefore I've put the entire function inside a try except statement. When there's an exception in one of the samples, the program doesn't terminate. The way I made it so control c still works to terminate the program is I made the first except statement handle specifically the keyboard interrupt exception and exited the program there.

At the top of the command_center() function is where many of the variables are defined that will keep track of the behavior of each mitochondria across frames. The seek_target variable starts out full of empty values meaning none of the mitochondria start out seeking eachother. This is the same for the merged list because none of the mitochondria start out merged with eachother. The rotate_speed, wiggle_intensity, drift_speed, and widths variables are all populated with random values between the lower and upper limits defined for each of those attributes. The drift_direction X and Y is chosen for each mitochondria by choosing a random value between -1000 and +1000 for each X and Y. These two numbers are used as direction by taking the angle of the hypotenus of the triangle drawn by these two lines in the coordinate space. The length of this hypotenus isn't used for anything, it's only useful for it's angle.

The command_center() function contains a loop that runs once per frame. For each frame, the execute_behaviors() function is called to move the mitochondria according to the behaviors they are marked to be executing. Then functions are called to generate the positions of the mitochondria from their curve points and width. After the locations are known that is used to render the complete images both ground truth and the blurred tiff image.

## A.7 execute_behaviors()

I loop over the mitochondria and the global variables are checked to see which behaviors each mitochondria should be executing. The mitochondria are actually looped over twice, once for the seek behaviors and again for all the rest of the behaviors. That way if a seeker mitochondria is close enough to it's seeked mitochondria to make that final hop, it doesn't escape at the last moment by executing it's own seek or drift behavior. The drift function is ignored if a mitochondria is seeking, so the drift doesn't fight against the seeking. These functions for wiggling, seeking etc can modify the global variables, notably the curve_pointX and Y variables. That has an effect on the position of a mitochondria in a frame and going forward in future frames as it's location changes that makes a continuous movement and effects subsequent decisions. For example when a mitochondria reaches the mitochondria that it was seeking, it might execute a kiss and run whereby it changes the drift speed global variable to some value, sets the seek_target global variable to -1 which means it's no longer seeking, and sets some direction to drift.

## A.8 loop_through_samples()

generate_save_mito_in_batch() was renamed to loop_through_samples(). In the old code this function called command_center() (formerly named parallel_run()) in parallel once for each sample. It ran each sample in parallel. I made it run each sample sequentially in order to fix a bug that caused the entire program to crash. Now it runs each sample one after another in a loop.

## A.9 wiggle()

To achieve the shivering behavior of mitochondria seen in microscopy videos, every frame we wiggle each mitochondria. Each mitochondria has a wiggle intensity associated with it which is chosen at the start of the program at random between the user provided minimum and maximum wiggle intensity values. To achieve this wiggling, I choose two random numbers between zero and the wiggle intensity for every curve point and change the point's x and y coordinates by these values to achieve a shift by a random amount in a random direction. The wiggle intensity is randomly chosen for each mitochondria at the start of the program before the first frame is rendered.

## A.10 seek()

The seek behavior involves a seeker mitochondria and a seeked mitochondria. The seeker will move itself in the direction of the seeked and stop when it reaches the seeked. When the seeker meets the seeked, the two mitochondria might merge or a kiss and run might be performed. A kiss and run is when the seeker retreats away from the seeked and the seek behavior is ended. If they merge together, and the point on one of the merging mitochondria at which they merge is not one of the two ends at the tip of the mitochondria, there can be a chance that this merge point will become a split point where the small bit hanging off the end of the new larger mitochondria is chopped off and begins retreating away in some direction.

The seek function takes as an argument the id of the seeker mitochondria. This is all that's needed because the id of the seeked mitochondria can be looked up in the global seek_target variable. When a new frame is being generated and a mitochondria is performing a seek manouver, the drift function is called if the seeker is not already overlaping on top of the seeked mitochondria. The drift function requires as arguments either 1) the direction we should be moving given by a X and Y value, and the distance along the hypotenus that we should

move, or 2) the exact X and Y values that we should move. In this case we can get the distance and imprecise direction easily so that is calculated in the following way. First the direction is needed and we loop through each curve point of the seeker mitochondria and find the distance to every other point in the seeked mitochondria. We make a note of the shortest distance found and the direction from this seeker point to the seeked. The distance sent to the drift function is found from the global seek_speed variable.

Merging two mitochondria together is what most of the code in this function is dedicated to. It's important to properly update the global variable merged which contains information about which mitochondria are merged with which other mitochondria. I also change the drift speed, rotation, and drift direction of the seeked to be the same as the seeker so they stay connected. Initially I did quite some work building the infrastructure to find the average drift speed, drift direction, and rotation of all of the merging mitochondria. This idea felt more natural and I abandoned it eventually because I was having trouble making it look good when the seeked mitochondria was drifting in a direction which was at a 90 degree angle to the seek direction of the seeker. In this case as the seeker draws nearer to the seeked mitochondria, it must change direction quickly as the seeked mitochondria continues drifting. This change in direction is sometimes abrupt and it makes the collision look very wrong when only the final directions and speeds are taken into consideration. During testing sometimes it was looking as if the the final drift speed and directions were nonsensical and random. This is especially the case when you consider that the points pursuing eachother are the curve points, so it can look as if they overlap and pass through eachother slightly before the merge takes place, and that small delay can mean a lot for the seeker's seek direction when the order of the various mitochondria's drifting occures in a way that has consequences on the seek direction that has not been fully gamed out.

## A.11 split()

This function provides functionality for splitting a mitochondria into two parts. The variables that should be given as arguments to this function are the index of the curve point that you want the split to take place at and the mitochondria's id which should be the one to be split.

If the split point is on the end of the mitochondria, then a split does not occur but any mitochondria which were merged to the mitochondia on that end point will be seperated. A split should not effect mergers which were on points other than the split point.

When a mitochondria is split into two parts and there were mitochondria merged to it on that split point, the larger half keeps the merger. So if a split happens on curve point 1 and there were mitochondria merged at curve point 1, then those mitochondria will now be merged to the mitochondria formed from curve points 1, 2, and 3. And the smallest mitochondria resulting from the split made from old curve points 0 and 1 will not be given any of these merged mitochondria.

The curve points belonging to the mitochondria that is to be split in two are organized into two groups where the split point belongs to both groups. One group contains the points below the split point and the other group countains those above the split point. In a mitochondria with four curve points, one group contains three points and the other group contains two points. These curve points will be given to the two new mitochondria formed by the split. The length of those two mitos is checked and either of them will be less than 280 coordinate space units then the outter two endpoints are moved away from eachother such that the resulting mitochondria will be 280 coordinate space units in length. Before this endpoint spread function was implemented there were many errors when a very small mitochondria was produced relating to the wiggle. The wiggle behavior changes the length of the mitochondria by an amount related to the static wiggle intensity variable, and the wiggle will fail if it cannot find a wiggle motion that lets it stay within a certain percentage of it's original length defined in the elasticity variable. This means that very small mitochondria will wiggle too much and fail due to the very tight elasticity contstraints inherent in short mitochondria. Another approach to solving this issue could be adjusting the wiggle intensity down for smaller mitochondria, however I think limiting the mitochondria's minimum length is a good solution because very small mitochondria stop looking much like mitochondria so it addresses multiple problems.

Now that we have two smaller mitochondria, we need to add curve points to them such that they have 4 again like before. The mitochondria with just two curve points gets two added between it's end points and the other mitochondria gets one added between it's last curve point and the second to the last one. When these additions are made, several variables are updated accordingly. Most notably the global merged variable that keeps track of where the merges are. So any merges on curve points that got bumped up or down in this reshuffling get adjusted accordingly so the curve point ids reference the same points as before the additions.

Finally, the global variables keeping track of things like wiggle intensity, drift speed, length, and seek target are all updated and incremented to include an element that represents this newly created mitochondria. The smaller mitochondria made up of two of the old large mitochondria's curve points is the

one given the new id equal to the old highest id plus one.

## A.12 connect_merged_points()

When two mitochondria are marked as merged with eachother, it means there is a merge point on each mitochondria where they should be touching. The merge point is a curve point. After each wiggle, the merge points drift apart and so the two mitochondria become disconnected. Not to mention if there are several mitochondria merged to each other in a complicated network, we need to keep their merged point pairs overlapping with each other after each wiggle behavior. The connect_merged_points() function uses drifts to move an entire mitochondria enough to make one of it's merged points overlap with the corisponding merge point on another mitochondria. The reason I don't simply move only one or both merge points is because then it could stretch the length of a mitochondria, and then on the next wiggle, there would need to be some coordinated motion in the wiggle behavior toward that direction in which they got stretched. This is concievable but I hesitate to do this because this would require lots of testing to iron out to be sure there's no unintended side effects.

In a large group of interconnected mitochondria, they are moved to line up with their merge points in the following way. One after another, each mitochondria is dealt with in a loop. For each one, we find all mitochondria merged with it and move those merged mitochondria toward it such that the merge points overlap. This is done the same way with each of the remaining mitochondria. After this is finished, often one or two pairs of merge points will be left overlapping successfully, however the merge point pairs that we connected first might have been separated when the last pairs were connected because we moved the mitochondria from their position thus separating the merge points with other mitochondria they should be merged with. This process is repeated several times, each time checking if it is finished connecting all the merge point pairs. Sometimes in a complex network, a never ending loop can develop where the mitochondria never settle on a final position that succesfully connects all merge point pairs. Every ten iterations the order that the loop passes over the mitochondria in is changed to scramble any never ending loops. This order is changed ten times.

## A.13 network_and_dissolve()

This function checks if we are on a particular frame number and if yes then it initiates a seek behavior or it triggers a split. It is meant to display some basic functions of the code without taking into account the proximity of mitochondria to each other or their status as already merged or not. In this example I initiate seek behaviors on the first 6 mitochondria one after another with the seek target some other arbitrary mitochondria, and after a few hundred frames I tell them to split on arbitrary curve points one after another. This results in the first half of the video what looks like mitochondria networking together to quickly form a complex bundle and then later they all start dissolving into little bits. The network created will look different depending on the value held in the probability_of_merge variable which will determine how often a successful seek will result in a merger. There are other variations on this function. One of them networks the mitochondria but doesn't dissolve them.

## A.14 Keeping Time

I start a timer at the beginning of each frame and stop it when it's finished and print that time to the command line. I also run a seperate timer at the beginning and end of the PSF function because I quickly realized that function was eating up the vast majority of the run time. This helps get some sense of the run time of the entire program after just a few frames. I also print out the time of day when each frame finishes. This data is all printed as well to a text file. This is helpful if the program has been running over night to see when different events occured more easily.

## A.15 write_to_attributes

I only print to the command line a few details to check quickly while running the program, but I also print more details to a text file. This is useful to get more detailed information but also it's nice to have a copy of the print statements in case the program was run in a headless environment.

## A.16 get_mito_center_line_pts()

This function has two purposes. It generates random curve points for a new mitochondria, and it produces a string of points that connect those curve points

along a smooth curved line. While generating a frame of the video which is not the first frame, we need to skip the part of this function that generates an entirely new position for the mitochondria. I have modified it to receive the current curve points, if these are sent then it means this is not the first frame, and the only thing needed is to produce and return the center line points.

## A.17 generate_save_mito()

The purpose of this function is to get a list of points that make up the mitochondria by calling the get_mito_center_line_pts() function and the get_mito_surface_points() function. In the old code, there were specialized sections of this function for the first mitochondria and the second one. I have modified this to work with an arbitrary number of mitochondria by storing their attributes in lists and looping over each mitochondria. For each mitochondria, the function still calls those two main functions for getting the center line and surface points.

In the old code the list of points created by generate_save_mito() was written to disk and then later this list was retrieved when rendering the tif image but I changed it so it simply returns the list back to the command_center() so we don't have to waste time screwing around writing and reading from the disk.

I removed the code for plotting the mitochondria in a 3D graphic. Although it was cool to see and test in the beginning of the project, I don't see a use for it now that I know the points are being generated correctly in three dimensional space and I know how to manipulate them.

## A.18 render()

This function recieves a list of all points contained in all the mitochondria and uses that to generate a black and white ground truth image and a blurred .tiff image. In the old code the canvas size was 2000 coordinate space units and in the render_mito() function, the number 1344 was used as the canvas size. Therefore when I made the canvas size variable I changed it in render_mito such that it will have the same proportion to the canvas size as 1344 has to 2000. I used the same proportional modification to the size_x variable which is the width of the canvas in pixels. In the old code this was 128 and I have changed it to be equal to the old canvas divided by 2000 times 128. This has worked well generating images from the mitochondria points.

The render() function can print both the blurry .tif images and .png ground truth images and it can have either of those disabled, but if both are disabled then this function doesn't run at all. For printing the .tif images there are several different options. If psftogether is set to 1 then the code which is run looks similar to the old code, where one huge list of points is submitted to the psf function. If psftogether is set to zero then a list of all the points in one entity is sumbitted at a time, and the resulting images are added together afterward to create the final image. In this case, the possibility of parallelization is available where the psf function for each entity could be run in parallel and then added together afterward in the same way when they are all finished. This option is not available when parallel samples is turned on.

When generating the .png ground truth image, if any point lies outside the canvas an error is thrown. Therefore before sending the points to the ground truth image generating function save_physics_gt(), points are deleted from the list to be sent which are found to lie outside the canvas. Sometimes this still throws an error, but I was able to fix this by creating a loop where I send the trimmed down set of points to the function and if it throws an error, then I trim down the set of points further by slightly decreasing the canvas size variable until it works successfully. I correctly guessed that the problem was caused by some small descrepency between the canvas size variable and the way that this save_physics_gt() function interperates the canvas size.

## A.19 Size of Canvas

I brought the max-xy variable from the batch file for batch 2 b2.csv into the code as a command line argument for testing. It used to be strictly 2.000 but now it can be anywhere between 3.000 and 20.000 but beware the larger it is the more RAM it will use to calculate the PSF in rendering the .tif images. In generator-batch-parallel.py, the only time that save-physics-gt() is called, I changed the max-xy argument from 1344 to be proportionally reduced from the max-xy argument the same relative size as 1344 is to 2000, and the image size variable from 128 to be proportional to the max-xy.

## A.20 GUI

I created space in the gui to manipulate variables related to vesicles such as the number and radius range of vesicles.

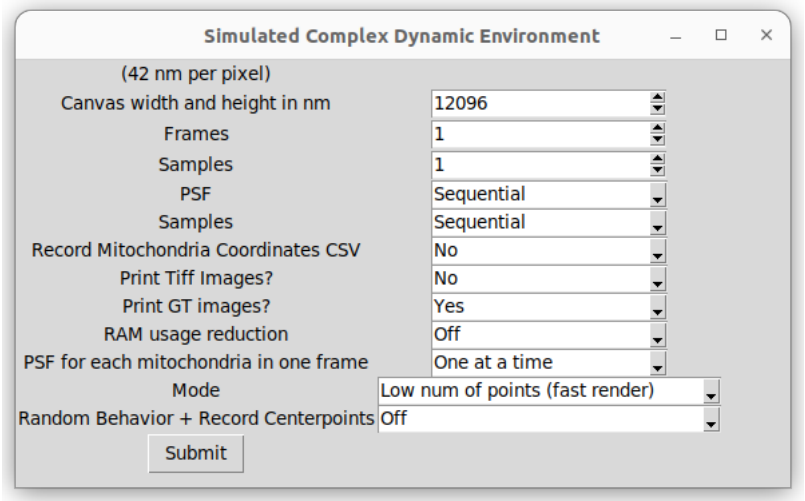


**Figure A.1:** General program settings

## A.21 Vesicles

I copied and pasted the code from the vesicle code into my code

## A.22 rohit csv

I built code to translate the mitochondria id from my id system where the two parts of newly split mitochondria are given the old id and a new one to rohit's sytsem where both parts are given two new ids while the old id is abandoned and used those ids to print the centerpoints of the mitochondria into a csv.

## A.23 Dead Ends

When I increased the canvas size, my computer would use more and more memory until the program eventually crashed when the canvas was around double the size of the original from the old code. The original canvas size was very small so this needed to be fixed. I investigated which part of the code was responsible for the high memory usage using print statements and sleep timers, so whenever I printed some text I would put the program to sleep for five seconds so I could see the memory graph level out to a certain value. The PSF function turned out to be the culprit. This function takes the array of points which make up the mitochondrias and outputs a blurred image that mimics the

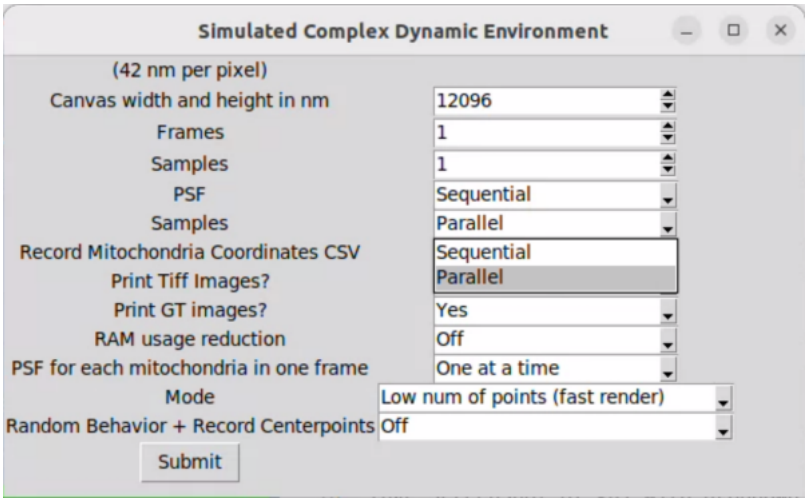


**Figure A.2:** Parallel Samples Dropdown

optical effects of a microscope. I splintered the calculations within this function into their respective atomic pieces to further pinpoint the exact parts of the calculations that required the most RAM.

My first idea that I could imagine easily reducing the memory usage was to render this blurry image first for one mitochondria, and then add that image to an image rendered of another mitochondria and so on for all of them, and in that way I could do the PSF function for one mitochondria at a time instead of all at once. This helped but for larger canvas sizes that would be needed for more than around five mitochondria it was not enough memory usage reduction.

The next idea I had to reduce memory usage was cutting the canvas into smaller squares and rendering those separately and then concatenating them back together. I spent quite some time trying to figure out how to adjust the coordinates of the points so that they centered around the origin for each fragmented piece of canvas. This did not work in the end because the PSF function outputted different brightness pixels based on the total number of points submitted, so in some fragments there would only be a sliver of mitochondria and you could subsequently see the boundary between fragments clearly and that makes this approach useless.

I discovered a library called h5py. This proved perfect for my use case. This library allows you to easily save arrays to disk and then perform numpy calculations on them as if they are saved in memory. I converted each variable in my splinters of the calculations of the PSF function into these h5py objects and monitored the memory usage change and made detailed notes about this,

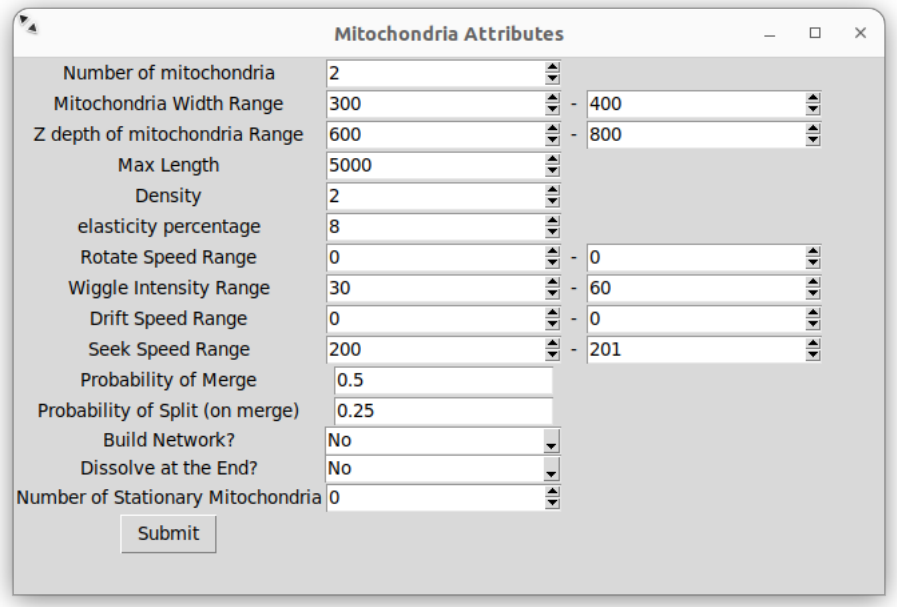


**Figure A.3:** Mitochondria Attributes

making sure to only write the largest variables to disk. The amount of time used to run the program grew substantially. I ran a test to render one frame containing 100 mitochondria on a canvas of 430 pixels by 430 pixels and it took my computer over seven hours to complete the image.

## A.24 Bugs

If the width of a mitochondria is below 222, there is a bug that causes an exception in the get_points() function.

# B

# Bugs

The movement of a kiss and run should be improved. Currently the movement of the kiss and run is quite unnatural, the seeker drifts off in a random direction but it would look more realistic if it instead made an intentional sprint of a certain distance and stopped. This would more closely mimic what can be seen in real microscopy videos, and it would reduce the chances of a mitochondria drifting all the way off screen and be lost to the simulation.

Currently there is no motion in the Z dimension. The way the Z coordinates are chosen when generating the points of the center line skeleton of a mitochondria is by drawing a straight line between the high z and the z low user inputted value. It would be better if the mitochondria wiggled randomly in the z dimension and if the initial z coordinates are chosen randomly instead of drawing a straight line. This could be achieved by saving the Z value of curve points and not just the X and Y values. Then these could be modified by behaviors and the mitochondria would more realistically drift in and out of focus which is probably an important effect to capture for the machine learning model to get a more full picture of how mitochondria can behave over time.

There is a lack of creativity in triggers for behaviors, there could be triggers for seek behaviors or fission behaviors which are dependant on factors such as shape of the mitochondria or proximity of the mitochondria to other mitochondria or size of the network that the mitochondria is part of or length of the mitochondria.

Currently seek and fission behaviors are largely triggered arbitrarily regardless of the state of these other factors. For example, a mitochondria should not be allowed to start seeking another mitochondria if it is currently in a connected network with that mitochondria.

The fission behavior could be improved by making the two mitochondria into one smoothly curving whole one instead of having an abrupt change of direction at the point where the two mitochondria meet. There is a bug where sometimes in large networks the function to keep the fusion points together doesn't succeed. If the two fusing mitochondria became one smoothly curving mitochondria, this would also remove the need for the function which connects fused curve points and it would fix the bug which prevents the fusion points from being reconnected. In a network where a mitochondria is attached to another on a point which is not an end point however, the current system could be kept where two separate curves are drawn with one attaching and abruptly changing direction on that fusion point. Another way to fix the issue of the function which connects fusion points together not working would be to remove the problem entirely by changing the way the wiggle is done. The wiggle motion could happen to all mitochondria in a network at the same time, and in this way a different wiggle could be chosen until one configuration is found that doesn't violate the elasticity rule of any of the fused mitochondria.

More thought should be given to the question of how often a network of mitochondria will perform a seek behavior and if this would involve them all moving together as a network or if a single exterior facing mitochondria would change shape and stretch or reach around rather than actually translating.

A good way to solve the problem discussed in Spread Endpoints (Section 4.7) would be to add a more simple and robust triggering function for behaviors where many checks can be performed on the environment before following through with a behavior trigger. In this example with spreading endpoints, a fission behavior could be double checked to see if it would cause a new mitochondria to be spawned who is shorter than the minimum length parameter and then the fission behavior could be aborted. This function could return a sign that this behavior is triggered in the end or not. Or even better, all handling of parameters relevant to tracking triggers could be performed in this function. This would provide a way for the code to have less repeating sections and it would add a centralized location to go for known required changes.

The current system used to connect fusion points together after a wiggle occurs in failures in complex networks. Other solutions should be investigated such as trying different wiggle configurations for every mitochondria in a network where the fusion points are given the same wiggle movement. This can be tried many times until the elasticity requirements of each involved mitochondria are

not violated. This is discussed in Methodology (chapter 4)

Currently when a fission-on-fusion behavior is conducted, the fusion which induced the fission-on-fusion behavior can be terminated. This is not ideal because in effect, a fusion can be the cause of it's own unfusing which is not realistic. This can happen if the fusion point is on an end point of a mitochondria and if the probability of fission-on-fusion is high enough for a fission-on-fusion event to be triggered. The default behavior then is for all fusees on this fusion point to be terminated including the fusion which triggered this behavior. However, the possibility of a kiss and run behavior triggering a fission should be investigated for future iterations of CODS too because this may not be so unrealistic.

There is a bug where sometimes a random movement is induced in groups of mitochondria off screen sometimes at high speed. This could be caused by one mitochondria trying to begin seeking a mitochondria which it is already in a network with. This could be fixed by detecting whether a mitochondria is currently in the same network as another mitochondria and preventing any seek behavior between these two mitochondria.

Currently, during a seek if there is a small distance left to go before the seeker reaches the seeked mitochondria, the distance traveled by the seeker is reduced to be the remaining distance so that the next frame shows them touching. This is not realistic. If there's a kiss and run then the seeker should be already moving away from the seeked by the time the next frame arrives if the times of the frames do not perfectly line up with the time of the kiss and run. This can be adjusted for future iterations of CODS

# C

# Video Links

## C.1 Fission and Fusion

The mitochondria near the center of this video suffers a fission event and then fuses with a nearby mitochondria. https://drive.google.com/file/d/1uCkZ2F8N6ab1pceI50kTlVHi3DOLpUw5/view?usp=share_link

## C.2 Simulated Kiss and Run

This video features the seeker drifting offscreen after performing the kiss and run manouvre. https://drive.google.com/file/d/136EW_OWLfvK1PCS0I46fJ3s2YhubpGPq/view?usp=sharing

## C.3 Wiggle

This mitochondria which is short in length is a good example of the wiggle behavior that CODS simulates. https://drive.google.com/file/d/16Iech09w4rHjAWQgogbNHtm5DglUrjxQ/view?usp=share_link

## C.4 Simulated Wiggle

This video is a CODS simulation where a mitochondria can be seen performing a wiggle behavior. https://drive.google.com/file/d/13AFYjvyDQDwgOgQb7LCtCbKqB2pY9aXr/view?usp=sharing

## C.5 Drift

This video contains mitochondria being pulled by what seems to be a current in the medium they are immersed in. https://drive.google.com/file/d/1PRp_uab2EqofMgTI3HT7Hj_cpNXOidCP/view?usp=sharing

## C.6 Simulated Drift

Mitochondria in this video generated by CODS are set to drift at different rates. https://drive.google.com/file/d/1TK_J2oqesXCI1Vqu-lJFiyXMYUYLDdgK/view?usp=share_link

## C.7 Seek and Kiss and Run and Fusion

This video shows a long mitochondria reaching up from the bottom of the screen to perform a kiss and run behavior and then shortly thereafter fuse with the seeked mitochondria. https://drive.google.com/file/d/1hC6CwWf8ZOWUWtDvWcFZkyNzk-OTnpUz/view?usp=share_link

## C.8 Simulated Kiss and Run Not Endpoint

This simulation showcases the ability for simulated mitochondria to perform seek manuevers toward joining two points which are not mitochondria endpoints. https://drive.google.com/file/d/1YxFrmdEmF4qxGwZu7lAveHeNI5exe-C7/view?usp=sharing

## C.9 Seek and Fusion

It's more difficult to clearly identify the objects of interest in this video. The action happens near the bottom where there's some empty space with a mobile mitochondria with quite short length. It moves in curved movements toward the right edge of the screen until it encounters a long mitochondria. The fusion that happens next is quite clean and clear. After the fusion the two mitochondria begin acting as one agent, and the connection appears seamless. https://drive.google.com/file/d/1Y_DzTuWShZ7ypR64n79i1Ca6HDE8eVhh/view?usp=share_link

## C.10 Simulated Seek and Fusion

Seek behaviors in CODS simulations sometimes look odd because they can translate mitochondria along an axis perpendicular to their length, but this simulation video exhibits a mitochondria performing a more natural looking seek motion. https://drive.google.com/file/d/1goEv7NgNfw6K3Issd_Zdf-iZzaRa3qoF/view?usp=share_link

## C.11 Simulated Seek, Fusion, and Curl up Into Doughnut

This Fusion has the seeker curling up into a tight ball afterward which looks similar to the observations described in this paper [12]. https://drive.google.com/file/d/15lPN6-BFN1aon3ikWtbY-Dk0UO06NvjL/view?usp=sharing

## C.12 Kiss and Run and Fusion

The mitochondria in this video bump into eachother before fusing into one long mitochondria. https://drive.google.com/file/d/1eXHK4RuqmA43eBNI1BAJxq0c9xR17vjp/view?usp=share_link

## C.13 Simulated Kiss and Run Multiple

This simulated mitochondria bumps into the seeked mitochondria to illustrate that mimicking this behavior is possible with a few changes to the CODS parameters. https://drive.google.com/file/d/1eLvJQeQRu0oWalhPm2G4UD40d

IAidBe1/view?usp=sharing

## C.14 Fission Multiple

The large mitochondria in the center of this video fissions off a large chunk, and then that chunk undergoes another fission event producing a very small ball that propells itself away. https://drive.google.com/file/d/19EjPrYn6LvCRY5HpEcKjNUXKfXS8N_QL/view?usp=share_link

## C.15 Simulated Multiple Fission

CODS can produce more than just kiss and run behaviors multiple times in a row. https://drive.google.com/file/d/1eNfrkoi3PyqzHcGB8p_Sw7XDNMdZaPNJ/view?usp=sharing

## C.16 Fission on Fusion

This fission on fusion event is hard to see but if you look carefully in the center there is a possible fission triggered by fusion event. https://drive.google.com/file/d/1byM3dUGD_NKCP64feB1pcVdx6BVNu9Rf/view?usp=sharing

## C.17 Fission on Fusion 2

This fission on fusion event is also tricky to make out but it happens near the bottom of the srceen. https://drive.google.com/file/d/1oq8Pxu4IHU31WCRIHviJd5wG3du7LicQ/view?usp=sharing

## C.18 Simulated Fission on Fusion

This video features a simulation of a fission on fusion behavior taking place. https://drive.google.com/file/d/1HVw9tWwuBvnBWkA5bRWw6TI6eAlWhf0g/view?usp=sharing

## C.19 Drift Inference

This is the output of the MMTracking [37] tool when prompted to do inference on the Drift video C.5 video after being trained on CODS data with three epochs. https://drive.google.com/file/d/1W8NPaUSewhFPXHmeA-zLygol0_LlrTCD/view?usp=share_link

## C.20 Fission and Fusion Inference

This is the output of the MMTracking [37] tool when prompted to do inference on the Fission and Fusion video C.1 video after being trained on CODS data with three epochs. https://drive.google.com/file/d/1MkfTvvMjs2z8wnpy3QJaPZTydyKMdOWn/view?usp=share_link

## C.21 Fission Multiple Inference

This is the output of the MMTracking [37] tool when prompted to do inference on the Fission Multiple video C.14 video after being trained on CODS data with three epochs. https://drive.google.com/file/d/1cwHm4eOBcrGBD4Pmgnr0doii3ErjIsgc/view?usp=sharing

## C.22 Kiss and Run and Fusion Inference

This is the output of the MMTracking [37] tool when prompted to do inference on the Kiss and Run and Fusion video C.12 video after being trained on CODS data with three epochs. https://drive.google.com/file/d/1mPdo1WHseAeuJ2k3nJcnsLQKPrgsdGZJ/view?usp=sharing

## C.23 Seek and Fusion Inference

This is the output of the MMTracking [37] tool when prompted to do inference on the Seek and Fusion video C.9 video after being trained on CODS data with three epochs. https://drive.google.com/file/d/1C4kEOp4onmtyqH-4ilr3w1jpgZo07XzO/view?usp=share_link

## C.24 Seek and Kiss and Run and Fusion Inference

This is the output of the MMTracking [37] tool when prompted to do inference on the Seek and Kiss and Run video C.7 video after being trained on CODS data with three epochs. https://drive.google.com/file/d/1jkDZ_Lbf-fAm6Ad-EajBYGpQ3AIAbMEM/view?usp=share_link

## C.25 Simulated Fission on Fusion Inference

This is the output of the MMTracking [37] tool when prompted to do inference on the Simulated Fission and Fusion video C.18 video after being trained on CODS data with three epochs. https://drive.google.com/file/d/1XWxQCvLAE2PH_0PBvFRteloG4imBH3Sw/view?usp=sharing

## C.26 Simulated Seek, Fusion, and Curl up Into Doughnut Inference

This is the output of the MMTracking [37] tool when prompted to do inference on the Simulated Seek, Fusion, and Curl up Into Doughnut video C.11 video after being trained on CODS data with three epochs. https://drive.google.com/file/d/1PqBgIIL_Yi-FCal3sZFuQ54s7dbPdBlf/view?usp=share_link

## C.27 Simulated Kiss and Run Multiple Inference

This is the output of the MMTracking [37] tool when prompted to do inference on the Simulated Kiss and Run Multiple video C.13 video after being trained on CODS data with three epochs. https://drive.google.com/file/d/1BWP9Gwo-tt3Hsb9mrstMCTe-X0FOiZ_R/view?usp=share_link

## C.28 Simulated Kiss and Run Not Endpoint Inference

This is the output of the MMTracking [37] tool when prompted to do inference on the Simulated Kiss and Run Not Endpoint video C.8 video after being trained on CODS data with three epochs. https://drive.google.com/file/d/1atZkECAHwGrjVpI-PPOkORBeke_UBNd2/view?usp=share_link

## C.29 Wiggle Inference

This is the output of the MMTracking [37] tool when prompted to do inference on the Wiggle video C.3 video after being trained on CODS data with three epochs. `https://drive.google.com/file/d/1pP8KNSo_DdE1zL5bQnmkVC7H-7RxGdPA/view?usp=share_link`