Worcester Polytechnic Institute

Biophysical Insights into Moss Morphology and Organelle Motility

A Major Qualifying Project Report submitted to the

Faculty of the Worcester Polytechnic Institute in Partial Fulfillment of the

Requirements for the Degree of Bachelor of Science

By

Kyle Lemoi October 5, 2012

Submitted To:

Professor Erkan Tüzel, Advisor,

Department of Physics

Abstract

In the past decade, *Physcomitrella Patens* has become a popular model organism for its simple development and powerful molecular tools, which include a complete genome sequence, rapid transient RNAi, and homologous recombination. This project focuses on developing approaches which are able to provide quantitative metrics in two areas: moss morphology and chloroplast motility. We begin by introducting the concept of the fractal dimension and presenting experimental data obtained via the box-counting algorithm. A simple model of plant growth based on L-systems is developed which is able to recapitulate the experimental results. A tracking algorithm was devised and used to track chloroplasts. Based on the trajectories of the chloroplasts, we find, through an analysis of moments of displacement, that microtubles are the primary facilitators of chloroplast motility in moss.

Contents

1 Acknowledgements **5**

2 Introduction **6**

3 Analysis and modeling of plant morphologies **8**

3.1 Introduction 8

3.2 Existing methodologies 8

3.3 Using fractals as a shape descriptor 10

3.4 Application of fractal dimension measurement to moss morphology 13

3.5 A simple model of plant growth 19

3.6 Summary 25

4 Quantitative analysis of organelle motility **27**

4.1 Introduction 27

4.2 Development of an algorithm for tracking chloroplasts 27

4.3 Moments of displacement 29

4.4 Experimental Results 32

4.5 Summary 36

5 Conclusions **37**

6 Appendix **38**

A Box counting algorithm code **38**

B Growth model code **47**

C Tracking algorithm code **52**

D Auxiliary codes **58**

D.1 getAllFiles.m 58

D.2 cell2txt 60

D.3 track_txt 61

D.4 fractalbcount_txt 62

List of Figures

1 Light response of chloroplasts . . . 7
2 Qualitative nature of common morphology metrics . . . 9
3 Convex hull example . . . 10
4 Coastline of the United Kingdom . . . 11
5 Fractal dimension of coastlines . . . 12
6 Fractal dimensions of common fractals . . . 13
7 Alpha-trimmed mean filter example . . . 14
8 Box counting algorithm: box counting . . . 15
9 Box counting algorithm: fitting . . . 16
10 Morphology data for Physcomitrella Patens . . . 18
11 String rewriting example . . . 19
12 Cell length and angle distributions for Physcomitrella Patens . . . 21
13 Example of simulated growth of the first model . . . 23
14 Comparison of first growth model to experimental results . . . 23
15 Distributions of the number of balls, activie balls, and branchesl . . . 25
16 Final growth model . . . 25
17 Averaged and recursively averaged mean squared displacement of drug and light treatment conditions . . . 34
18 Higher moments of displacement for light conditions . . . 36

List of Tables

1 Comparison of measured fractal dimension to the analytical value for several fractals . . . 16
2 The effective diffusion coefficient, D_{eff}, and exponent, α, for Grimp RNAi . . . 33
3 The effective diffusion coefficient, D_{eff}, and exponent, α, for Control (pUGi) . . . 33

4 The effective diffusion coefficient, D_{eff}, and exponent, α, for different levels of illumination 33

1 Acknowledgements

The author would like to thank the members of the Vidali Lab for the use of their data. Without your hard work and experiments, none of this would be possible. The author would also like to thank Professor Erkan Tüzel for his continual guidance and advice over the course of the project.

2 Introduction

In the past decade, the moss *Physcomitrella Patens* has become a popular model organism for its simple development and powerful molecular tools, which include a complete genome sequence, rapid transient RNAi, and homologous recombination. Two aspects of the moss that have come under recent study are its tip growth and organelle motility.[1][2] Tip growth is an essential function for numerous species such as seed plants, where it is observed in their root hairs, and pollen tubes.[6] Recent experiments on *Pyscomitrella Patens* have revealed rich dynamic behavior of chloroplasts in response to light, as shown Figure 1. Both of these functions are believed to be mediated by the cytoskeletal network in moss. In order to gain further insight into the phenomena, and to develop a mechanistic understanding of the underlying mechanisms responsible, quantitive analysis is necessary. This project focusses on developing approaches which are able to provide quantitative metrics in two areas: moss morphology and chloroplast motility. Tools from theoretical biology and statistical physics were then used to explain the obtained data. We will first discuss the work done to describe moss morphology. This will include an introduction to common metrics that are used in describing morphology, followed by the idea of the box-counting fractal dimension, and its implimentation. Algorithmic growth models, which draw inspiration from L-systems [7], a class of string rewriting programs, are then presented to provide a better understanding of the data.

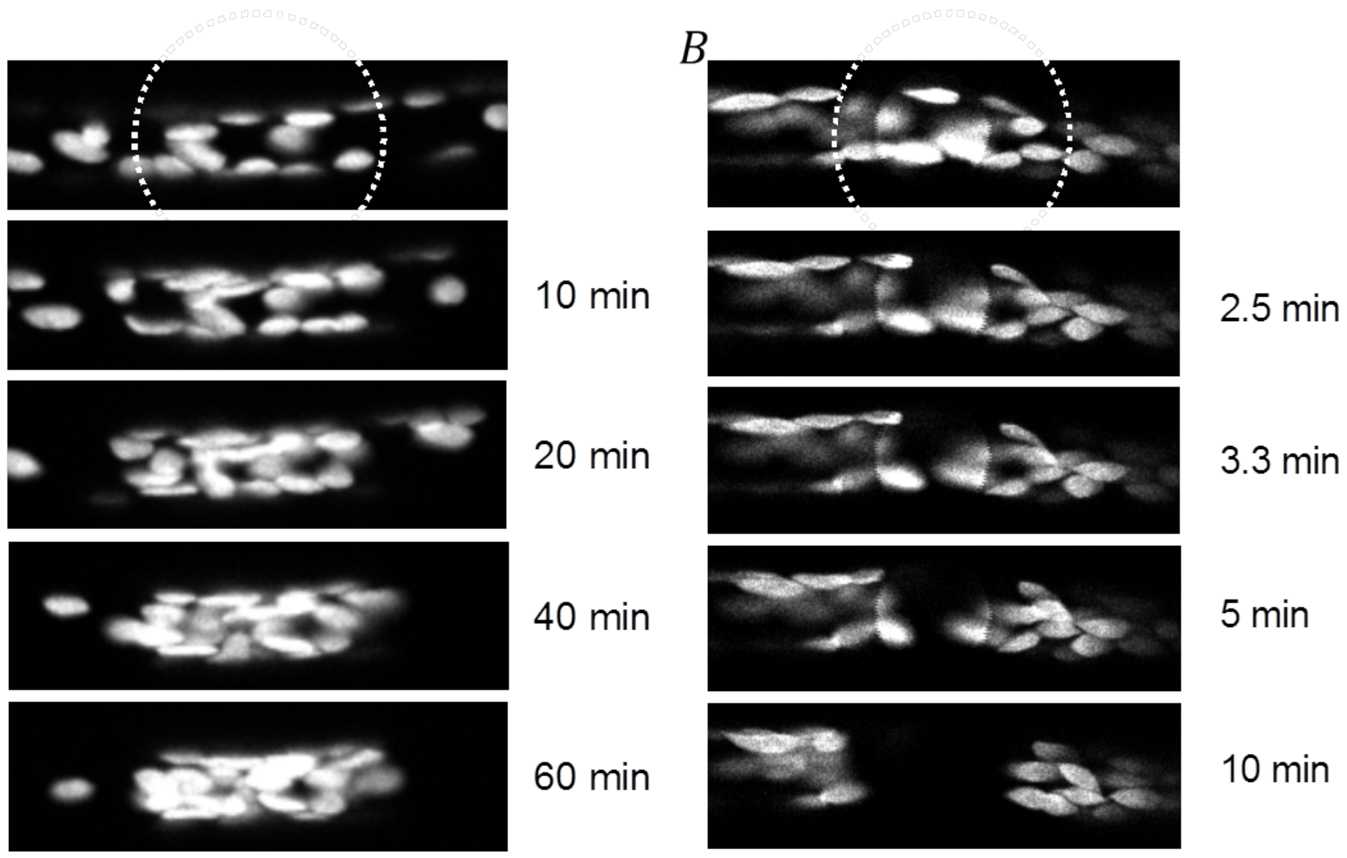


Figure 1: A) Chloroplasts will accumulate in a region of low intensity light, outlined here in dashes. B) In contrast, an avoidance response is displayed in reponse to high intensity light. The underlying molecular mechanism for this behavior is not yet understood in moss.

We will then discuss our work in organelle motility. We will begin by detailing a tracking code written to obtain information about the motion of organelles, specifically chloroplasts. We will then analyze the motility data using several metrics based on statistical mechanics. We end by providing a summary of the work and future directions.

3 Analysis and modeling of plant morphologies

3.1 Introduction

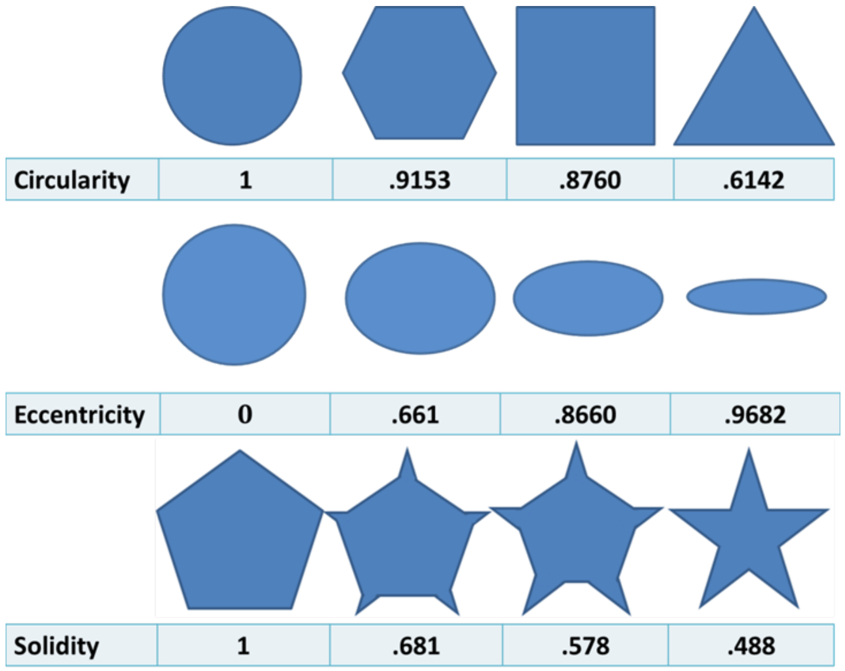
In the Vidali Laboratory at WPI, moss *Physcomitrella patens* is the model organism being studied. As part of their studies the Vidali Laboratory sought to characterize how moss grows. To compliment commonly used metrics, such as area and eccentricity, we sought to develop the use of fractal dimension to characterize moss morphology. In addition to the use of the fractal dimension, a simple model was developed to describe the growth of moss. We will begin this chapter by reviewing existing methodologies in Section 2.2. We will then discuss the definition and qualititative nature of the fractal box counting dimension in Section 2.3. In Section 2.4, we will discuss how the fractal dimension is calculated and present data attained from analyzing stacks of Physcomitrella projections. In Section 2.5 we will discuss the simple model that was developed to describe growth, and compare its quantitative results to the obtained data. We will end the chapter with a discussion.

3.2 Existing methodologies

In order to characterize different phenotypes, various metrics such as area, eccentricity, solidity, and circularity have been used.[3][4][5] Area describes how much space a plant occupies in a confocal image. The other metrics give insight into the nature of the plant's shape. For instance, the solidity is the ratio of the area to the convex area, where the convex area is the area of the smallest convex polygon which will enclose a body . Eccentricity describes how elongated an ellipse is. If we bound the projected plant image by an ellipse, the eccentricity of the plant is defined to be

$$\sqrt{1 - \frac{b^2}{a^2}} \, ,$$

where b is the semi-major axis and a the semi-minor axis. Starting at zero for a circle, it approaches unity as the ellipse increases its elongation. Finally, circularity is a ratio of the area to the perimeter squared, i.e. $C = \frac{A}{p^2}$. By convention, the circularity as defined above is scaled, by a factor of $\frac{1}{4\pi}$, such that a circle will have a circularity of one, i.e. $C = \frac{1}{4\pi}\frac{A}{p^2}$. This is the definition that will be used here. Representative values for the circularity, ecentricity, and solidity are shown in Figure 2.



Circularity	**1**	**.9153**	**.8760**	**.6142**
Eccentricity	**0**	**.661**	**.8660**	**.9682**
Solidity	**1**	**.681**	**.578**	**.488**

Figure 2: As the ratio between the area and the perimeter decreases, as shown in the polygons at top, the circularity decreases. The circularity measure is normalized to one for a circle. Eccentricity decreases as the ratio between the major and minor axes increases. As it is defined as the ratio of the convex hull area to the area, a polygon, like the pentagon shown at bottom left, will have a solidity of 1. If there are indents in the shape, the solidity will decrease.

Although each of the metrics above provide insight into some aspect of plant morphology, the branched nature of moss can potentially affect their effectiveness. As shown in Figure 3, a long branch can dominate eccentricity, solidity, and convex hull measurements, giving limited information into the general topology of the plant. For the convex hull, a problem of a different nature also arises. The algorithm used in finding the convex hull proceeds by iteratively refining a bounding box. Should the number of iterations programmed turn out to be insufficient, the numeric values obtained would be inaccurate, and subject to variation.

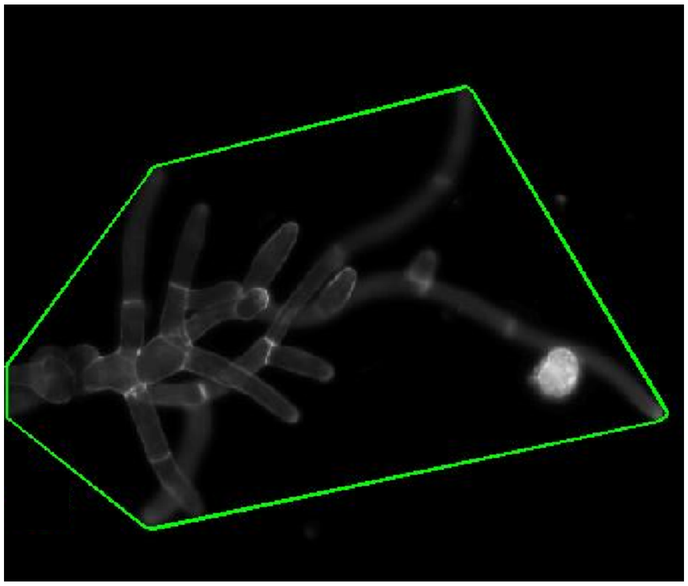


Figure 3: Shown is a microscopy image of a moss plant enclosed, in green, by its convex hull. The plant originates from the left, and the two long branches at right expand the bounding polygon.

3.3 Using fractals as a shape descriptor

Given the limitations of standard metrics outlined in the previous section, one desires a metric capable of describing how completely the entire plant fills in a space as it grows. One such metric is the fractal dimension , which we will illustrate below using Mandelbrot's well-known coastline problem. Suppose we want to find the length of the coastline of the UK. To do this, we first use some long length scale as shown in Figure 4 (left). This gives a coastline of 13 units long. We notice that with this large scale, we are not able to get into many of the bays, so we take a measuring scale half as long (Figure 4, middle). With this smaller measuring tool, we get a longer coastline measurement of 14 units length. Still there are inlets we cannot accurately measure with this shorter measuring length, so we cut the scale in half again. This time the total coastal length is 17.75 units. In principle, we can keep cutting the scale down to the point where we measure so finely as to have a characteristic scale going in and out at each grain of sand giving a practically infinite length. The challenge is then to properly describe the length of the coastline.

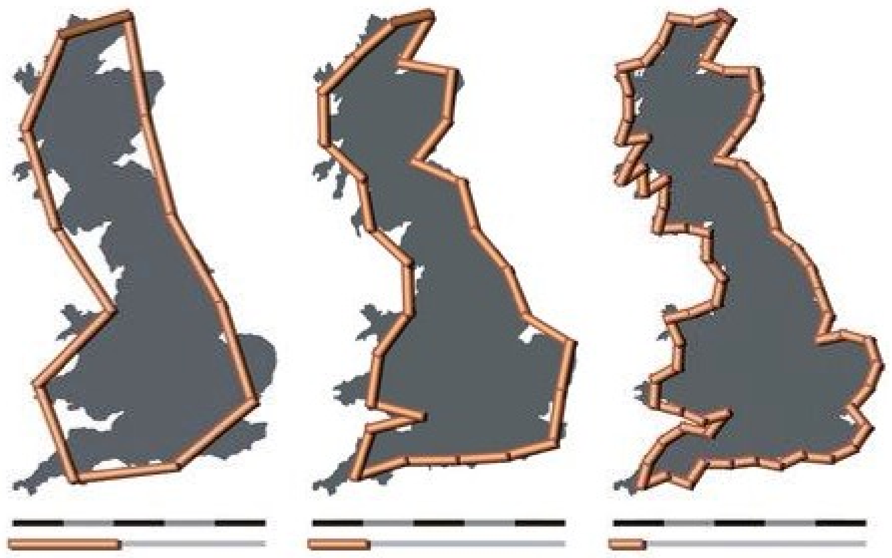


Figure 4: The coastline of the United Kingdom is traced using different length segments. As the segment length shortens, notice that one is able to more accurately trace the coast yielding a longer coastline.

As shown with the coastline problem, it can be difficult, if not , to determine an exact value for a metric accounting for scaling. We can document, however, how the coastal length changes as we alter the scale. This concept is the idea behind the fractal dimension, a potentially non-integer dimension value denoting how objects scale. Fractal dimensions are commonly used to describe the complexity of an object and how well it fills space. The metric used by Mandelbrot and Richardson, from whom he reproduced the plot shown in Figure 5 (taken from Ref [10]), is the slope of the total length vs. length scale graph on a log-log plot. For the west coast of Britain this gives a value of 1.25.

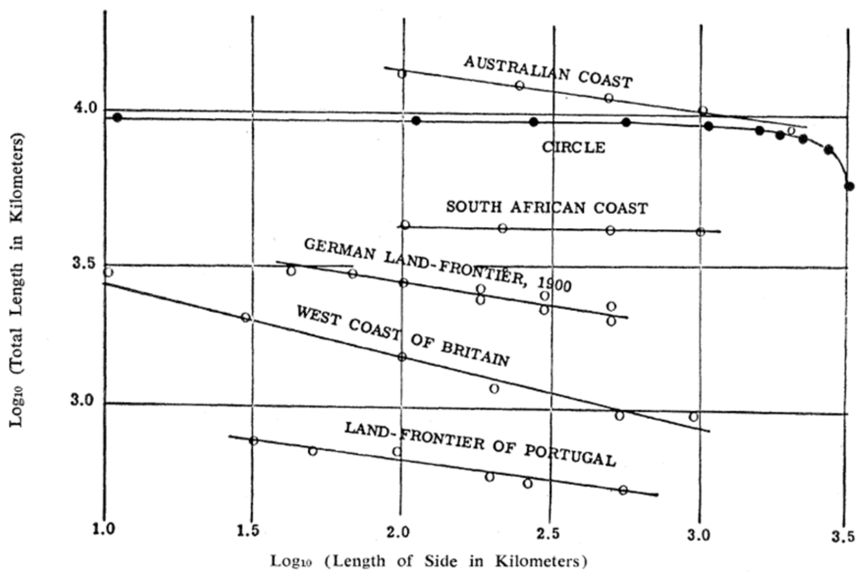


Figure 5: Although the actual value of the length of the coastlines change with the length scale they are being measured at, it changes in a linear fashion. We can use the slope of this linear change to describe how a system scales.

In order to calculate the fractal dimension of plant cells, drawing lines along the contours would not be practical. We, therefore, need a more generally applicable approach. For ease of implementation, we decided upon the Minkowski-Bouligand, or box-counting, method for calculating the fractal dimension. Similar to the coastline problem, the general idea of the box-counting is to ask how many boxes of some side length are needed to completely cover an object and how that number changes with box size. More precisely, given some set, S, and the number of boxes, N, of some side length, ϵ, needed to span the set, the fractal dimension is defined as

$$D = \lim_{\epsilon \to 0} \frac{log(N)}{log(\frac{1}{\epsilon})} \quad .$$

We should note that the fractal dimension, D, of an object may be greater than its topological dimension. Some well known fractals and their analytically calculated fractal dimensions are shown in Figure 6.

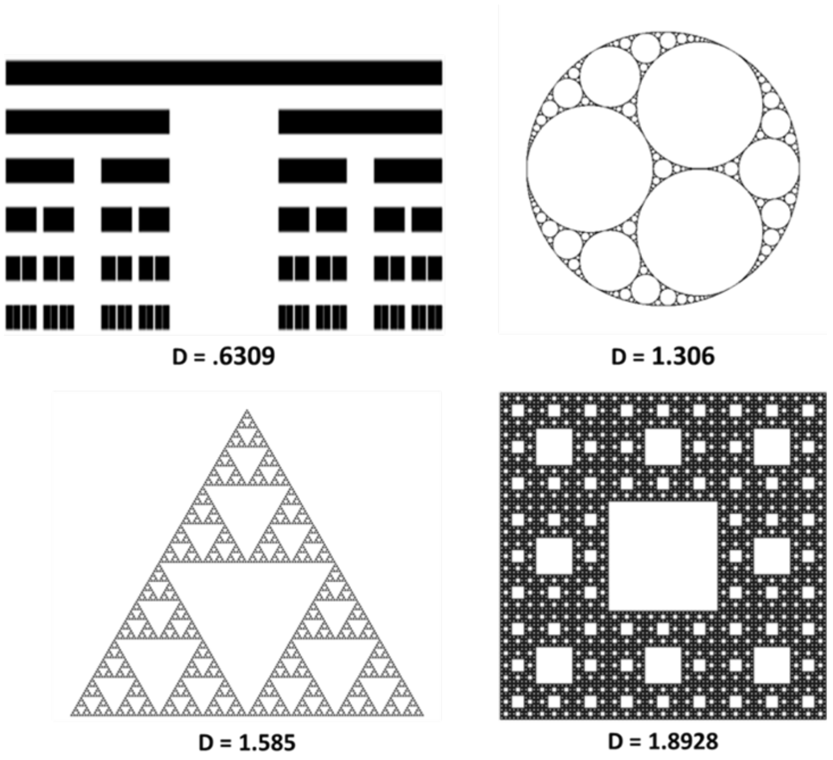


Figure 6: From left to right on the top row are the Cantor set and the Apollonian gasket. On the bottom row there is Sierpinski triangle and the Sierpinski carpet. The analytically calculated fractal dimension are also shown. Notice that with increasing dimension there is a tendency to fill in the space more completely.

3.4 Application of fractal dimension measurement to moss morphology

With mathematical constructs like those in Figure 6, it is usually possible to analytically derive the fractal dimension. However, Tto calculate the Minkowski-Bouligand dimension for an image however we need a systematic method with which to count the number of boxes needed for a collection of side lengths, hereafter referred to as a box-counting algorithm. To do the counting, a Matlab code (see Appendix A) was written using its image processing toolbox. The code first reads an image and does preliminary processing consisting of the application of an alpha-trimmed mean filter and a thresholding.

The alpha-trimmed mean filter operates by masking a region about a pixel. After ordering the values of all pixels in the region, the highest and lowest percentages, the alphas, are removed. The pixel is reassigned as the average of the remaining values. This is done simultaneously for all pixels in the image. A one-dimensional example is illustrated in Figure 7. A threshold removes most of the remaining noise.

Having cleaned the image, the code then identifies the objects in the image, and puts them into a list. From the identified objects, those objects touching the sides are removed, as the entire body of an object must be within the image to be analyzed. Those objects outside of an area threshold are removed as well. This is done to prevent small objects, which are either noise or moss cells which are mostly out of the image plane; and large objects, indicative of an overlap of multiple cells, from distorting the data. Having limited the list to good candidates, the counting may be performed. Looping through all the remaining objects listed, an image is created which isolates each object in turn.

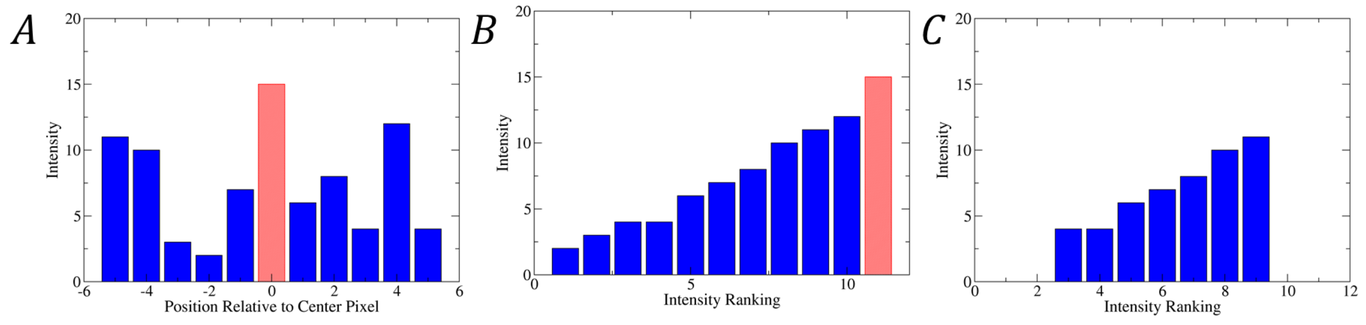


Figure 7: Alpha-trimmed filtering is illustrated in one dimension. A) The filter is applied to the "pixel" shown in red diagonals at the center. The initial value of the pixel is 15. The alpha-trimmed mean filter will work using it and the five pixels to either side of it. B) The filter starts by ordering the values. C) The highest and lowest values are then removed. Taking the average of what is left one finds $\mu \approx 7.4$. The "pixel" we started with is reassigned this value. Note that the pixel originally under consideration is no longer present. This is an example of the capability of the filter to remove outliers, either high or low, from the data. Should noise give a particularly high value at a point, the filter will lower it enough through the averaging that it will be removed in the later thresholding.

The next step is then to create a grid, in which the spacing of the grid is the effective box side length. The total number of grid boxes in which the object is present is recorded along with the effective side length. In the definition of the Minkowski-Bouligand dimension, we take the limit as $\epsilon \to 0$. With real data this is not feasible. Instead, the data is plotted on a log-log plot and a fit is performed. A step-by-step sketch of this procedure is outlined in Figures 8 and 9.

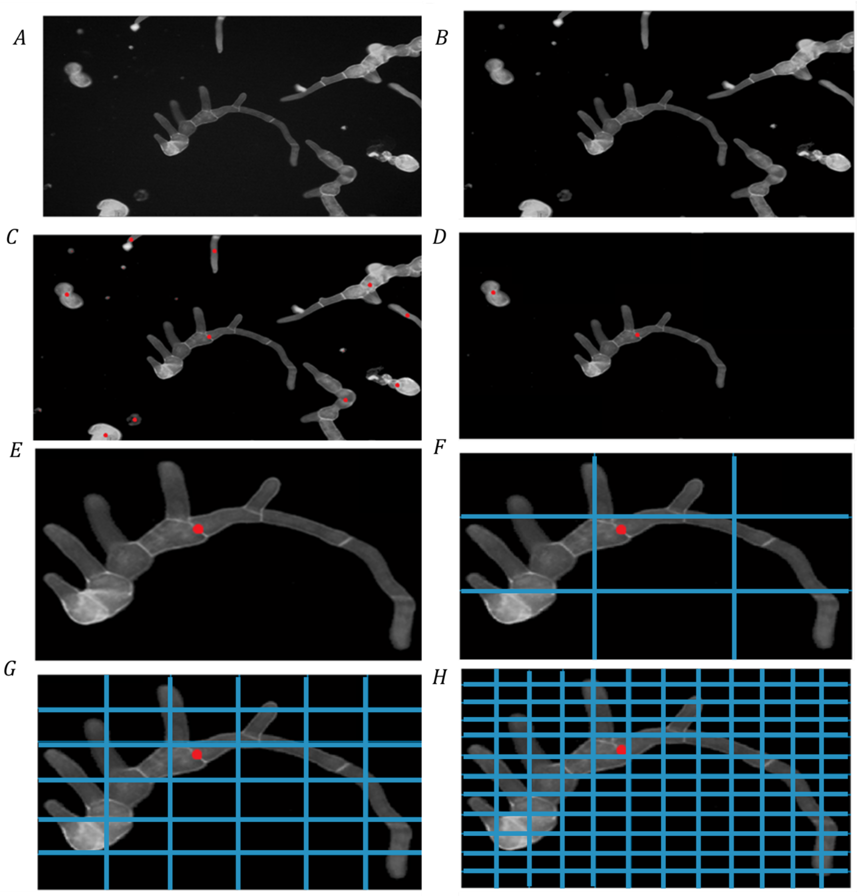


Figure 8: A) An image is loaded into the workspace. B) A threshold is applied and filtering done to clean up the image. C) With the image cleaned up all the objects in the image are identified. D) Objects that are touching the sides of the image or are outside of an area limit, either too high or low, are removed. E) A cell is selected and isolated as a separate image. F) A grid with a given spacing is overlaid on the image and the number of grid boxes in which the cell appears (7 in this case) are counted. G-H) This process is repeated iteratively with finer grids.

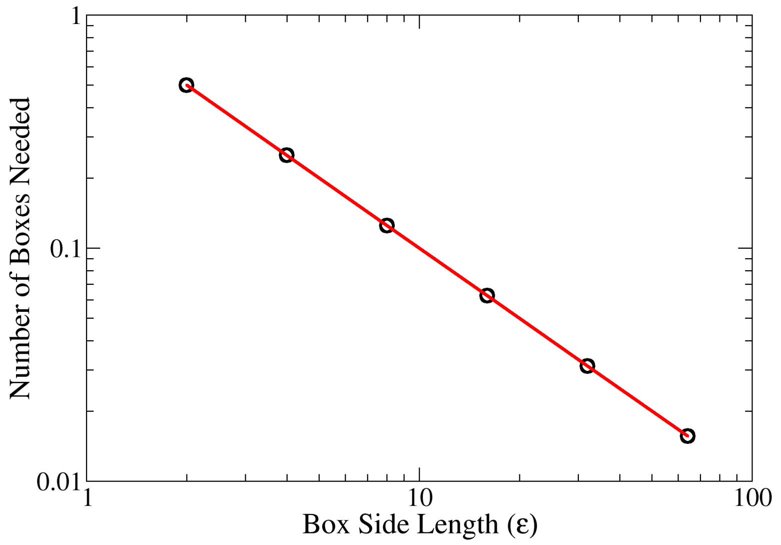


Figure 9: Having repeated the process of Figure 8 and obtained a a sufficient number of points to perform a fit, we plot the number of boxes needed to cover the cell vs. the size of the boxes (the grid spacing) on a log-log scale. A linear fit is done in order to determine the fractal dimension. Note that the grid sizes used here are large for demonstration purposes. When actually calculating the box-counting dimension we want to avoid using too large a box size as the number of boxes the cell is in will saturate, skewing the fit significantly.

We then tested the algorithm we developed in Matlab against the analytical value of the fractal dimensions for several known fractals. The results are shown in Table 1. The values obtained are reasonable, especially given the limited number of iterations with which the fractals were generated. Confident in the accuracy of the algorithm after comparing it to the analytical values, we used it to characterize the physical data.

Table 1: The fractal dimension with the box counting algorithm gives values for the fractal dimension which are reasonably close to the analytical values of the fractal dimension for several well known fractals.

Fractal	Analytical Fractal Dimension	Measured Fractal Dimension
Cantor set	.631	.692
Apollonian gasket	1.31	1.28
Sierpinski triangle	1.59	1.54
Sierpinski carpet	1.89	1.82

After running the code, the data was separated by the specific cell line of moss and day of growth. There was a total of 5,295 cells analyzed. The results are shown in Figure 10. Both area and fractal dimension show a clear day to day increased. After the first day, no overall trend was shown in solidity. Thus the area can provide a metric for comparing samples between different days but, given its definition, does not provide any information about the shape of moss. The solidity gives insight into morphology but it cannot be used to distinguish between days. The fractal dimension possesses the best aspects of the area and the solidity; it changes from day to day, and it provides a descriptor of the shape of moss. As a tradeoff, it can be harder to interpret what a specific value of fractal dimension means.

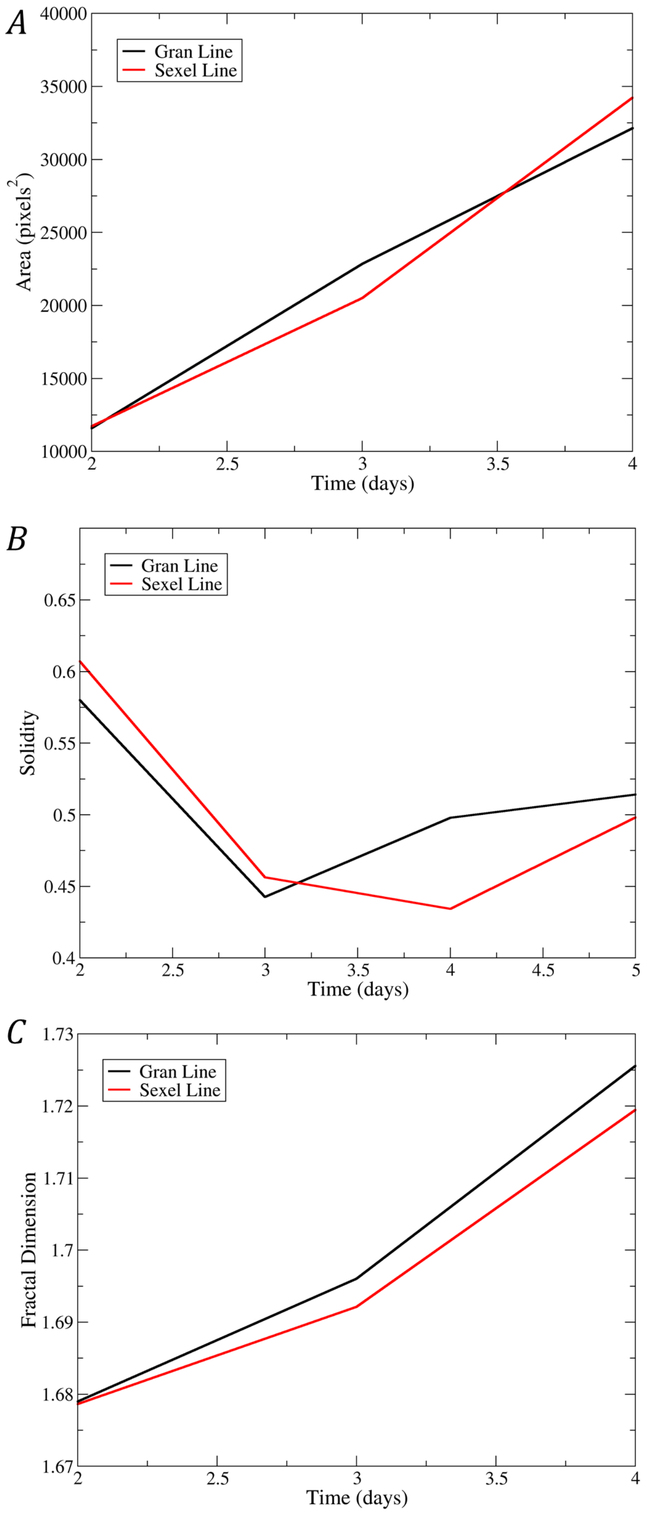


Figure 10: A) The area steadily increases over time but it provides no insight into the shape of the moss. B) The solidity provides insight into the shape but it does not show a progression over time. The fractal dimension both has a day to day progression and provides a descriptor of morphology. It can however be somewhat difficult to interpret.

3.5 A simple model of plant growth

In order to better understand the data describing moss morphology over time, a model of the growth is necessary. Constructing a growth model taking into account all the underlying physical processes is a challenging task and is beyond the scope of this project. Instead, here we seek a simple phenomenological model to explain our observations. In the literature, a common method used in describing plant growth is L-systems.[7] Named after Aristid Lindenmayer, L-systems compose a class of string rewriting models. They begin with an initial string and a set of rules specifying how the string is to be rewritten. Proceeding iteratively from the initial string, all the characters are rewritten simultaneously at each step, as illustrated in Figure 11.

Rules
$a \rightarrow bc$
$b \rightarrow a$
$c \rightarrow b$

a
bc
ab
bca
$abbc$
$bcaab$
$abbcbca$
$bcaababbc$

Figure 11: On the left are the string rewriting rules. On the right, a sequence of characters are shown after the applications of these rules in successive steps to an initial string "a".

In order to construct patterns or images, a second set of rules is introduced which assigns a drawing command to every character. For example for the strings of Figure 11, we could define a as make a line, rotate by θ degrees, and then draw a second line. b and c could be assigned similar graphical rules as well and with them one could convert the string of characters to an image. This technique has proven successful in modeling the growth patterns of feathers, trees , ferns, and other organisms. Several years ago, F. Fracchia of Simon Fraser University and N. Ashton of the University of Regina attempted to model *Physcomitrella patens* using L-systems.[8] They used 9 cell types (their strings) and 13 rules to achieve a reasonable resemblance to Phycomitrella cells. Their approach, however, is a simplified

L-system and taking into account other factors, such as branching angles and growth rates, would involve 65 production rules.

Given the required level of sophistication, we felt that using L-systems for our task would prove difficult and minimally fruitful for the amount of time which would need to be invested. Instead we drew inspiration from their algorithmic approach to plant growth. As a first attempt at modeling the system, we sought to capture the following several basic elements of moss growth:

1. Moss grows in branches.

2. These branches are made up of segments. The segments are of variable length.

3. Branches grow by the creation of a new segment at its end. Any grown segment curves from the path of the previous section. The amount of curvature varies.

4. Periodically a new branch will be formed by budding off of a branch. This budding occurs at variable angles.

In addition, we observed that branching tended to occur, on average, after there were four segments in a branch. Once four segments were made it was observed that every segment at least three segments away from the end had a branch coming from it. We then measured the following three quantities: the lengths of segments, the angle of curvature made by branches when growing, and the branching angle, the angle at which a bud deviated from the stalk it budded from. The measurement methods and their distributions are shown in Figure 12.

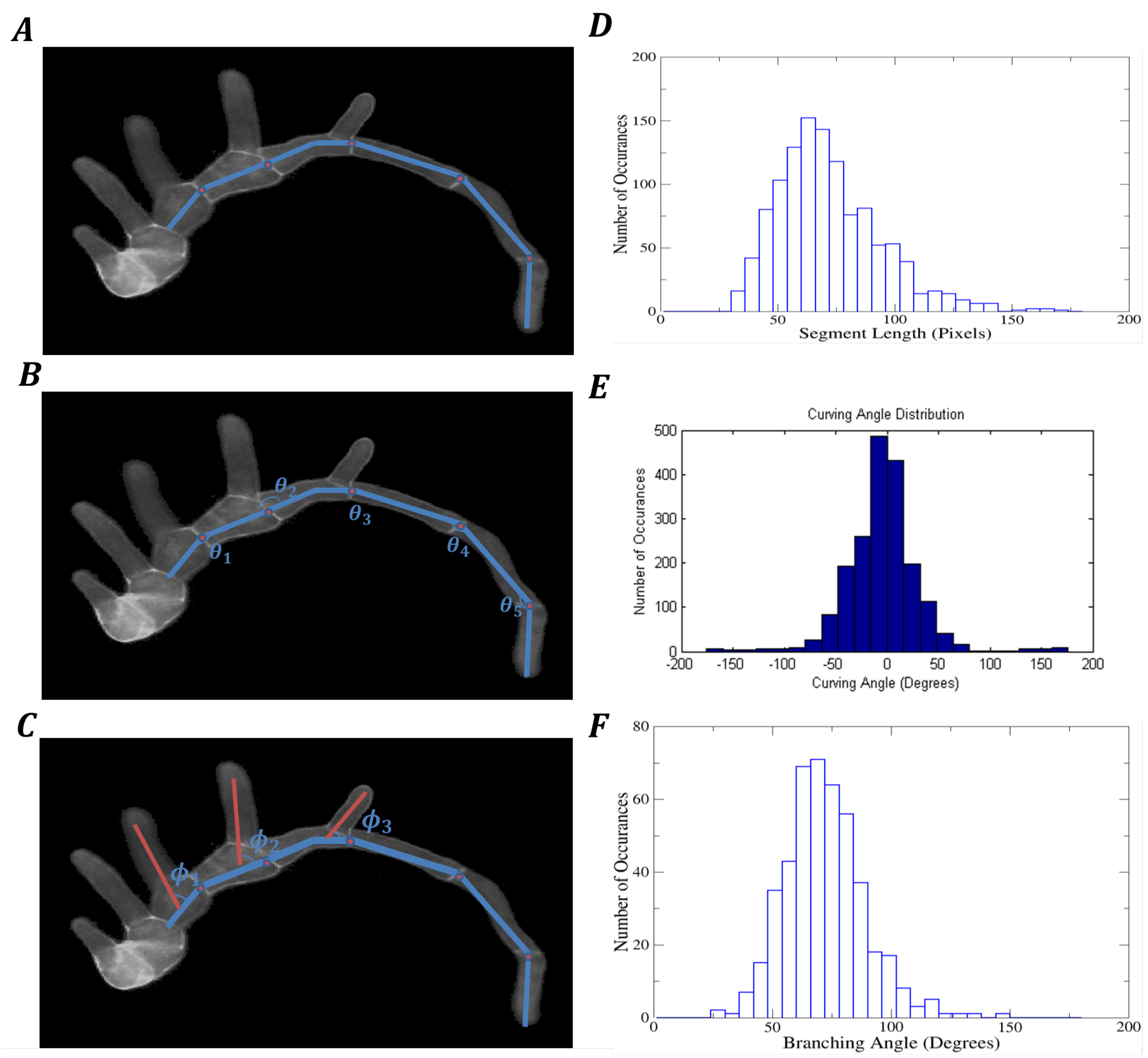


Figure 12: A) When getting the lengths of the segments, measurements were made from the center at one end of the segment to the center at the other end of the segment. B)Using the center-center line that was used to calculate the length as orientation direction of each segment, the curving angles, θ_i, were determined between adjacent segments. C) Again making use of the center-center line, the branching angles , ϕ_i, were measured. The results of the length, curving angle, and branching angle measurements are shown in D),E), and F) respectively.

Using the distributions and growth pattern observations we came up with the following growth algorithm:

1. Choose a number of segments, N, desired on a branch. This will be the branch's length.

2. Pick a starting angle and place an intial segment, n_1, oriented at it.

3. Repeatedly append segments to the end of the branch until the number of segments reaches the appropriate length. With each addition, the orientation angle will be changed by some amount, $\delta\theta$, based on the measured curving angles.

4. If the length of the branch, N, is at least four, then at each segment greater than three segments, numbered as n_i, away from the end, n_1,n_2,...n_{N-4}, a new branch will be produced. The angle the new branch grows in will be the orientation angle of the segment it grows from, θ, offset by a branching angle, ϕ. After placing an initial segment at the specified angle, the branch grows according to steps three and four here. The number of segments on the new branch is set as $N-(3+i)$. This length is chosen based on the assumption that all branches grow at equal rates and that a branch will only start growing once its parent branch reached the minimum required length, four segments.

Each segment's length, curving angle, and branching angle are drawn from a normal distribution whose mean and standard deviation match those found experimentally. An example of the resulting growth is shown in Figure 13.

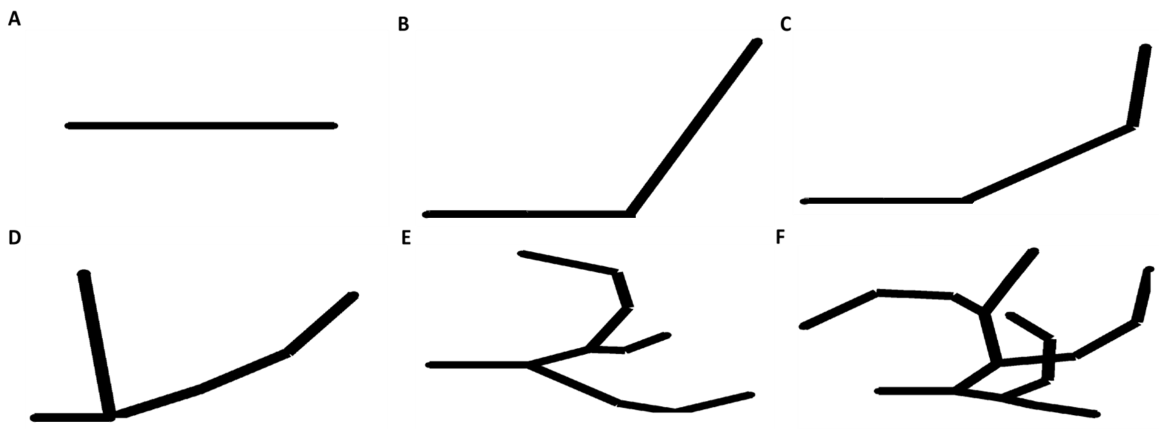


Figure 13: A) The algorithm starts by laying down a single segment. B-C) Segments are appended to the end of the previous branches. Their orientation is offset from that of the segment they were appended to. D) If the initial branch is sufficiently long, a secondary branch will form. E) With a longer initial branch, more secondary branches are created. Notice that as one moves along the main branch that the secondary braches are smaller as one approaches the end. F) With a long enough initial branch, secondary, and later tertiary, branches will begin to have buds as well.

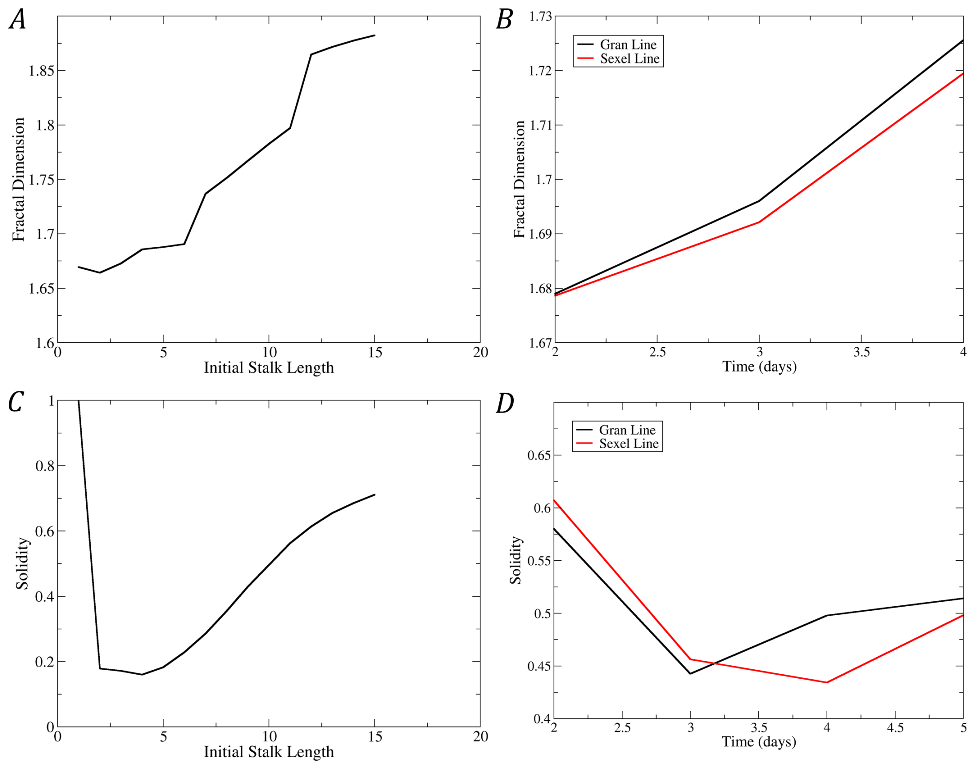


Figure 14: A,B) The model's fractal dimension fit the trend of the experimental data reasonably well . C,D) The solidity, despite having the qualitatively correct shape, is quantitatively wrong. The simulated values lie well outside the bounds of the experimental data.

Creating one thousand images for each initial stalk length from one to twenty yielded the results shown in Figure 14. Both the fractal dimension and the solidity, Figure 14 A,C, exhibit overall characteristics similar to the experimental data, Figure 14 B,D. The fractal dimension starts at appromiately 1.68 and grows assymptotically towards two. Considering the solidity, we do see a somewhat qualitatively correct figure however quantitively it is incorrect. It begins at an initial high point of 1 before dropping to arround .2, both of which lie far outside the range of the experimental data.

Building upon the initial model, it was noted that the plants in the raw images grow from ball-like regions. We will call these regions bulbs. It was observed that the number of bulbs varied and that each bulb could have one, many, or no branches growing from it. Distributions, Figure 15, were obtained for the number of bulbs present on each organism, the number of bulbs that were active, i.e. had a branch growing from it, and the number of branches per active bulb. The model was modified such that a plant would initially have a number of bulbs present. The number of those bulbs which were active was then chosen, and those bulbs that would be active were selected. For each active bulb, a number of branches was drawn. Each branch would grew at the same time per the previously outlined rules. Examples of this growth are shown in Figure 16 A and B. Thus each had the same length at any given step. Using this model, the results shown in Figure 16 C,D were obtained. The fractal dimension displayed the same characteristics as before. However, the solidity decreased to .43, on the order of the experimentally observed minimum, while having a similar qualitative development with increased length as the experimental data did over time.

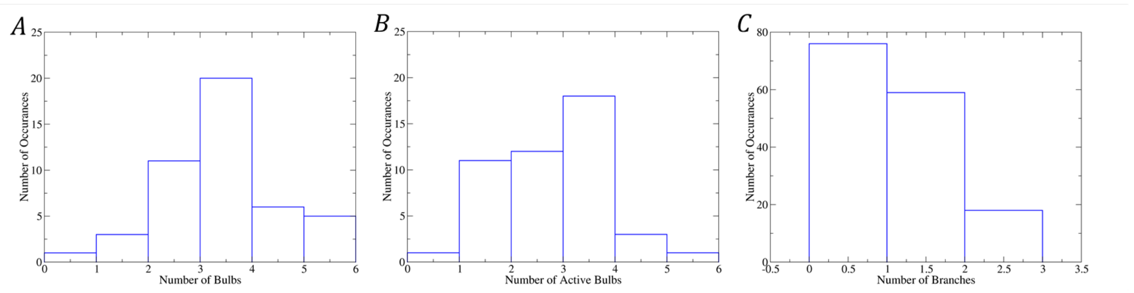


Figure 15: Histograms of the number of bulbs, number of active bulbs, and the number of branches on each bulb on moss plants are shown in A, B, and C respectively.

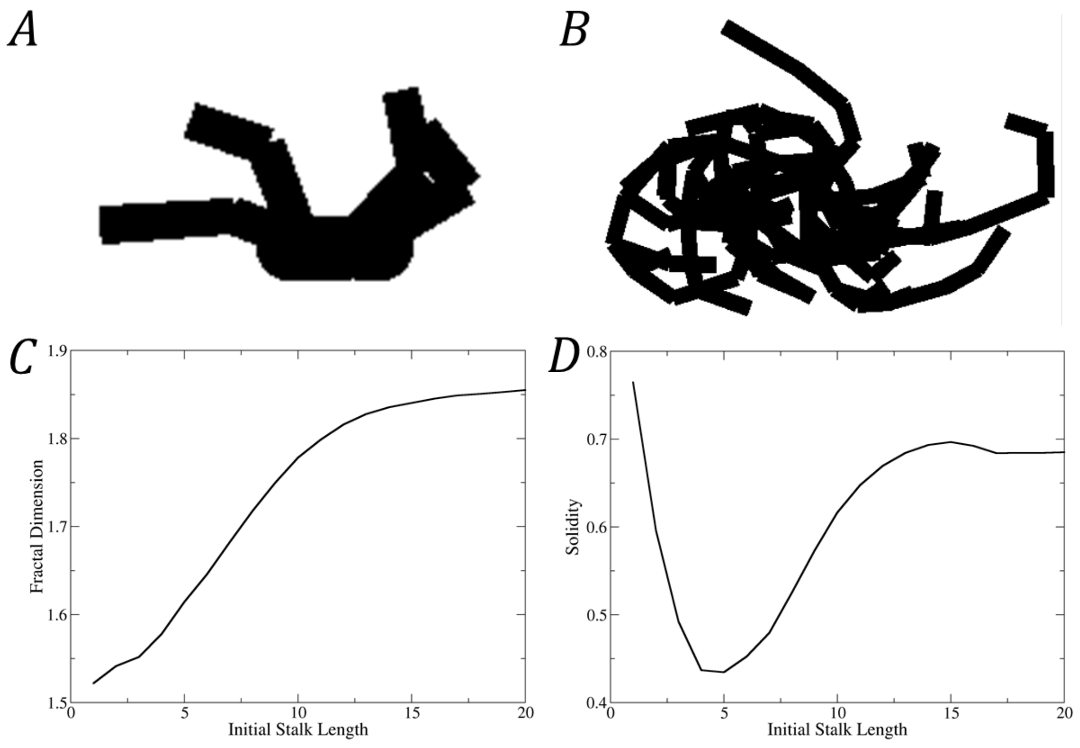


Figure 16: A,B) Branches grow from multiple bulbs. Each branch follows the same growth rules as the first model. C) The fractal dimension displays the same characteristics shown in the previous models. D) By introducing multiple bulbs and branching points, the solidity decreased to approximately .45 before increasing. Unlike the decrease to values of .2 in the previous models, .45 agrees well with the values that were found experimentally.

3.6 Summary

In characterizing moss morphology there are several common metrics. As part of this project we sought to develop the fractal dimension as another tool by which to analyze physical data. To do this, a code implimenting the box counting algorithm was developed, and the fractal dimension was calculated. In order to interpret the results, a course-grained model of plant growth was developed. Although this model effectively described the fractal dimension, it was unable to capture the qualitative aspects of solidity seen in the data. Several observations about the points where the plant grows from and the number of such points were made. Adopting these features into the model, we were able to replicate the values of fractal dimension and solidity found experimentally.

4 Quantitative analysis of organelle motility

4.1 Introduction

In addition to studying cellular growth and organization, we would like to understand chloroplast motility. Moss optimize for growth in the light they are exposed to in their early stages of growth. Responses of chloroplasts to different light conditions show how sophisticated this molecular machinery is. The cytoskeletal network, composed of microtubules and actin, and the molecular motors associated with each, is known to play a key role in organelle transport in cells. Our goal here is to examine the involvement of the cytoskeleton in the transport of chloroplasts. We did this by analyzing the trajectories of organelles in two experiments conducted at the Vidali Lab; one in which plants were under different lighting conditions and a second with different drug treatments. To study the trajectories, a tracking algorithm was developed. It will be detailed in Section 3.2. The use of moments of displacement to describe trajectories will be detailed in Section 3.3 followed by results from experimental data in Section 3.4. We end with a summary in Section 3.5.

4.2 Development of an algorithm for tracking chloroplasts

To proceed quantitatively in the study of organelle motility, one needs the ability to track organelle positions over time, and metrics to analyze the data. We will describe our tracking approach first. Since chloroplasts autofluoresce in red, they can easily be recorded in the red channel of a confocal microscope. This makes them ideal for tracking. The difficulty arrises from the anisotropic shape of the chloroplast. As most tracking packages assume a spherical object, a Matlab script had to be written for the task. Similar to the image processing done in the fractal analysis, we set the conditions that all chloroplasts must meet in order to be tracked, specifically that they are between a high and low area threshold, and are not in contact with the edge of the viewing window. Those conditions defined, the code, shown in Appendix C, operates as follows:

1. The user inputs the name of a stack of images in .tif format to be analyzed.

2. The code loads the first frame of the stack and applies the alpha-trimmed mean filter and a threshold. To make use of tools in Matlab's image processing toolbox, the frame is then converted to a binary image.

3. All chloroplasts in the image satisfying the outlined conditions are identified. Their centroid locations are calculated and recorded.

4. The next frame is loaded, and steps 2-3 are repeated. If any of the identified chloroplasts' regions envelop the location of a centroid identified in the previous frame, it is assumed that the chloroplast is the one that was associated with the centroid at a later time. The updated location of the current frame is appended to the previous centroid location associated with that chloroplast. Should an object be identified which does not correspond to a previously identified chloroplast, a new entry is created and the centroid recorded. Also, should an identified chloroplast envelop multiple centroids, it is assumed that the identified chloroplast is in fact an artifact of an overlap of two, or more, previously separate chloroplasts; as it is not a single chloroplast, it is removed as a candidate for tracking.

5. Step 4 is repeated until all frames have been analyzed. The data is then saved.

This method has the advantage that it runs quickly and is relatively straightforward to implement. However, the simplicity of the algorithm comes with drawbacks. It operates under the assumption of a constant shape. However, chloroplasts can divide and possibly be imaged during their division process. The algorithm also cannot be used to identify and track chloroplasts that form clumps. Although one is able to distinguish chloroplasts in an image by eye, when converting to a binary image the chloroplasts become a single entity, wherein individual chloroplasts are difficult to resolve. Given this limitation, we confine ourselves to the study of isolated chloroplasts.

4.3 Moments of displacement

Having developed a tool to track chloroplasts, we decided to use the mean squared displacement and higher moments to characterize the system. Given a particle undergoing agitation in uniformly random directions, the average displacement from its original position will be zero by symmetry. However, taking the square of displacement it can be shown that the mean squared displacement over time will go as

$$< x^2 >= 2Dt,$$

where D is a positive real constant which we call the diffusion coefficient. This random motion is called diffusive motion. To see how this comes about, consider a particle moving in one dimension. We will assume that it moves in discrete steps. At each step the particle can go left or right some distance d. The probabilities to step left and right will be denoted p_l and p_r, respectively. Suppose that the chance of taking a step in each direction is equal, i.e. $p_l = p_r = p = 1/2$. The average position of the particle after N steps is then given by

$$< x >= N(p_r d + p_l(-d)) = N(\frac{1}{2}d - \frac{1}{2}d) = 0.$$

The mean squared displacement, on the other hand, can be written as

$$< x^2 >= N(p_r d^2 + p_l(-d)^2) = Nd^2.$$

If the amount of time for each step is τ, then the total time t would be $t = \tau N$. Using $N = \frac{t}{\tau}$, the expression for the mean squared displacement becomes

$$< x^2 >= \frac{t}{\tau}d^2.$$

If we define the diffusion coefficient as $D = \frac{d^2}{2\tau}$, we recover the desired result,

$$< x^2 >= 2Dt.$$

Although one could simply look up this relationship and its derivation in any standard statistical mechanics text, by going through it one realizes that the thought process behind the derivation can be used as a quick test of hypotheses about underlying mechanisms of transport. For instance, let us consider the case in which an element of the system is suspected to inhibit the motion. As a first guess, one can suppose that an obstacle periodically prevents particles from moving. More specifically, one can say that there is a probability, ϕ, that the particle does not take a step. Due to this element, $p_l = p_r = p = \frac{1}{2}(1-\phi)$. It follows that

$$< x >= N(p_l(d) + p_r(-d) + \phi(0)) = Nd(\frac{1}{2}(1-\phi) - \frac{1}{2}(1-\phi)) = 0,$$

and

$$< x^2 >= N(p_l(d)^2 + p_r(-d)^2 + \phi(0)) = (1-\phi)Nd^2 = 2D(1-\phi)t. \qquad (1)$$

This result means that in the presence of an obstacle, the nature of diffusion doesn't change but the diffusion coefficient is rescaled by $1-\phi$.

In genera,l however, it is possible that the mean-squared displacement is given by

$$< x^2 >= 2D_{eff}t^{\alpha},$$

for some non-negative α. For systems with pure diffusion motion, $\alpha = 1$. If the forces are directed in some way, one gets a value of $\alpha > 1$. For example, an object traveling at constant velocity in a direction moves according to $x = vt$. Its mean squared displacement is then $< x^2 >= v^2t^2$. Thus, in the case of constant velocity $\alpha = 2$. If motion is somehow inhibited from moving it will have $\alpha < 1$. An example where this would occur is crowding.

In summary, the mean squared displacement of a trajectory can serve a quantitative metric to describe how directed a trajectory is.

Note that D_{eff} used above stands for an effective diffusion coefficient. Its units will differ from the diffusion coefficient defined previously for $\alpha \neq 1$. One can obtain a diffusion coefficient with correct units is by allowing the diffusion coefficient to change over time and define a trajectory's mean squared displacement as

$$< x^2 >= 2D(t)t$$

Work is being done to determine how best to define $D(t)$. Wu and Berland have proposed an instantaneous diffusion coefficient, $D_{inst}(t)$, defined as the slope of the mean squared displacement as a function of time. In addition to being dimensionally correct, the diffusion coefficient defined as such enables one to determine the propagator in a stochastic differential equation formulation of Brownian motion.[11]

In addition to calculating the mean squared displacement, the second moment of the displacement, one can also investigate its higher moments. For some moment, q, the average q^{th} moment of displacement can be described as

$$< x^q >= 2D(q)t^{\alpha(q)}.$$

Under most conditions, plotting $\alpha(q)$ will give a plot which will consist of multiple linear sections with different slopes. Each slope is associated with a different transport mechanism. By counting the number of linear sections we can gain an understanding of how many mechanisms or modes of transport are present in the physical system. Further, if $\alpha(q)$ is nonlinear it is indicative of rare events, such as brief periods of rapid motion or jumping, occurring in a trajectory.

4.4 Experimental Results

In the Vidali Lab, two sets of experiments were conducted to study organelle motility. The first set of experiments was to determine how chloroplasts respond to light. A control phenotype was grown under red light. It was then imaged with a confocal microscope under a red light and under blue and red light at different intensitites of blue light, five percent and ten percent. The second set of experiments examined the influence of the cytoskeletal network on chloroplast motility in moss. Two phenotype lines, Control (pUGi) and Grimp RNAi, were cultured. Cells from each line were given one of several drug treatments prior to imaging: a control with no drug added, Latrunculin to depolymerize the actin leaving only microtubules, Oryzalin to depolymerize microtubules leaving only actin, a treatment of both Latrunculin and Oryzalin which completely removed the cytoskeleton, and a treatment of EtOH, a solvent used in introducing Latrunculin and Oryzalin.

The mean squared displacements for the different light and drug conditions are shown in Figure 17 A,B,C. Since the data is noisy, performing fits is difficult, if not impossible. To address this, a recursive averaging method was implemented. Typically one considers the displacement of an object relative to its position in the first frame. In the recursive averaging method, given an object in one frame, the displacement of that object is calculated relative to its position in all previous frames. The displacement will be calculated from frame two on relative to the position in frame one, from frame three on relative to the position of frame two, and so on. Using every frame as a starting point in this manner gives cleaner trajectories, shown in Figure 17 D,E,F. Fitting a power law to each of the recursively averaged curves yields the values shown in Tables 2, 3, and 4.

Examining the fitting results of the drug treatment conditions for both Control (pUGi) and Grimp RNAi, a few common trends emerge. With the entire cytoskeleton intact, the chloroplasts' motion is superdiffusive. Upon removing both cytoskeletal components, the motion becomes subdiffusive. This supports the notion that the cytoskeletal network as a whole serves to, in some way, facilitate the motion of the organelles. Further analysis

Table 2: The effective diffusion coefficient, D_{eff}, and exponent, α, for Grimp RNAi

Drug Condition	D_{eff} (95% confidence interval)	α (95% confidence interval)
Control	.0994 (.0944,.105)	2.04 (2.03,2.05)
EtOH	.710 (.684,.735)	1.74 (1.72,1.75)
Latrunculin	1.03 (.968,1.09)	1.56 (1.55,1.58)
Oryzalin	.0941 (.0474,.141)	1.06 (.952,1.18)
Latrunculin and Oryzalin	.995 (.868,1.12)	1.08 (1.04,1.12)

Table 3: The effective diffusion coefficient, D_{eff}, and exponent, α, for Control (pUGi)

Drug Condition	D_{eff} (95% confidence interval)	α (95% confidence interval)
Control	.547 (.513, .582)	1.51 (1.49,1.52)
EtOH	.485 (.451, .519)	1.45 (1.43, 1.46)
Latrunculin	.0794(.0758,.0830)	1.57(1.57,1.58)
Oryzalin	.0412 (.0388, .0436)	.967 (.935,.999)
Latrunculin and Oryzalin	.531 (.486, .576)	.694 (.630,.757)

Table 4: The effective diffusion coefficient, D_{eff}, and exponent, α, for different levels of illumination

Light Condition	D_{eff} (95% confidence interval)	α (95% confidence interval)
90% Blue + Red	.0127 (.0125,.01302)	1.49 (1.48,1.50)
5% Blue + Red	.0201 (0209,.0221)	1.46 (1.45,1.47)
Red	.0441 (.0429,.0452)	1.39 (1.39,1.40)

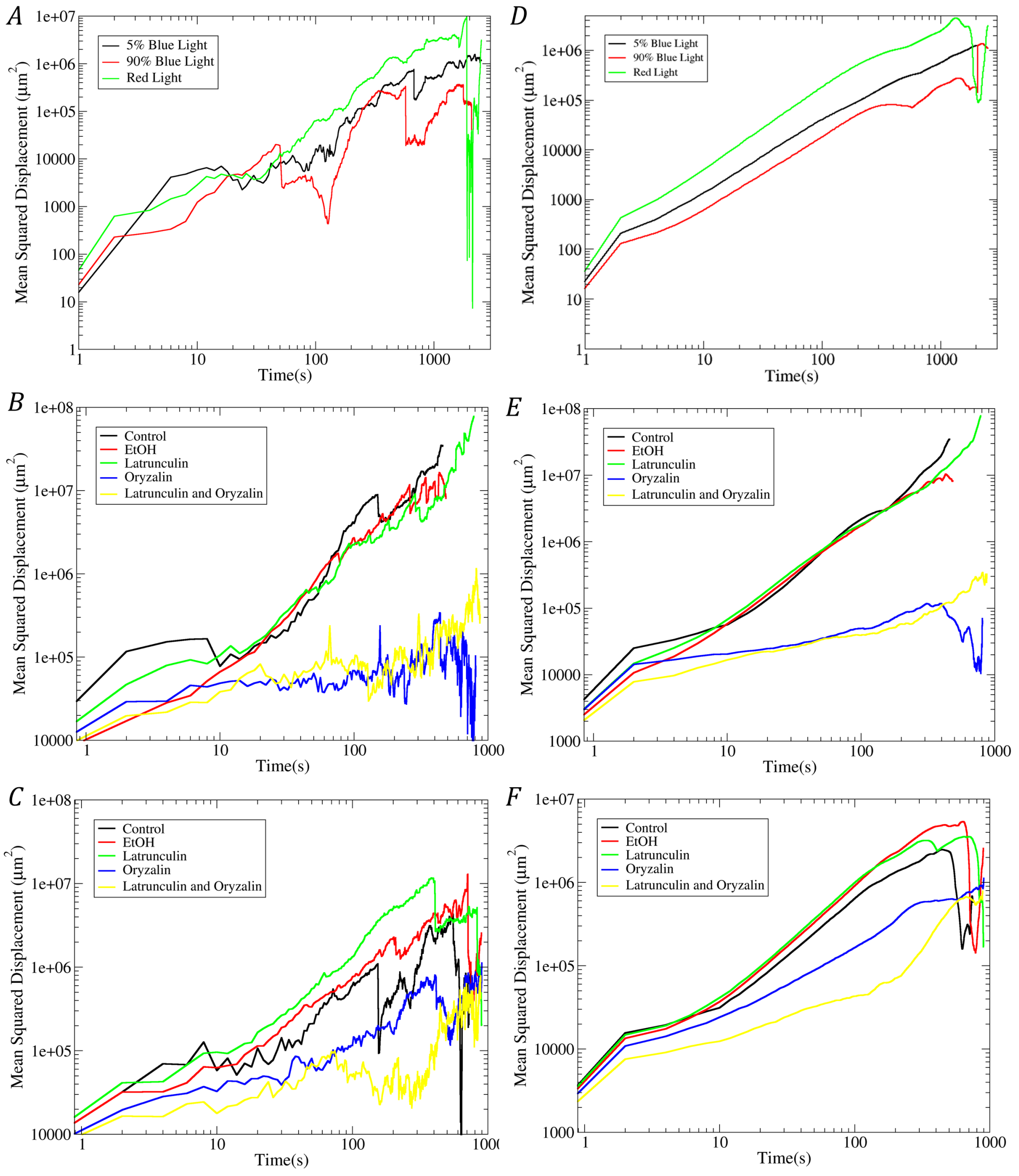


Figure 17: A-C) The mean squared displacement as a function of time for the light, Grimp RNAi, and Control (pUGi) conditions, A,B, and C respectively, is noisy. This makes getting a reasonable fit difficult. D-F) By doing a recursive averaging, explained in the text, one can obtain cleaner trajectories. Trajectories obtained using the recursive averaging for the light, Grimp RNAi, and Control (pUGi) conditions are shown in D, E, and F respectively, each to the right of the trajectories without the recursive averaging for the same condition.

of the results shows that upon removing the actin network, the diffusion exponent's value decreases but the overall motion remained superdiffusive. However, upon removing the microtubule network, the chloroplasts' motion became diffusive or subdiffusive. This suggests that in moss, the microtubules facilitate the motion of chloroplasts, presumably via molecular motors, in the absence of the actin cytoskeleton.

Upon changing the intensity of blue light, we see that the exponent undergoes a relatively slight change. On the other hand, a more dramatic effect on the diffusion coefficient is observed. The reduction of blue light intensity from 90% to 5% resulted in a change of the effective diffusion coefficient from .012 to .021. For comparison, the red light only condition had a diffusion coefficient of .044. These features can be seen qualitatively in Figure 17 D by noting the vertical offsets between the curves and their near parallel slopes. Let us now recall the derivation of mean squared displacement in the presence of obstacles, i.e. Eq.(1). In that scenario, the diffusion coefficient was decreased proportional to the chance of stalling while keeping the exponent the same. This mimics what was shown experimentally. The diffusion coefficients changed depending on the light intensity observed with little effect on the exponent. This suggests the involvement of a passive cross-linker.

Although determining the exact biological agent responsible for the change in motility is a difficult task, we can get more information about the types of mechanisms involved through analysis of the higher moment plots. There was insufficient data to perform the analysis on the drug treatments as any noise or variance in the system gets amplified with the higher powers. However, there was sufficient data to investigate higher moments in the case of differnt light intensitites. The plots of the power coefficient, $\alpha(q)$, are shown in Figure 18. For the 90% and 5% maximum intensity blue light nonlinear relationships were shown. This suggests that under blue light there is some mechanism of transport which rarely manifests itself. This behavior is not observed under pure red light. For the red light data $\alpha(q)$ is nearly linear.

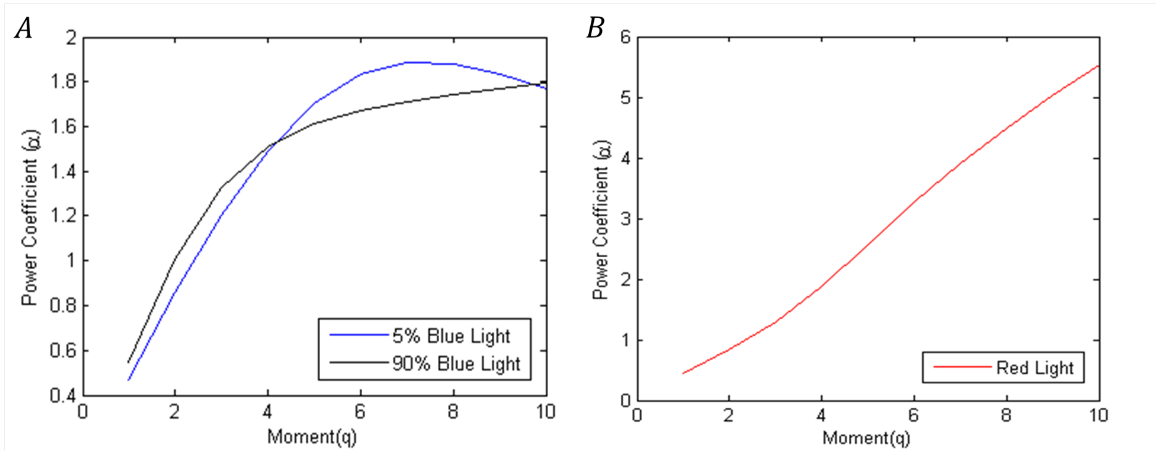


Figure 18: A) The coefficient, $\alpha(q)$, for both of the blue light conditions exhibited nonlinear behavior indicating rare events. B) The pure red light condition was composed of seemingly linear sections suggesting a series of mechanisms working at all times.

4.5 Summary

Through the development and implementation of a tracking code, data describing the trajectories of chloroplasts were obtained. Analysis of the trajectories of chloroplasts in plants subjected to drug treatments interrupting cytoskeletal functions suggests that in Physcomitrella Patens the microtubule network is primarily responsible for organizing chloroplast motion. Continuing the study of trajectories, under different light intensitites, similar exponents of the mean squared displacement were observed for each condition but there was a relatively large change in the diffusion coefficient. Looking at plots of the coefficients for higher powers, the blue light conditions had nonlinear relations, indicative of rare events or jumps, while the red light condition had a seemingly linear behavior. In the future, modeling and simulations must be performed to explore what mechanisms could give rise to these experimental results.

5 Conclusions

In this project, tools have been developed to study morphology and chloroplast motility in moss. In studying morphology we introduced the box-counting fractal dimension. The fractal dimension had the advantage over metrics currently in use, specifically area and solidity, that it clearly shows day to day increases, potentially enabling it to be used to determine the stage of growth, while providing a qualitative insight into the nature of a plant's shape. Using a simple model we were able to generate plants in silico which resembled Physcomitrella Patens and qualitatively mathed our experimental measurements. Current work is being done to choose an appropriate mapping between the stalk length and the number of days, to make the model more accurate.

To study chloroplast motility, a tracking code was written. It was used to track chloroplasts in moss under several drug and light conditions. Looking at the mean squared displacement of the chloroplasts' trajectories we found that control and Latruculin treated cells, having the microtubule network intact, and the actin network removed, chloroplasts moved in a superdiffusive, directed, manner. Upon removal of the microtubule network the motion became diffusive or subdiffusive regardless of whether the actin network was in place. We also observed that intensity of blue light affected the effective diffusion coefficient. Noting the similarity to what one expects in a random walk which allows for pauses, one possibility for this behavior is the existance passive cross-linkers or molecular motors that can be influenced by light intensity. To gain further insight, higher moments were considered. The higher moments suggest that the trajectories of chloroplasts in the presence of blue and red light are characterised by rare jumps whereas the only red light condition has a steadier, more continual mechanism at work.

6 Appendix

A Box counting algorithm code

```
function [area eccentricity perimeter solidity fboxcount] = ...
    fractalbcount(filelist, output)

%For every image in a .tiff stack, specified by a string, or a set of ...
    stacks, specified by a list stored as a cell array,this program ...
    calculates the fractal dimension of each object in an image via the ...
    box counting algorithm. It also stores other metrics, specifically ...
    the area, eccentricity, perimeter, and solidity, which are ...
    calculated by the regionprops function. Results are stored in a .mat ...
    file named using the input "output."

clear all;

%matrix into which each object's fractal dimension will be placed
fboxdata = [];

%Dummy Matrices which will later hold data

  area= zeros(2000,1);
 eccentricity= zeros(2000,1);
  solidity = zeros(2000,1);
  perimeter= zeros(2000,1);
  count = 1;
  %If a single file is given as a string, perform the box count for that
  %file.
if isstr(filelist)
```

```
        [area, eccentricity, perimeter, solidity, fboxdata, count] = ...
            boxcount(filename,area,eccentricity, ...
            perimeter,solidity,fboxdata, count);
else
  %If multiple files are given as a cell array, perform the box count on
  %each
  if iscell(filelist)
 for k=  1:size(filelist,1);
    %  number of images in the stacks
    filename = filelist{k};
    [area, eccentricity, perimeter, solidity, fboxdata, count] = ...
        boxcount(filename,area,eccentricity, ...
        perimeter,solidity,fboxdata, count);
 end
else
      disp(sprintf('%s is an incompatible file type.',filelist))
      end
end



 % Removes excess zero entries in the matrices containing data
area=area(1:(count−1),1);
eccentricity= eccentricity(1:(count−1),1);
solidity= solidity(1:(count−1),1);
perimeter= perimeter(1:(count−1),1);


%creates histograms
subplot(2,3,1)
hist(area,20)
xlabel( 'Area (pixels)', 'fontsize', 8 ); ylabel( '# occurances', ...
    'fontsize', 8 );
```

```
title( {'Area'; sprintf( 'Mean = %.3g \\pm %.3g', mean(area), (std( ...
    area )/(size(area,1)^(.5))) )});

subplot(2,3,2)
hist(perimeter,9)
xlabel( 'Perimeter (pixels)', 'fontsize', 8 ); ylabel( '# occurances', ...
    'fontsize', 8 );
title( {'Perimeter'; sprintf( 'Mean = %.3g \\pm %.3g', mean(perimeter), ...
    std( perimeter )/(size(perimeter,1)^(.5)) )});

subplot(2,3,3)
hist(eccentricity,5)
xlabel( 'Eccentricity', 'fontsize', 8 ); ylabel( '# occurances', ...
    'fontsize', 8 );
title( {'Eccentricity'; sprintf( 'Mean = %.3g \\pm %.3g', ...
    mean(eccentricity), std( eccentricity )/(size(eccentricity,1)^(.5)) )});

subplot(2,3,4)
hist(solidity,9)
xlabel( 'Solidity', 'fontsize', 8 ); ylabel( '# occurances', ...
    'fontsize', 8 );
title( {'Solidity'; sprintf( 'Mean = %.3g \\pm %.3g', mean(solidity), ...
    std( solidity )/(size(solidity,1)^(.5)) )});

subplot(2,3,5)
hist(fboxdata(:,2));
xlabel('Fractal Dimension', 'fontsize', 8); ylabel('#occurances', ...
    'fontsize', 8);
title ({'Fractal Dimension';sprintf( 'Mean = %.3g \\pm %.3g', ...
    mean(fboxdata(:,2)), std(fboxdata(:,2))/(size(fboxdata(:,2),1))^(.5))});

save(output)
end
```

```

function [area, eccentricity, perimeter, solidity, fboxdata, count] = ...
    boxcount(filename,area,eccentricity, perimeter,solidity,fboxdata, count)
numimages = size(imfinfo(filename),1);

%Identifies every object in a .tiff stack and finds the fractal ...
    dimension of each

for i=1:N1

%imports an image

a=imread(filename, 'Index',i);

% converts the image to a double class image

a= im2double(a);

%An alpha-trimmed mean filter is applied.  This defines a region around a
%pixel.  The average of the interquartile range of the region is ...
    defined as
%the new value of the pixel.

f=imfilter(a,ones(3,3),'symmetric');

for n=1:8;
    f= imsubtract(f, ordfilt2(a,n,ones(3,3)));
end

for n = 4:9;
    f=imsubtract(f,ordfilt2(a,n,ones(3,3)));
end

```

```
f= f/-1;

% The filtered image is converted to a binary image, I.

I=im2bw(f,graythresh(f));

%finding the dimensions of I for later use

[sizex sizey] = size(I);

% All the objects in the image are labeled.
  [L,nn] = bwlabel(I);
  final_selected=zeros(sizex,sizey);

% Various stats for each object in the image are found.
  allstats = regionprops(I,'Area','Centroid','Perimeter',...
'Solidity','Image','PixelList','Eccentricity');

  areamin=1000;


for i=1:nn
    %Selecting objects with an area greater than the specified minimum
    %which do not touch the borders of the image
    if (allstats(i).Area>areamin)
        col=allstats(i).PixelList(1,1);
        row=allstats(i).PixelList(1,2);
        final_selected=bwselect(L,col,row,4);
        if sum(final_selected(:,1)) == 0
            if sum(final_selected(1,:))==0
                if sum(final_selected(sizex,:))==0
                    if sum(final_selected(:,sizey)) ==0

```

```
                        % recording the area, eccentricity, solidity, and ...
                            perimeter of images fitting the criteria

                            area(count,1)= allstats(i).Area;
                            eccentricity(count,1)=allstats(i).Eccentricity;
                            solidity(count,1)=allstats(i).Solidity;
                            perimeter(count,1)=allstats(i).Perimeter;
                            count = count+1;

% Matrix in which to store the box size and the number of boxes needed to
% cover the image
[numbercount] = [;];


 %box sizes desired
 for n= [1,2,4,8,16,32,48,64];

iterx2=1;
itery2=1;
count2=0;

%finding number of square boxes which fit in the x and y
%directions

nx= floor(sizex/n);
ny= floor(sizey/n);

%remaining columns and rows after square box region

remainx = sizex-nx*n;
remainy = sizey-ny*n;

%borders of area for square boxes
```

```

borderx= nx*n;
bordery= ny*n;

%Find the number of boxes needed to cover a region right of the one ...
    with square boxes.  These boxes will have a decreased number of columns.

if remainx≠0
    a = borderx:sizex;
    b = 1:n;
    for itery2= 1:ny
        if sum(sum((final_selected(a,b))))≠0
            count2=count2+1;
            b= b+n*ones(1,n);
            itery2=itery2+1;
        else b= b+n*ones(1,n);
            itery2=itery2+1;
        end
    end
end

%Find the number of boxes needed to cover a region below the one with ...
    square boxes.  These boxes will have a decreased number of rows.


if remainy≠0
    a= 1:n;
    b= bordery:sizey;
    for iterx2=2:nx
        if any(find(final_selected(a,b)))
            count2 = count2+1;
            a= a+n*ones(1,n);
            iterx2= iterx2+1;
        else
            a = a+n*ones(1,n);
            Box = final_selected(a,b);
            iterx2 = iterx2+1;
        end
    end
end

%Finds if the image foreground is present in a corner region at bottom
%right.

if remainy≠0
    if remainx≠0
        a= borderx:sizex;
        b= bordery:sizey;
        if sum(sum((final_selected(a,b))))≠0
            count2 = count2+1;
        end
    end
end

iterx2=1;
itery2=1;
a=1:n;
b=1:n;
%Finds the number of boxes needed to cover the region possessing square
%boxes.
for itery2 =1:ny
    for iterx2=1:nx
        if sum(sum((final_selected(a,b))))≠0
            count2 = count2+1;
            a= a+n*ones(1,n);
            iterx2= iterx2+1;
        else a = a+n*ones(1,n);
            iterx2 = iterx2+1;
        end
    end

   % if iterx>nx
        a=1:n;
        b=b+n*ones(1,n);
        iterx2=1;
        itery2=itery2+1;
   % end
end
%adds the boxsize (n) and the count2 associated with it to a dummy matrix
[numbercount]=vertcat(numbercount,[n,count2]);

 end

%fits a first order polynomial to the loglog plot data
 [fitting]= [polyfit(log(numbercount(:,1)),log(numbercount(:,2)),1)];
%Adds the fractal dimension figure to a matrix
fboxdata = vertcat(fboxdata, [numbercount(1,2) -fitting(1)]);
                    end
                end
            end
        end

    end
end
end
end
```

B Growth model code

```
% plantgrowth(numruns) impliments the final version of the plant growth ...
    algorith described in text.
% Note that all previous growth algorithms can be obtained as special ...
    cases of this code by
% commenting out appropriate plotting lines.
function plantgrowth(numruns)
for runnum = 1:numruns
meanbranchang = 71.76*pi/180;
stdbranchang= 16.16*pi/180;
meancurve = 24.219747*pi/180;
stdcurve = 35.47917*pi/180;
radius = 13; %ballpark estimate from 20 measurements
bulboffset= 11;
bulbcutoffs = 1-[1/46 4/46 15/46 35/46 41/46 1]; %P(number of bulbs)
branchcutoffs = 1-[76/153 138/153 1]; %P(number of branches)
%        lengthcutoffs = 1-[2/115 57/115 90/115 107/115 1];
       toremovecutoffs = 1-[24/45 38/45 44/45 1];% P(number of active ...
           bulbs)
plant.init = 10^8*[1 1 1 1];
        clc
%     disp(sprintf('run: %d/10000',runs))

   gen1=[];


bulbrand = rand(1);
numbulbs= (find(bulbrand>bulbcutoffs,1));
active = randperm(numbulbs);
removerand = rand(1);
```

```
toremove = find(toremovecutoffs>removerand,1);

if ¬isempty(toremove)
if length(active)≥toremove
    active = active(1:(end−toremove));
else
    active = [];
end
end

for bulb = 1:numbulbs

    if length(intersect(bulb,active))>0
        branchrand = rand(1);
        numbranches = find(branchcutoffs>branchrand,1);
        for branches = 1:numbranches
            gen1 = [gen1; bulboffset*(bulb−1) 0 2*pi*rand(1) 1];
        end
    end
end
if size(gen1,1)>0
index = 0;
startpoints = [];
[plant index startpoints] = makebranches(gen1,plant,index,startpoints);

for gen = 2:10
    fld = fields(plant);
    starter = [];
    for i = 1:length(fld)
        dat = plant.(fld{i});
        if ¬isempty(dat(dat(:,4)==(4+(gen−2)),:))&...
           ¬isempty(dat(dat(:,4)==(4+(gen−2)−3),:))

```

```
            starter = [starter;[dat(dat(:,4)==(4+(gen-2)-3),:)]];
        end
    end
    starter(:,4)=starter(:,4)+3;
    [plant index ...
        startpoints]=makebranches(starter,plant,index,startpoints);
end
end

plantplotter(plant,startpoints,numbulbs,runnum)
end
end

function plantplotter(plant,startpoints,numbulbs,runnum)

for i = 1:20
    mkdir(sprintf('step%d',i))
end

figure(1)
            clf
    set(gcf,'Position',[0,0,900,900])
    bulbs = [];
for i = 1:numbulbs
    bulbs = [bulbs;[11*(i-1) 0]];
        end

        axis off
for i = 1:20
    touse = startpoints(startpoints(:,2)<=i,1);
    figure(1)
    clf
    scatter(bulbs(:,1),bulbs(:,2),12^2,'filled')
```

```
        axis([-700 700 -700 700])
        axis off
%      for i=1:numbulbs
%          scatter(11*(i-1),0,(14+1.3*2*((rand(1)<.5)-.5))^2,'filled')
%      end

    for n = 1:length(touse)
        name = sprintf('branch%d',n);
        dat = plant.(name)(:,[1,2,4]);
        dat = dat(dat(:,3)<=i+1,:);
        figure(1)
        hold on
        plot(dat(:,1),dat(:,2),'LineWidth',7.5)

    end
    saveas(gcf,sprintf('step%d\\runnum_%d_step_%d.png',i,runnum,i))
end
end

function plantplotter(plant,startpoints,numbulbs,runnum)

for i = 1:20
    mkdir(sprintf('step%d',i))
end

figure(1)
            clf
    set(gcf,'Position',[0,0,900,900])
    bulbs = [];
for i = 1:numbulbs
    bulbs = [bulbs;[11*(i-1) 0]];
        end

        axis off
for i = 1:20
    touse = startpoints(startpoints(:,2)≤i,1);
    figure(1)
    clf
    scatter(bulbs(:,1),bulbs(:,2),12^2,'filled')
        axis([−700 700 −700 700])
        axis off
%      for i=1:numbulbs
%          scatter(11*(i−1),0,(14+1.3*2*((rand(1)<.5)−.5))^2,'filled')
%      end

    for n = 1:length(touse)
        name = sprintf('branch%d',n);
        dat = plant.(name)(:,[1,2,4]);
        dat = dat(dat(:,3)≤i+1,:);
        figure(1)
        hold on
        plot(dat(:,1),dat(:,2),'LineWidth',7.5)

    end
    saveas(gcf,sprintf('step%d\\runnum_%d_step_%d.png',i,runnum,i))
end
end
```

C Tracking algorithm code

```matlab
%track(filename) identifies and tracks objects across frames of a .tiff
%file.
function track(filename)
try
currentused = [];
minarea = 40;
    maxarea = 1500;
if exist('chloroplastnumber')==1
    start = chloroplastnumber;
else
    chloroplastnumber = 1;
    start = chloroplastnumber;
end
numimages = size(imfinfo(filename),1); %determine the number of frames

for p=1:numimages
    %apply alpha trimmed mean filter
    I=alphfilt(filename,p);
     [sizex sizey] = size(I);
    %All objects in the image are located, labeled, and the area,
    %centroid ,and list of pixels composing each image are found

    [L,nn] = bwlabel(I,4);
    allstats = regionprops(I,'Area','Centroid', 'Orientation', ...
        'Eccentricity', 'PixelList');
    showimage = zeros(sizex,sizey);

    if p==1 %if it is the first frame
```

```
        for i = 1:length(allstats)%For all objects in the image between ...
            the specified area constraints
            if  minarea ≤ allstats(i).Area&  allstats(i).Area ≤ maxarea
                col=allstats(i).PixelList(1,1);
                row=allstats(i).PixelList(1,2);
                final_selected=bwselect(L,col,row,4);
                                %Check to make sure they not touching ...
                                    the borders
                if sum(final_selected(:,4)) == 0 %top edge
                    if sum(final_selected(4,:))==0 %left edge
                        if sum(final_selected(sizex-4,:))==0 %bottom edge
                            if sum(final_selected(:,sizey-4)) ==0 ...
                                %right edge
                                %If area and border checks are passed, a
                                %data set is created for the chloroplast.
                                %This data set records the centroid of the
                                %chloroplast (x,y), its area
                                %(pixels^2),orientation, and eccentricity.
                                name = sprintf('chloroplast%d',...
                                               chloroplastnumber);
                                chloroplastdata.(name)= ...
                                    [allstats(i).Centroid, ...
                                    allstats(i).Area, p, ...
                                    allstats(i).Orientation, ...
                                    allstats(i).Eccentricity] ;
                                %A log of all chloroplasts used in the
                                %first frame is kept
                                currentused = vertcat(currentused, ...
                                    chloroplastnumber);
                                chloroplastnumber= chloroplastnumber+1;
                                showimage = ...
                                    bwselect(I,allstats(i).PixelList(:,2),...
                            allstats(i).PixelList(:,1))+showimage;
```

```
                              end
                          end
                      end
                  end
              end
          end
      end
      if p ≠1 %if it is not the first frame
         for i = 1:length(allstats) %loop over every found object
                              if minarea≤allstats(i).Area...
                                &allstats(i).Area≤maxarea
            %check to make sure it is between the area constraints
             newchlor = allstats(i).PixelList;
             if length(newchlor(newchlor(:,1)≤4&...
              newchlor(:,2)≤4&...
              newchlor(:,1)≥(sizex−4)&...
              newchlor(:,2)≥(sizey−4),:))==0 %border checks
                     intsct = [];
                     for ii = 1:length(usedprevframe) %Check to see if ...
                         the pixels of the current chloroplast intersect ...
                         any of the centroids of the chloroplasts from the ...
                         previous frame.
                         name = sprintf('chloroplast%d',usedprevframe(ii));
                         if length(intersect(allstats(i).PixelList,...
                          floor(chloroplastdata.(name)(end,[1,2])),'rows'))>0

                             oldarea = chloroplastdata.(name)(end,3);

                             if .8*oldarea≤allstats(i).Area≤1.2*oldarea
                             intsct = [intsct;usedprevframe(ii)];
                            %makes a list of all intersecting chloroplasts
                         end
                         end
            end

            if length(intsct)≤1 % A chloroplast intersecting 2 ...
               or more previous centroid locations indicates a ...
               merger or overlap.  It is not recorded as we are ...
               studying the dynamics of individual particles, ...
               not pairs or clumps.
            if length(intsct)==0 %If the chloroplast doesn't ...
               intersect with any centroids of the previous ...
               frame, it is assumed that the chloroplasts is a ...
               newly identified one and an entry in the struct ...
               (chloroplastdata) is made for it.
                name = sprintf('chloroplast%d',chloroplastnumber);
                currentused = [currentused;chloroplastnumber];
                chloroplastnumber = chloroplastnumber+1;
                chloroplastdata.(name)= [allstats(i).Centroid, ...
                   allstats(i).Area,p,allstats(i).Orientation, ...
                   allstats(i).Eccentricity];
            else
                if length(intsct)==1 %If the chloroplast ...
                   intersects only one centroid, it is assumed ...
                   that the chloroplast the one that was found ...
                   in the previous frame at a later time.  The ...
                   current centroid location is appended to its ...
                   data struct.
                    name = sprintf('chloroplast%d',intsct);
                    chloroplastdata.(name)=[chloroplastdata.(name);[[allstats(i).
                       allstats(i).Area,p,allstats(i).Orientation, ...
                       allstats(i).Eccentricity]]];
                      currentused = [currentused;intsct];
                end
            end
            end
```

```
                end
            end
        end
    end
            usedprevframe = currentused;
        currentused = [];
%      imname2= sprintf('framenum%d',p);
%      imagedata.(imname2)=showimage;
    clc
    disp(sprintf('file: %s',filename))
    disp(sprintf('frame = %d/%d', p,numimages))
end
%Data is saved in miscellaneous formats (.mat,.txt,.xls)
a = fields(chloroplastdata);
for i = 1:length(fields(chloroplastdata))
if size(chloroplastdata.(a{i}),1)<5
    chloroplastdata = rmfield(chloroplastdata,a{i});
end
end
a= fields(chloroplastdata);
num = 1;
for i = 1:length(a)
    name = sprintf('chloroplast%d',num);
    chloroplastdata2.(name)=chloroplastdata.(a{i});
    num = num+1;
end
chloroplastdata = chloroplastdata2;
if exist('..\Tracking Results','dir')≠7
     mkdir('..\Tracking Results')
 end
savedata= [];
for i = 1:length(fields(chloroplastdata))
    name = sprintf('chloroplast%d',i);
    savedata = [savedata;[ones(size(chloroplastdata.(name),1),1)*i ...
        chloroplastdata.(name)]];
end
[dum1 fname dum2]= fileparts(filename);
saveloc = sprintf('..\\Tracking Results3\\%s',fname);
mkdir(saveloc)
save(sprintf('%s\\%s_tracking_results_struct.mat',saveloc,fname),...
     'chloroplastdata')
save(sprintf('%s\\%s_tracking_results.mat',saveloc,fname),...
     'savedata')
save(sprintf('%s\\%s_tracking_results.txt',saveloc,fname),...
     'savedata','-ascii')
save(sprintf('%s\\%s_tracking_images.mat',saveloc,fname),...
     'imagedata')
catch
   dlmwrite('..\Files_with_errors.txt',filename,...
     '-append','newline','pc','delimiter',''); %If error occur ...
         (typically because no trackable objects were found), a txt is ...
         written which records all files which had errors.  These are to ...
         be checked manually afterward.  Based on manual investigation, ...
         one may wish to rerun the tracking code with looser (or ...
         stricter) area constraints.
end
end
```

D Auxiliary codes

D.1 getAllFiles.m

```matlab
% X= getAllFiles(folder)
% returns a cell array listing all files in a specified directory (folder).
%
% X = getAllFiles(folder,'str1','str2',...)
% returns a cell array listing all files in a specified directory (folder)
% whose path contains all specified strings ('str1','str2',etc.).

function filelist = getAllFiles(folder,varargin)
dirdat = dir(folder);
ind = [dirdat.isdir];

%adds in the files from this folder
filelist = {dirdat(¬ind).name}';

% find things matching input strings

if nargin>1

    for k = 1:length(varargin)
    if ¬isempty(filelist)
            tomatch = varargin{k};
    good = [];
    for i = 1:size(filelist,1)
        if ¬isempty(strfind(filelist{i},tomatch))==1
            good = [good;i];
        end
    end
    filelist = filelist(good);
    end
    end
end

%if there is anything left that matches the input search terms, if any,
%tack on the entire directory structure to them

if ¬isempty(filelist)
    filelist = cellfun(@(x) ...
        fullfile(folder,x),filelist,'UniformOutput',false);
end
%Gets a list of the folders which are not above the current working ...
    folder ('.','..')

sub = {dirdat(ind).name};
notabove = ¬ismember(sub,{'.','..'});

%for all subfolders, run the code again and tack on the found files

for n = find(notabove)
    dummy = fullfile(folder,sub{n});
    filelist = [filelist; getAllFiles(dummy,varargin{:})];
end

end
```

D.2 cell2txt

```
% cell2txt(filelist,output)
% takes a cell array (filelist)and outputs the contents of each cell in a
% .txt named (output).txt.  Note:  This was written to compliment the
% getAllFiles code which returns a single string per cell in a cell array.
% It will break under other circumstances (ex. using it on cells containing
% structs).

function cell2txt(filelist,output)
savename = sprintf('%s.txt',output);
for i = 1:length(filelist)
    dlmwrite(savename,filelist{i},'delimiter','','-append','newline','pc');
end
end
```

D.3 track_txt

```matlab
%track_txt(txtname) calls the track function for every .tiff specified ...
    in a
%txt.

function track_txt(txtname)
% addpath(genpath(fileparts(pwd))) %This line is used to slightly speed ...
    up the track
% % code if the files of interest are in a subdirectory of the parent
% % directory of where this code is located.  It adds all files under ...
    the parent directory
% % to Matlab's path. In typicaly circumstances, leave this commented.
a = textread(txtname,'%s');
for i = 1:length(a)
    track(a{i})
end
end
```

D.4 fractalbcount_txt

```
%fractalbcount_txt(txtname) calls the fractalbcount function for every ...
    .tiff specified in a
%txt.
function fractalbcount_txt(txtname)
a = textread(txtname,'%s');
for i = 1:length(a)
    fractalbcount_txt(a{i})
end
end
```

References

[1] Vidali, L., G. Burkart, et al. (2010). ”Myosin XI Is Essential for Tip Growth in Physcomitrella patens.” *Plant Cell* 22(6): 1868-1882.

[2] Fabienne Furt, Kyle Lemoi, Erkan Tüzel and Luis Vidali, ”Quantitative analysis of organelle distribution and dynamics in Physcomitrella patens protonemal cells.” *BMC Plant Biology* 2012, 12:70

[3] Vidali, L., van Gisbergen, P.A., Guerin, C., Franco, P., Li, M., Burkart, G.M., Augustine, R.C., Blanchoin, L., and Bezanilla, M. (2009) Rapid formin-mediated actin-filament elongation is essential for polarized plant cell growth. *Proc. Natl. Acad. Sci. USA* 106:13341-13346

[4] Vidali, L., Augustine, R.C., Kleinman, K.P., and Bezanilla, M. (2007) Profilin is essential for tip growth in the moss Physcomitrella patens. *Plant Cell* 19:3705-3722

[5] Vidali, L., Burkart, G., Augustine, R.C., Kerdavid, E., Tuzel, E., and Bezanilla, M. (2010) Myosin XI is essential for tip growth. *Plant Cell* 22:1868-1882.

[6] Vidali, L., McKenna, S.T. and Hepler, P.K. (2001) Actin polymerization is essential for pollen tube growth. *Mol. Biol. Cell* 12: 2534-2545

[7] Prusinkiewicz, Przemyslaw, and Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. New York: Springer-Verlag, 1990. Print.

[8] Fracchia, F. D., and N. W. Ashton. ”A Visualization Tool for Studying the Development of the Moss Physcomitrella Patens.” *IEEE Conference on Visualization, Proceedings.* IEEE Conference on Visualization., 1995. 364-67. Print

[9] Phillips, Rob, Jane Kondev, and Julie Theriot. *Physical Biology of the Cell*. New York: Garland Science, 2009.

[10] Mandelbrot, B. ”How Long Is the Coast of Britain? Statistical Self-Similarity and Fractional Dimension.” *Science* 156.3775 (1967): 636-38. Print.

[11] Wu, Jianron, and Keith M. Berland. ”Propagators and Time-Dependent Diffusion Coefficients for Anamalous Diffusion.” *Biophysical Journal* 95.4 (2008) 2049-2052