Jet Propulsion Laboratory
Machine Learning &
Instrument Autonomy Group

B-0128

To: File
From: M.C. Burl
Date: 2008/08/20
Subject: Toward Automated Analysis of *Arabidopsis* "Live Imaging" Data

1 Introduction

This memo[1] provides a brief summary of various activities conducted by the author[2] in support of the NSF-funded *Computable Plant* project [6]. Much of the work was performed in collaboration with Adrienne Roeder, a postdoctoral scholar working in the Caltech Biology Department in the lab of Professor Elliot Meyerowitz.

The overall goals of the project are: (1) to answer fundamental questions about the role of genetics and environment in the development of complex organisms and (2) to develop accurate, testable computer models that capture the developmental process at both structural (microtubule orientation, division planes, 3D models of nuclei and cell compartments, etc.) and functional levels (gene regulatory networks, environmental signaling, etc.). Much of the input to the process comes from imaging data collected with a confocal laser scanning microscope (CLSM) of fluorescently-tagged *arabidopsis* specimens, including both normal (*wild type*) and mutant variations. A unique aspect of CLSM is that it allows repeated, non-destructive imaging in 3D of live specimens, i.e., *live imaging*.

The project relies on a multidisciplinary approach that combines traditional biology and live imaging with automated image analysis and computer modeling. The main activities pursued by the author and discussed in this memo are as follows:

- **Segmentation of Nuclei in 2D**
- **Studies of Endoreduplication**
- **Growth and Lineage Analysis**
- **Recovering Nuclear Bounding Surfaces in 3D**
- **Segmenting Cell Compartments in 2D**
- **Recovering Cell Compartment Models in 3D**

Additionally, two appendices are included that discuss microscope coordinate systems and the various polyhedral representations that are useful for modeling 3D objects such as nuclei and cell compartments. A software package containing nearly two hundred Matlab functions that implement various algorithms and experiments was delivered to the internal Computable Plant website [3] on 2008/09/03, along with a README document describing the basic capabilities of each function.

[1] This document represents preliminary work performed by the author. The content does not reflect the official policy or position of the Jet Propulsion Laboratory or the United States Government. Secondary distribution, disclosure, or dissemination of this document without the express permission of the author is prohibited.

[2] Total effort expended was approximately 0.4 FTEs (2 years $\times \sim 20\%$/yr).

2 Terminology and Conventions

CLSM imaging produces an image stack consisting of a set of slice images of equal size taken at various depths (z-values) relative to the microscope. The slices are "virtual" in the sense that they are accomplished with an optics trick that requires no (or little) physical harm to the specimen. Fluorescent tags and/or dyes that are sensitive to particular wavelengths of light are typically introduced into the specimens prior to imaging to highlight different structures. For example, a nuclear-localized cyan fluorescent protein (CFP) may be introduced to highlight nuclei, while a lipophilic fluorescent dye, such as FM-64, may be introduced to highlight plasma membranes. We refer to the images that are taken at different wavelengths as *channels*.

Image stacks, which are 3D, can also be obtained at different times over the course of development (although these time snapshots cannot be spaced too close together due to some trauma caused to the specimen by the laser light, photo-bleaching, etc.). The raw output from a CLSM imaging experiment can be written as:

$$\mathbf{I}_c(x, y, z, t) \tag{1}$$

where c represents the wavelength channel, x and y represent the "horizontal" dimensions (within a single stack frame image), z represents the "vertical" dimension (stack level or depth from the microscope), and t represents the time dimension. When there is no danger of confusion, various unused indices will be omitted for clarity. For example, when talking about an image in a particular channel at a particular time, we may write $\mathbf{I}(x, y)$ to simplify the notation.

Usually, it is convenient to think of the various dimensions in Expression 1 as being integer-valued indices; however, it is sometimes necessary to work with the true real-valued units (meters, seconds, etc.). Appendix 1 discusses the various coordinate systems that are relevant to the imaging experiments.

A *maximum intensity projection image* of a stack over the z-dimension is defined as:

$$\mathbf{M}(x, y) \triangleq \max_z \mathbf{I}(x, y, z) \tag{2}$$

It is also possible to associate a depth with each pixel in a maximum intensity projection, e.g., using the z value that provided the maximum intensity value for each (x, y) pixel in the maximum intensity projection,

$$\mathbf{Z}(x, y) = \arg\max_z \mathbf{I}(x, y, z) \tag{3}$$

Segmentation of an image amounts to labeling a particular set of pixels as belonging to a specific object or structure. The result of segmentation is often represented as a *label image*, $\mathbf{L}(x, y)$, where each pixel in $\mathbf{L}$ takes on an integer value representing the ID of the object containing that pixel. A zero value is reserved for pixels belonging to the background. Typically, the pixels belonging to the individual segments (objects) are connected and we can equivalently represent a segmentation result with a set of boundary contours (or bounding surfaces in the 3D case). The conversion from a label image to boundary contours is called contour following and can be accomplished with the function `icluster_contours.m`, while the reverse operation of converting a set of boundary contours to a label image can be accomplished by a polygon filling operation, `polygon_mask.m`, which creates a binary image, followed by `icluster.m`. The regions enclosed by the various boundary contours must be disjoint for this operation to work properly.

3 Nucleus Segmentation in 2D

Reliable 2D segmentation of nuclei in CLSM imagery of fluorescently-tagged *arabidopsis* samples is important for a number of downstream applications, including studies of endoreduplication (Section 4), growth and lineage (Section 5), and extracting polyhedral models that represent the nuclear bounding surfaces in 3D (Section 6). Two distinct approaches to nucleus segmentation were pursued at different times and for different purposes over the course of the project. One approach is based on edge detection and finding closed contours; the other approach uses thresholding and morphology.

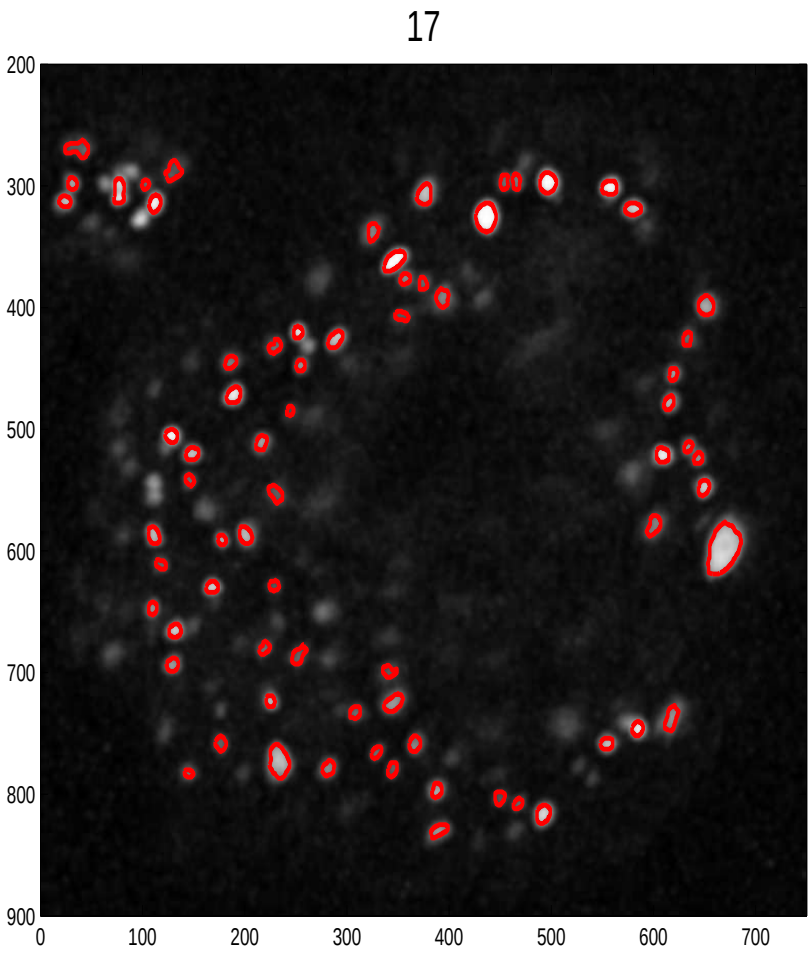

Figure 1: The grayscale image is the nucleus channel image from a particular stack level (17). Nucleus boundary contours at this stack level, as determined by the edge-based nucleus segmentation method, are overlaid in red.

3.1 Edge-Based Nucleus Segmentation

We initially pursued an approach to nucleus segmentation based on edge detection. This technique was targeted for finding nuclei in each individual image frame in an image stack. Each frame I, taken from the appropriate fluorescence channel containing the nuclei, is first smoothed in both the horizontal (x-y) and the vertical (z) directions via convolution with a compact Gaussian-like kernel. Edge contours are then extracted based on the spatial gradient of the smoothed frame using a procedure similar to the Canny edge detector [5]. Detected contours are evaluated to see whether they meet length and closure tests[3]. Any contour meeting both the test criteria is declared to be a bounding contour of a nucleus. Figure 1 shows a set of nuclear contours detected with this method on a particular frame from an image stack. The function `nucleus_stack.m` in the software delivery uses this approach to convert a stack of images into a stack of contours.

On the positive side, the edge detection approach is straightforward, very fast, and provides some robustness to background levels that may vary across the image. On the negative side, small quirks in the boundary of a given nucleus may result in complete rejection of the nucleus (e.g., if the edge detector wanders into the interior of the nucleus due to some irregularity in its boundary and texture in its interior, the nucleus will likely be rejected by the closure test). Also, nuclei that appear to overlap (for example, this happens

[3]The closure test merely compares the Euclidean distance between the start point and end point of a contour against a distance threshold.

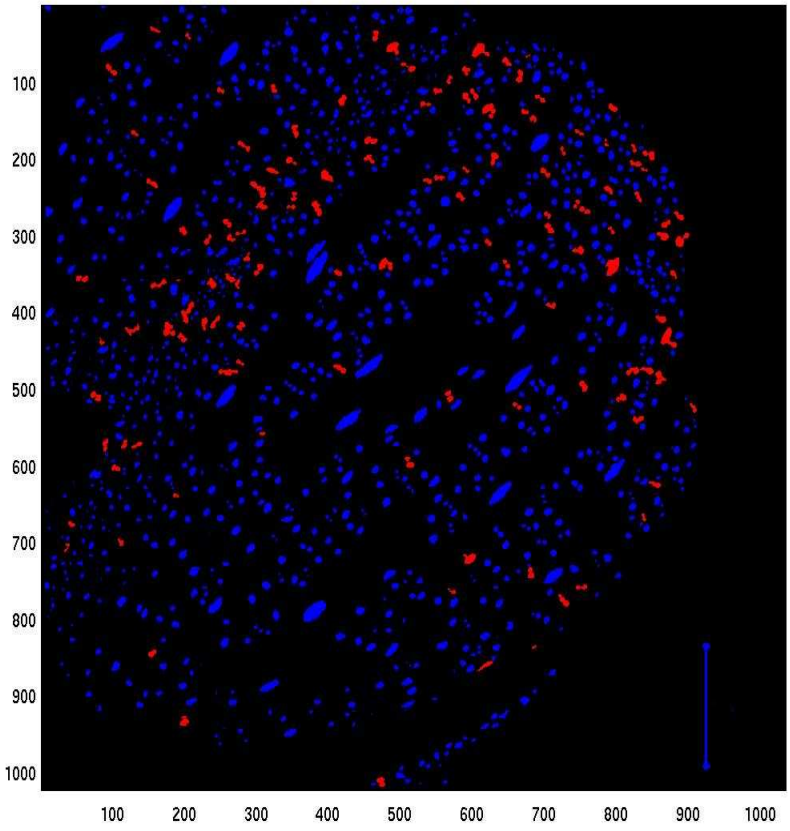

Figure 2: Initial segmentation of a nucleus channel image into blobs. The blobs shown in red were identified as needing further splitting based either on the presence of multiple local maxima within the region or deviation of the boundary contour from elliptical.

frequently in maximum intensity projections) will be lumped together within a single contour. Another drawback is that the smoothing needed to help insure that edge detection is reliable may slightly distort the size and shape of the nuclei (expanding the effective point spread function (PSF) of the microscope).

3.2 Thresholding and Morphology-Based Nucleus Segmentation

Later in the project, a thresholding and morphology-based approach to nucleus segmentation was developed. This approach was targeted toward segmentation of nuclei in projection images (usually of the plant sepal), but likely can provide reliable results in the stack frame case. With this approach, a (3×3) median filter is applied to clean up the incoming nucleus channel image. The output from the median filter is thresholded[4], and any pixels above the threshold are clustered into blobs using a generalized connected components algorithm (`icluster.m`). This initial segmentation into blobs is represented by a *label image* as discussed in Section 2. Several tests are applied to determine whether a particular blob needs to be split further. For example, we evaluate the number of local maxima from the median filtered image that fall into each blob. We also consider the boundary of the blob and its departure from elliptical. Blobs that need splitting are then split using Matlab's built-in watershed function. Figure 2 show a set of detected blobs. Those shown in red are identified as needing splitting.

[4]We have only used a single global threshold across the image, but adaptive thresholding could be investigated.

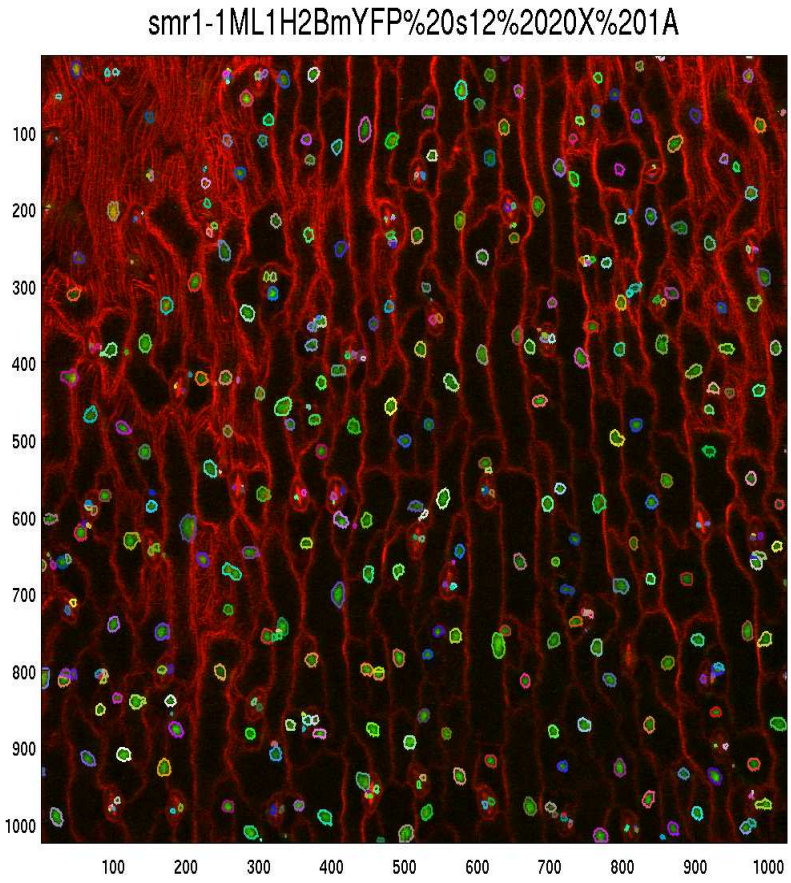

Figure 3: The boundary contours of various nuclei detected in the sepal of a *smr1* mutant using the thresholding and morphology approach are shown overlaid in random colors on the membrane channel (red) and nucleus channel (green).

No blobs are currently rejected based on size, but this can be easily accomplished in a post-processing step (e.g., using `icluster_moments.m`, `icluster_ellipse_params.m`, and `icluster_elim`). The overall thresholding and morphology based segmentation function is called `sepal_nuclei.m`, although it is not specifically limited to sepal nuclei. Figure 3 shows the segmentation of various nuclei detected in the sepal of a *smr1* mutant using randomly-colored boundary contours overlaid on the combined membrane and nucleus channels.

3.3 Refining Segmentation Results

Although both segmentation algorithms generally work well, the results are never perfect. Depending upon the intended use for the segmentation results, it may be desirable to apply automatic and/or manually editting. As mentioned above, rejection based on size and/or shape constraints can be accomplished with the `icluster_*.m` tools. An interactive segmentation editor is also available as `segmentation_editor.m`. Basically, this function takes as input the raw image, the label image produced by the automatic segmentation algorithm, the colors associated with each blob, and the set of boundary contours for the blobs. Through mouse clicks, blobs can be added or deleted. At this time, there is a constraint that blobs cannot be overlapping. Refer to the function documentation for more details on the user interface. The end result is a new label image, color set, and contour set, which represents the revised segmentation results. Figure 4 shows the difference between a rev0 (raw segmentation) product and a rev1 (edited segmentation) product.

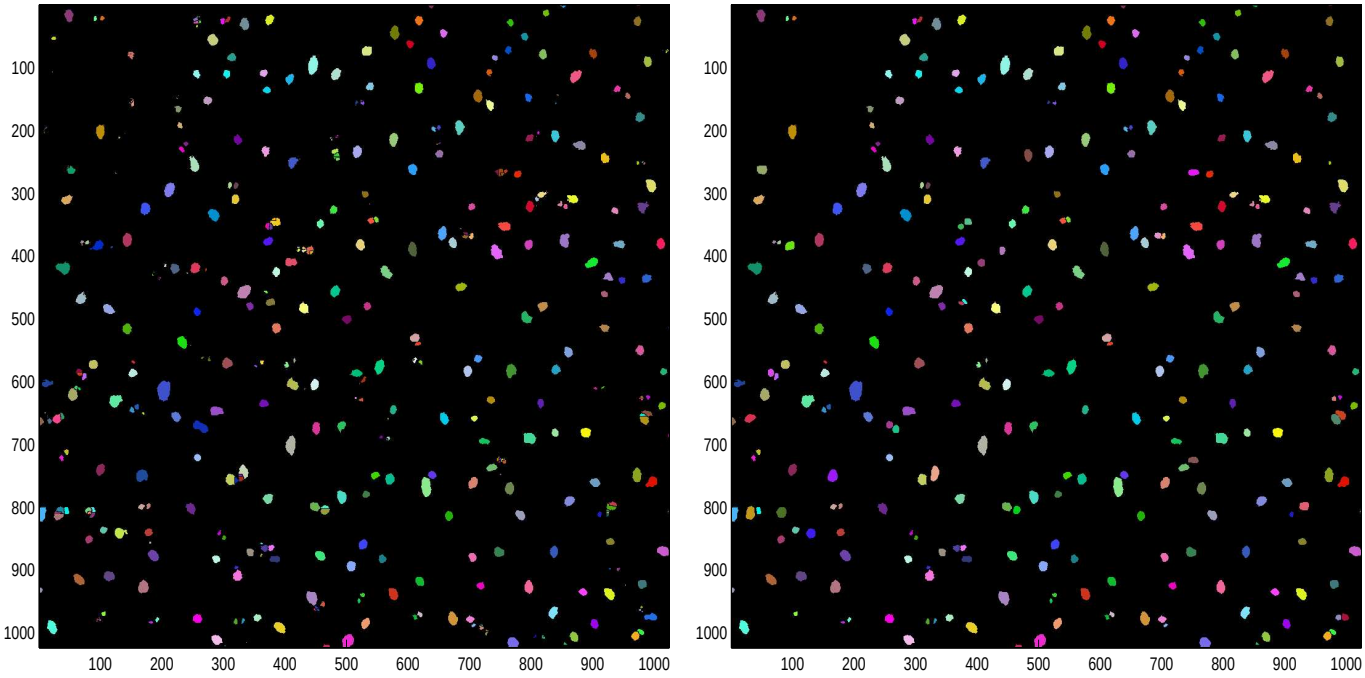

Figure 4: (a) Raw nucleus segmentation (rev0) for an smr1 image. (b) Corresponding nucleus segmentation after manual revision (rev1).

Note that the edge-based segmentation approach of Section 3.1 does not directly produce a label image and hence cannot be immediately used with the segmentation editor. However, we can apply the extra step discussed toward the end of Section 2 to convert the boundary contours into a label image. It is then necessary to re-extract the contours from the label image to insure that the ID numbers associated with the contours match the ID numbers used in the label image.

4 Endoreduplication

Several studies were conducted to gain insight into the mechanism of endoreduplication that is hypothesized to give rise to the giant cells that appear in the sepals of normal (*wild type*) *arabidopsis* plants. In a typical cell cycle, a cell replicates its DNA, then divides with half of the (now doubled) DNA going to each daughter cell. In endoreduplication, the cell replicates its DNA, but fails to divide; once a cell enters endoreduplication, it does not divide in future cell cycles leading to an increase in its DNA content at each step that presumably goes in powers of two.

One investigation considered the use of nuclear volume as a proxy for DNA content. (We also investigated the use of integrated intensity after removing the base background level as a different measure for DNA content.) Generally with the sepal we work with 2D projection images so it is not feasible to recover a true nucleus volume. Instead we construct a pseudo-volume by fitting a 2D ellipse to each segmented 2D nucleus. The principal axes of the 2D ellipse, $\lambda_{\min}$ and $\lambda_{\max}$, are used to define the pseudo-volume as follows:

$$\tilde{V} \quad = \quad \frac{4\pi}{3} \cdot \lambda_{\max} \cdot \lambda_{\min} \cdot \lambda_{\min} \tag{4}$$

This calculation basically assumes that the long dimension of the nucleus lies in a plane parallel to the imaging plane and that the nucleus is an ellipsoid of revolution about its long axis.

The nucleus segmentation procedure from the previous section (the thresholding and morphology based method) was applied to roughly a dozen sepal projection images from *wild type*, as well as to a similar number of images from the mutants *smr1* and *E10_19*. The raw segmentations (rev0 products) were refined through manual editing to produce rev1 versions of the segmentations. From the cleaned segmentations,

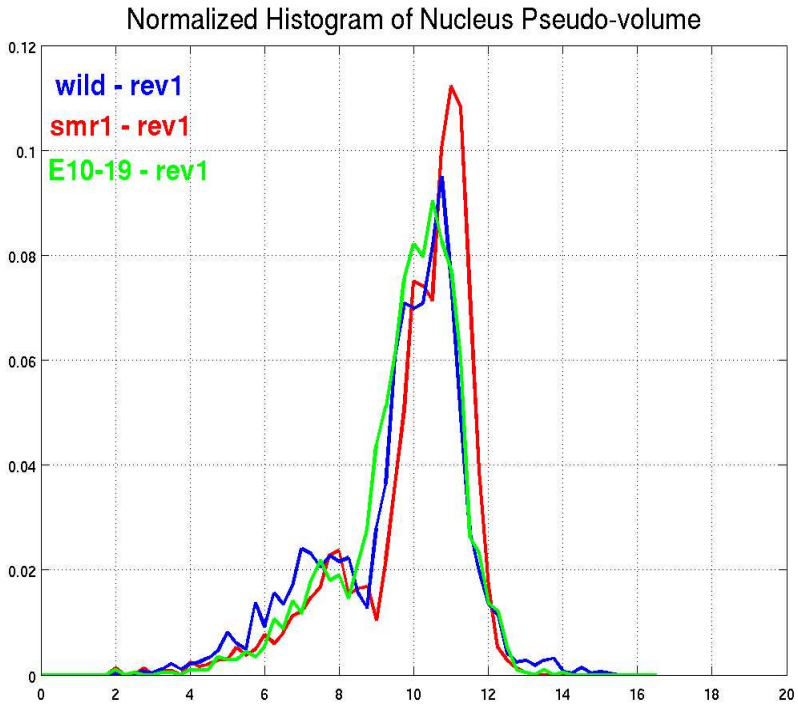

Figure 5: Histograms of nucleus pseudo-volumes over sepal for wild type, smr1, and E10-19 variants. The stronger upper tail for wild type (blue) seems to confirm the presence of more giant cells.

we extracted ellipse parameters and calculated pseudo-volumes for each detected nucleus. A normalized histogram showing the resulting distribution of pseudo-volumes for each of the three variants is shown in Figure 5 using a $\log_2$ scale on the x-axis. The heavier upper tail on the *wild type* (blue) curve along with the appearance of discernible semi-regular peaks in the histograms are noteworthy.

However, some follow-on experiments conducted by A. Roeder using a different direct mechanism[5] to measure histograms of ploidy level (DNA content) seem to indicate that the imaging approach with the particular fluorescent marker that was used does not provide a reliable enough measure of DNA content.

5 Growth and Lineage Analysis

Another pursuit during the project was to estimate growth rates and to follow cell divisions to establish lineage and progeny. The input for these studies consisted of time sequences in which a particular plant was imaged at (typically) eight hour intervals for several days. A. Roeder provided hand tracking over the sequences for a subset of the visible nuclei.

Considerable effort was required to convert the hand-tracking data into machine-usable form. For future work, this step should be better streamlined. More specifically, what is needed is a *tracking table*: a simple text file containing one line for each observation of each nucleus over the duration of the sequence along with the observed position coordinates and a nucleus ID that is persistent over time. The nucleus ID could be in *ancestor label* form or *lineage label* form. Ancestor labels use the ID of the earliest ancestor of a particular cell, while lineage labels keep track of cell divisions. Table 1 gives an example of a tracking table with lineage id labels. The label '2.1' in the **id** column indicates that this cell is the first daughter cell of cell number 2 (where the distinction between first and second daughter is arbitrary).

Two data sequences were examined: one, known as `flower5`, contained 3D nucleus positions, while the other, known as `flower6`, provided only 2D nucleus positions.

[5]Flow cytometry.

t	x	y	z	id
1	10.3	20.4	3.8	1
1	30.8	40.7	6.2	2
1	50.1	10.4	2.8	3
2	12.3	18.5	3.3	1
2	31.2	43.2	5.8	2.1
2	29.7	38.6	5.6	2.2
2	14.0	13.8	3.0	3
2	17.0	60.8	4.2	4
⋮	⋮	⋮	⋮	⋮

Table 1: Example tracking table used as input by the various growth analysis functions. Each observation of each nucleus over the duration of the sequence is recorded as a line in the table. The first column indicates the frame number (equivalently, time) of the observation, the next two (or three) columns indicate the coordinates of the nucleus centroid, and the final column indicates the id label. The id label may be an *ancestor label* or a *lineage label*, as shown here.

5.1 Principal Growth Rates

Given tracking data in the form of Table 1, we can easily determine correspondences between entries at time t and at time $t+1$. Suppose there are N correspondences. Assuming 3D coordinate data, let ${}^{(t)}\mathbf{X}$ and ${}^{(t+1)}\mathbf{X}$ be defined as follows:

$$ {}^{(t)}\mathbf{X} = \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_N & y_N & z_N \end{bmatrix} \tag{5} $$

$$ {}^{(t+1)}\mathbf{X} = \begin{bmatrix} x'_1 & y'_1 & z'_1 \\ x'_2 & y'_2 & z'_2 \\ \vdots & \vdots & \vdots \\ x'_N & y'_N & z'_N \end{bmatrix} \tag{6} $$

where point (x_k, y_k, z_k) in frame t corresponds with the point (x'_k, y'_k, z'_k) in frame $t+1$. For 2D coordinate data, the third (z) column would simply be omitted from each data matrix.

Given the data matrices, we can try to solve for the best affine transformation ${}^{(t+1)}\mathbf{A}_{(t)}$ from time t to time $t+1$:

$$ {}^{(t+1)}\mathbf{X}^T \approx {}^{(t+1)}\mathbf{A}_{(t)} \cdot \begin{bmatrix} {}^{(t)}\mathbf{X}^T \\ \mathbf{1}_N^T \end{bmatrix} \tag{7} $$

where for 3D data $\mathbf{A}$ is (3×4) and for 2D data $\mathbf{A}$ is (2×3). The vector $\mathbf{1}_N$ is an $(N \times 1)$ vector of ones. The best solution for $\mathbf{A}$ in a least squares sense is given by:

$$ {}^{(t+1)}\mathbf{A}_{(t)} = {}^{(t+1)}\mathbf{X}^T \cdot \begin{bmatrix} {}^{(t)}\mathbf{X}^T \\ \mathbf{1}_N^T \end{bmatrix}^{\#} \tag{8} $$

where # represents the pseudo-inverse. Note that $\mathbf{A}$ can be broken down as follows:

$$ \mathbf{A} = \begin{bmatrix} \mathbf{B} & \vdots & \mathbf{t} \end{bmatrix} \tag{9} $$

For the 3D case $\mathbf{B}$ is (3×3) and $\mathbf{t}$ is (3×1), and for the 2D case $\mathbf{B}$ is (2×2) and $\mathbf{t}$ is (2×1). The $\mathbf{B}$ portion represents rotation, scaling, and shearing, while $\mathbf{t}$ represents translation.

Principal growth directions and rates can be obtained by factoring $\mathbf{B}$ using the singular value decomposition (SVD).

$$\mathbf{B} = \mathbf{USV}^T \tag{10}$$

where $\mathbf{U}$ and $\mathbf{V}$ are (3×3) orthonormal matrices and $\mathbf{S}$ is a (3×3) diagonal matrix with non-negative elements in descending order along the diagonal. Consider two points in frame t: $\mathbf{p}_0$ and $\mathbf{p}_1 = \mathbf{p}_0 + \mathbf{v}_1$, where $\mathbf{v}_1$ is the first column of $\mathbf{V}$. Where do these points map in frame $t+1$?

$$\mathbf{p}'_0 = \mathbf{Bp}_0 + \mathbf{t} \tag{11}$$
$$\mathbf{p}'_1 = \mathbf{Bp}_0 + \mathbf{Bv}_1 + \mathbf{t} \tag{12}$$

The displacement vector $\mathbf{d}$ between the two points in frame t is given by:

$$\begin{aligned} \mathbf{d} &= \mathbf{p}_1 - \mathbf{p}_0 \\ &= \mathbf{v}_1 \end{aligned} \tag{13}$$

while the displacement vector in frame $t+1$ is given by:

$$\begin{aligned} \mathbf{d}' &= \mathbf{Bv}_1 \\ &= \mathbf{USV}^T\mathbf{v}_1 \\ &= \mathbf{US}\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \\ &= s_1 \cdot \mathbf{u}_1 \end{aligned} \tag{14}$$

The length of $\mathbf{d}$ is 1 (since $\mathbf{v}_1$ is a unit vector) while the length of $\mathbf{d}'$ is s_1 (since $\mathbf{u}_1$ is also a unit vector). Thus, the distance between $\mathbf{p}_0$ and $\mathbf{p}_1$ is scaled by s_1 as a result of the transformation. It turns out that for any direction (unit length displacement vector) we could have picked in frame t, the vector $\mathbf{v}_1$ is the one that gets scaled by the biggest factor. Thus, $\mathbf{v}_1$ is the principal growth direction and s_1 is the principal growth rate. In 3D, there is a 2D subspace orthogonal to $\mathbf{v}_1$ and we can ask what displacement vector in this subspace gets scaled by the biggest factor; not surprisingly the answer is $\mathbf{v}_2$ and the scale factor is $\mathbf{s}_2$.

Another way to think of $\mathbf{B}$ in the 3D case is that it transforms the three orthogonal unit vectors $\mathbf{v}_1$, $\mathbf{v}_2$ and $\mathbf{v}_3$ from frame t into the three orthogonal vectors $\mathbf{u}_1$, $\mathbf{u}_2$ and $\mathbf{u}_3$ in frame $t+1$ with lengths s_1, s_2, and s_3, respectively. Figure 6 illustrates this interpretation. In 2D the interpretation is similar except that $\mathbf{B}$ transforms the two orthogonal unit vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ from frame t to the two orthogonal vectors $\mathbf{u}_1$ and $\mathbf{u}_2$ in frame $t+1$ with lengths s_1 and s_2, respectively.

Figure 7 shows growth rate curves extracted from the *Flower 6* data. The principal (major and minor) growth rates are determined using corresponding nuclei between each pair of adjacent frames. The rates are then compound to show the cumulative growth relative to frame 1. Although we loosely refer to the two curves as "length" and "width"', there is no guarantee that the principal growth directions align with our macroscopic concept of length and width from looking at the sepal. There is also no guarantee that the principal growth direction remains consistent between all pairs of frames.

5.2 Bounding Box Growth Rates

The fact that the principal growth directions are not necessarily consistent with our macroscopic concept of length and width can be problematic. Hence, we developed another method to measure growth rates that first aligns the data points to an "atlas" in which the axes correspond to the macroscopic length and width directions. We then look at the growth rate of the axis-aligned bounding box.

Figure 8 shows the extracted nuclei from 6 frames of the *Flower 6* data. All nuclei from the same frame are shown in one color; hence, red shows nuclei from frame 1, green from frame 2, blue from frame 3 and so on. The colored rectangles show the (length-width) axis-aligned rectangles from each frame. The olive green lines show the tracking of each nucleus over the sequence. The black circle in the center represents the hypothesized single primordial nucleus before any divisions have taken place. Figure 9 shows the growth in

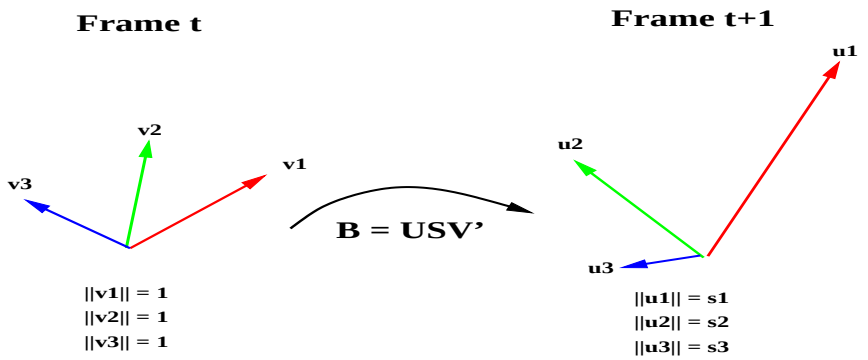

Figure 6: Principal growth directions and rates can be found from the singular value decomposition of the non-translation portion of the affine transformation between frame t and frame $t+1$. In particular, the vector $\mathbf{v}_1$ is the principal growth direction and s_1 is the principal growth rate.

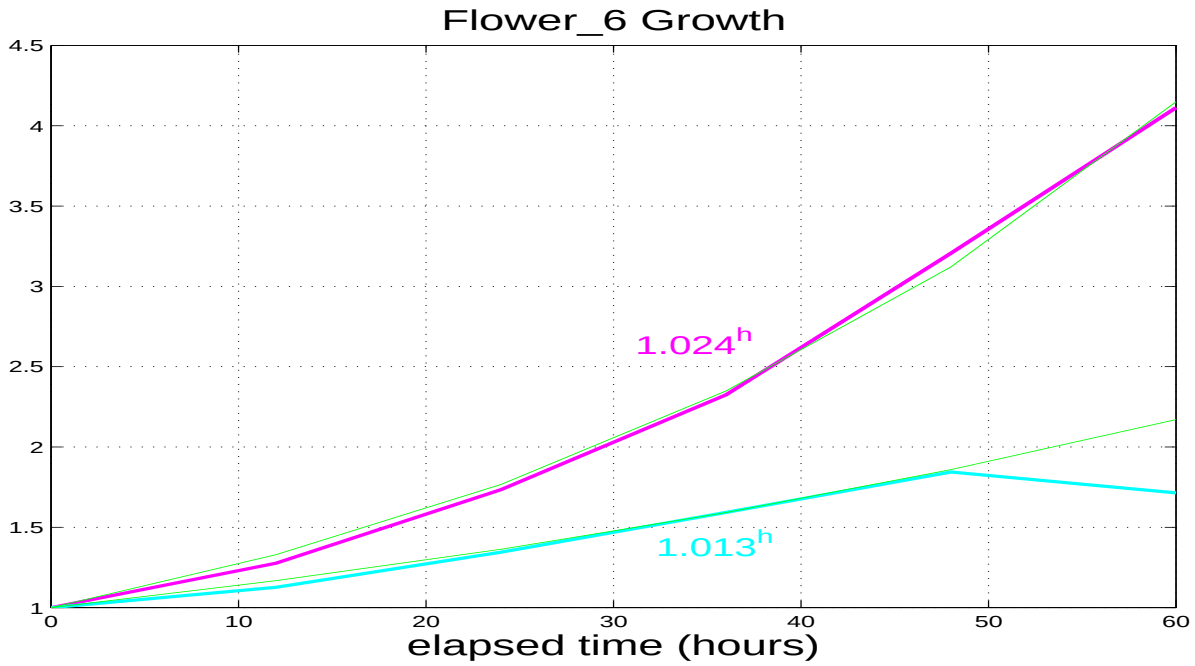

Figure 7: Growth rate curves extracted from *Flower 6* data using the compounded principal growth rates method. The magenta curve shows the compounded major growth rate starting from 1 at $t = 0$. The cyan curve shows the compounded minor growth rate starting from 1 at $t = 0$. The green curves show exponential fits to the growth curves.

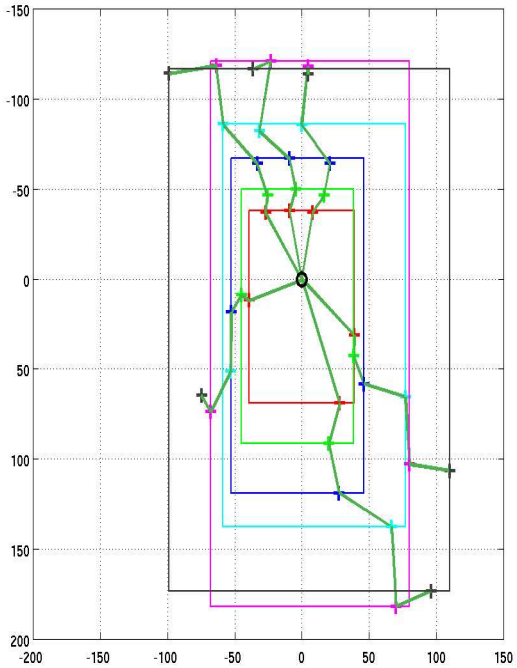

Figure 8: Growth rate curves extracted from *Flower 6* data using the macroscopic length and width axes and bounding box method. Nuclei from each frame are mapped to a coordinate frame that roughly aligns with the macroscopic length and width axes of the sepal. Nuclei from a given frame are shown in the same color along with a rectangle designating the axis-aligned bounding box. The olive green branches show the tracking of individual nuclei over time.

the bounding box length (red) and width (blue) over the sequence. Comparing with Figure 7, we see that in this case, at least, the two methods are reasonably consistent except at the tail end (between $h = 48$ and $h = 60$). Here, the principal growth axes between the last two frames in the sequence swapped from being roughly along the macroscopic length axis to being along the macroscopic width axis, which is probably not what we want (since the growth in width direction between the final two frames gets compounded with the growth in length direction from the earlier frames to yield the magenta curve of Figure 7).

5.3 Lineage Analysis

Knowing the detailed lineage history of particular cells is often a requirement for further analysis. Figure 10 shows lineage labels that were produced from the *Flower 5* image sequence. In particular, Figure 10c shows the number of descendents that were produced by cells starting at various locations in the early ($t = 1$) sepal.

With detailed tracking and lineage data, we can also conduct studies to look at the properties of individual cell lines versus time. Figure 11 shows a plot of nucleus area versus time for several sepal cells over the *Flower 6* sequence. The colored curves are for giant cells, which are not dividing. The black curves are for a single normal cell that is dividing, along with its daughters, granddaughters, etc. This experiment confirms that the nucleus area for dividing cells fluctuates within a fairly narrow range, while the nucleus area for the non-dividing giant cells grows exponentially over time.

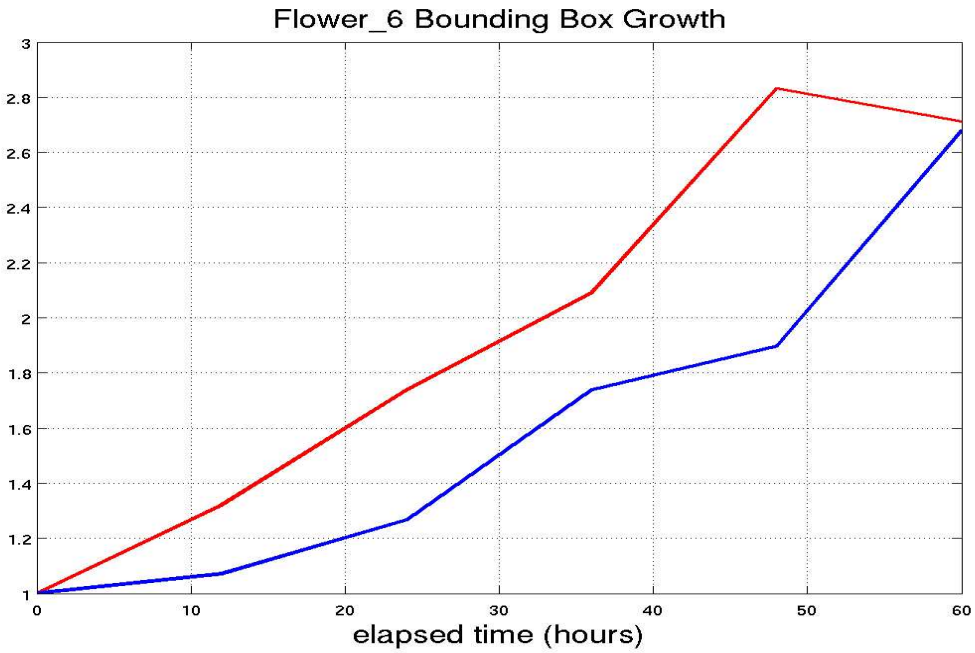

Figure 9: Growth rate curves extracted from *Flower 6* data using the bounding box method. The red curve represents compounded growth along the macroscopic length axis, while the blue curve represents compounded growth along the macroscopic width axis. Note that something unusual happens at the final time point, presumably due to an out-of-plane rotation of the sepal when placed under the microscope.

6 Nuclear Bounding Surfaces in 3D

Early in the project, a method was developed to recover nuclear bounding surfaces in 3D from an image stack and output the results as a triangulated hypersurface that is directly loadable by *Amira* [1]. The basic idea is to locate nucleus contours at each stack level and then link these across stack levels based on a contour distance measure (relative area of intersection could be used as an alternative) between contours on adjacent levels. Once contours are linked across stack levels, the contours corresponding to the same object are resampled to have the same number of points and similar origin, which makes the process of constructing a surface triangulation easy.

Figure 12a shows a maximum intensity projection image of real nuclei and plasma membranes in *Flower 74*. Figure 12b shows a similar image in which the real nuclei have been replaced by the extracted 3D nuclear bounding surfaces. Further details are available in [2].

Nucleus volume can be easily calculated from the surface triangulation assuming the nuclei are *star convex* with respect to their centroids. Essentially each surface triangle is connected to the object centroid to form a tetrahedron. Summing the individual volume contributions from each of the tetrahedrons gives the overall volume. See function `volumes_from_triangulated.m`. There is an issue, of course, on whether the volume inside the recovered nuclear envelope matches the true nucleus volume due to things such as the point spread function (PSF) of the microscope (particularly in the z-direction). Note, in particular, that the nuclei in Figure 12b appear to be cylindrical or rod-shaped; this shape is likely to be an artifact of the PSF.

7 Cell Compartment Segmentation and Analysis

A limited effort was also made to segment cell compartments from images of meristem provided by M. Heisler. A thresholding and morphology approach, similar to that used for segmenting sepal nuclei, provided some success. The procedure works as follows. The initial image is inverted, thresholded, and median-filtered to highlight the interiors of the cell compartments. Connected components are identified with `icluster.m`.

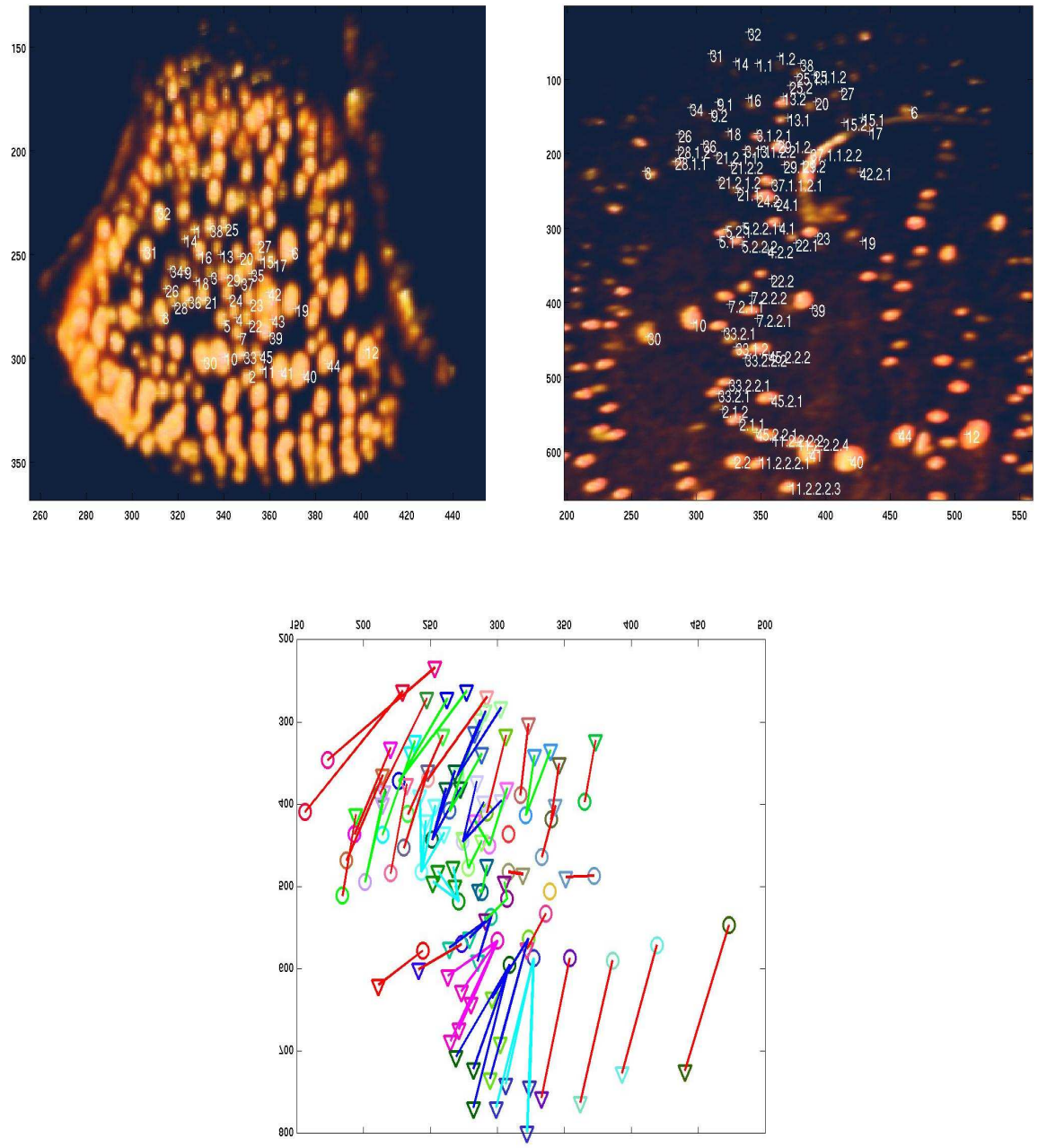

Figure 10: (a) Nucleus labels for the first image in sequence ($t = 1$). (b) Lineage labels for the final frame in sequence ($t = 17$). Note that the nucleus at the bottom middle (*11.2.2.2.3*) has undergone multiple rounds of division over the course of the sequence, while other nuclei (*40*, *12*) have not divided even once. (c) Mapping of nuclei from frame 1 to frame 17 with color coding of displacement vectors indicating number of descendents (red=1, green=2, blue=3, cyan = 4 or more).

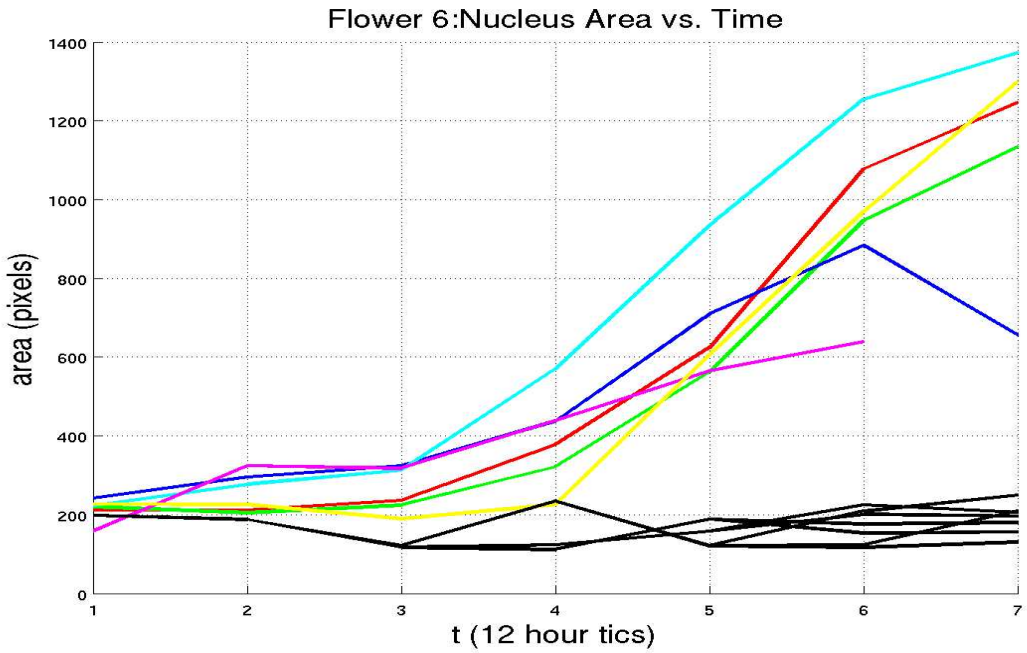

Figure 11: The colored curves show nucleus area versus time for several sepal giant cells over the *Flower 6* sequence. The black curves show nucleus area versus time for a normal sepal cell that is undergoing division. As the cell divides, we track each of the daughter cells (and their daughters), recording the nucleus area at each observation time. As expected, the nucleus area for the dividing cell remains within a fairly narrow range, while for the non-dividing giant cells, the area appears to grow exponentially versus time.

These components are then refined by eliminating regions that are too large or too small. Finally, boundary contours are extracted around each of the regions. Figure 13 show a meristem plasma membrane channel image and the corresponding compartment segmentation. Although we did not explore this option, we could potentially use the nucleus channel data in conjunction with the plasma membrane channel data to identify places where adjacent cell compartments were inappropriately joined into a single component. Such "super-cells" could then be split using watershed as in the sepal nuclei case (or the voronoi criteria).

8 Cell Compartment Models in 3D

Applying the 2D cell compartment segmentation approach of the previous section to each layer in a stack provides a set of contours versus stack level. Linking these contours across stack level as in the 3D nuclear boundary recovery algorithm enables all the contours from a particular cell compartment to be associated with one another. A triangulated polyhedron model of each cell compartment can then be recovered as shown in Figure 14.

The method for turning the 2D segmentations into a collection of 3D polyhedra is basically agnostic to the underlying 2D segmentation method. For example, planar junction graphs, such as those recovered by H. Jonnson, are often used to represent the network structure of the cell walls (edges and vertices). This representation can be easily transformed into a standard 2D segmentation (label image) or to a set of polygonal 2D cell compartment boundaries. The results can then be used as above to create 3D polyhedral representations of the cell compartments.

A slight drawback of the "union of polyhedra" representation shown in Figure 14 is that the contact between adjacent compartments which share a common cell wall is not explicit. In fact, there is is a small gap between adjacent polyhedra. A representation of the cell compartments in which adjacent compartments share a planar facet may be better-suited for certain types of computer modeling, for example, if the transport of some signaling chemical, such as auxin, from one compartment to the next is hypothesized to

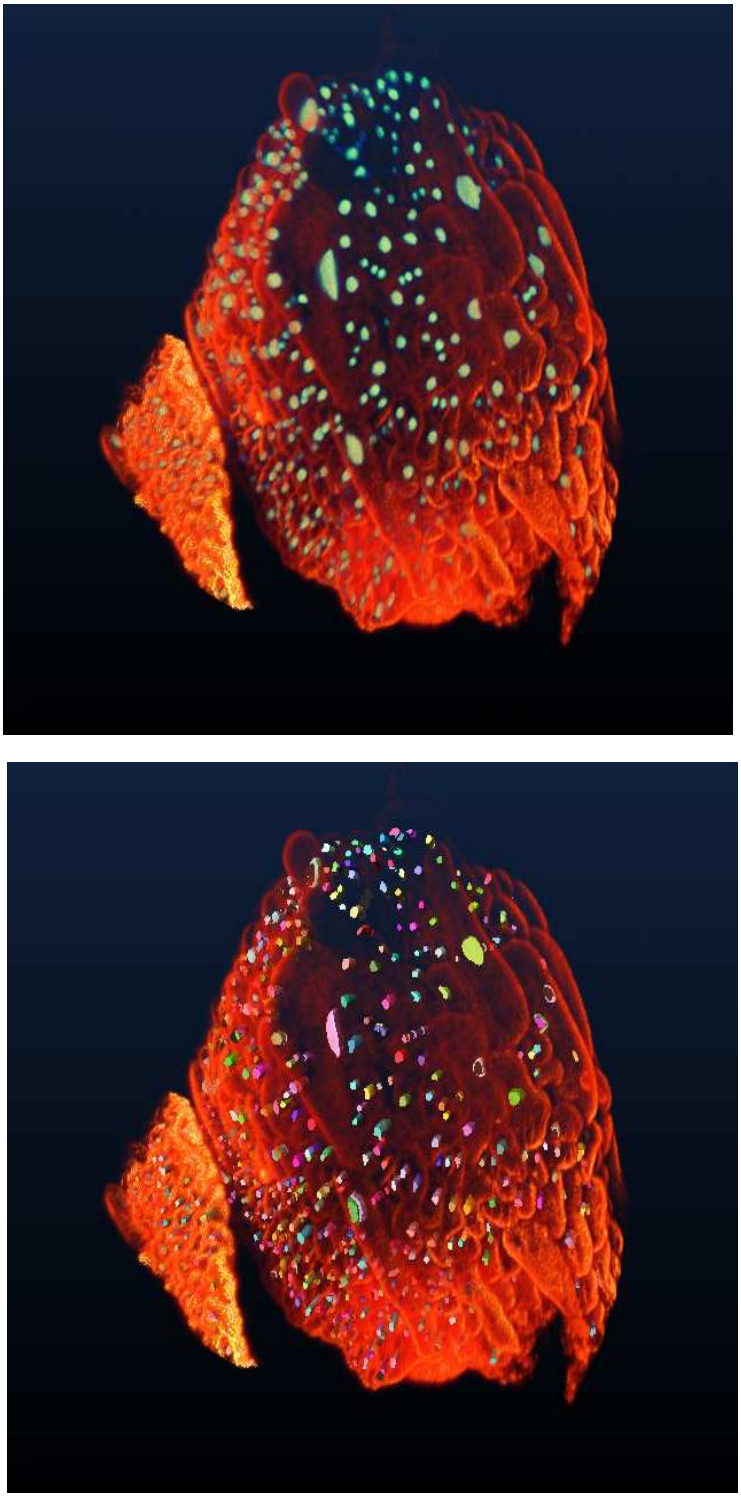

Figure 12: (a) Maximum intensity projection image of *flower74* with membrane channels in red and nuclei in green. (b) Maximum intensity projection image of *flower74* with nucleus channel replaced by randomly-colored, triangulated 3D nuclear bounding surface segmentations.

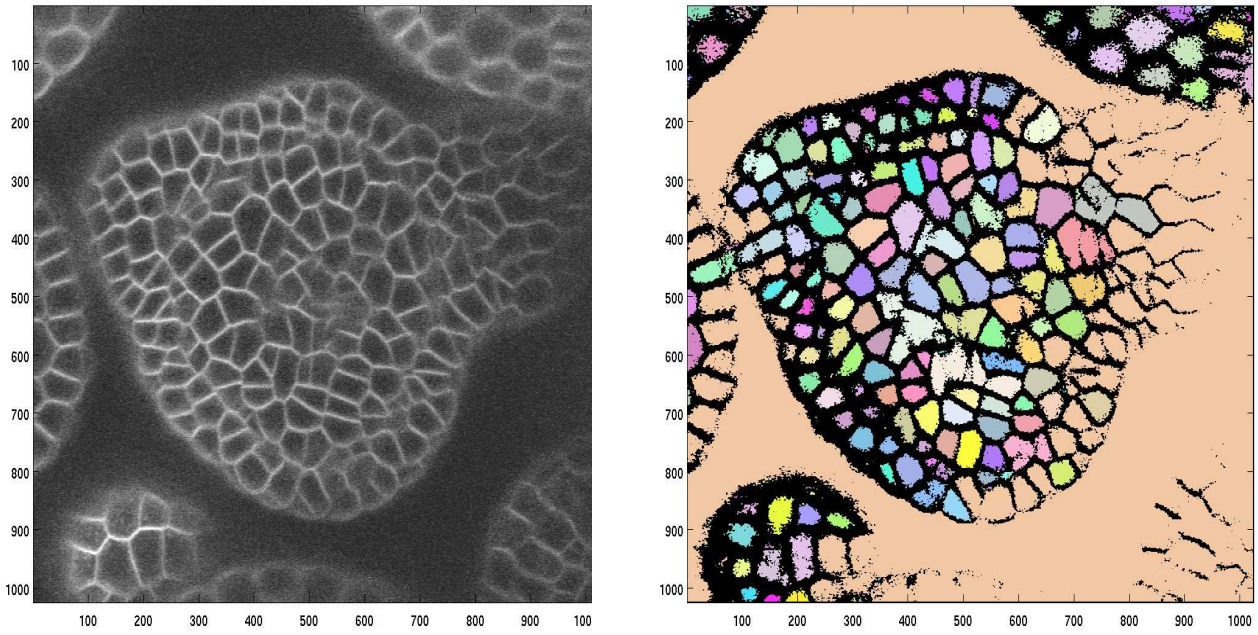

Figure 13: (a) Meristem image with square root ($\sqrt{\cdot}$) scaling. (b) Segmentation of meristem image into 2D cell compartments.

be proportional to the surface area of the contact region between the two compartments then it is somewhat difficult to get at this information from the "union of polyhedra" representation.

A series of experiments with synthetic data was conducted to explore algorithms for recovering a faceted "shared wall" representation of cell compartments. More details are provided in [4], but the main ideas are summarized here. The basic idea is to take a set of nuclei positions in 3D and construct an artificial 3D assembly of cell compartments using the voronoi criteria. Figure 16 shows a triangulated surface rendering of one such construction, along with equivalent voxelized representations (same cells, unfortunately different color scheme for the voxelized and triangulated versions).

Slice images were then obtained by mathematically transecting a plane (parallel to the x-y plane at various depths z) through the cell compartment assembly to produce a stack of 2D segmentations. Figure 17 shows a set of mathematical slices through a 3D Voronoi cell assembly at two adjacent slice levels. Here the 2D cell compartment regions are consistently-colored from one slice to the next (ground truth), but this would not be the case if the 2D regions were identified with a real segmentation algorithm.

The task then is to look at how these 2D segmentations with inconsistent labels between slices can be used to recover the 3D cell compartment structure. The first step in our proposed algorithm is to create a layered *affinity graph* in which the nodes at layer i consist of the segments found at the layer. Edges are defined based on the strength of connection between a segment in stack layer i and a segment in stack layer $i+1$. As shown in Figure 18, we used the relative area of overlap (square root of the area of intersection divided by area of union) as our measure of connection strength. Edges in the affinity graph are then pruned based on strength and by applying a mutual nearest neighbor criteria. If segment A on stack level i has its best match on level $i+1$ as segment B (and the match was of sufficient quality according to the relative area of overlap), then we also check whether the best match for segment B on level i is segment A. If so, these two segments were mutually the best match for each other, and the edge between them is retained. We refer to this as the forward-backward consistency check and is similar to the left-right consistency check that is commonly used to verify hypothesized correspondences in various stereo vision algorithms. Figure 19 shows the consistency check in more detail. Figure 20 shows the affinity graph after the consistency check has been used to prune links. In Figure 20b, *false positive* indicates that a link remains between two regions that is

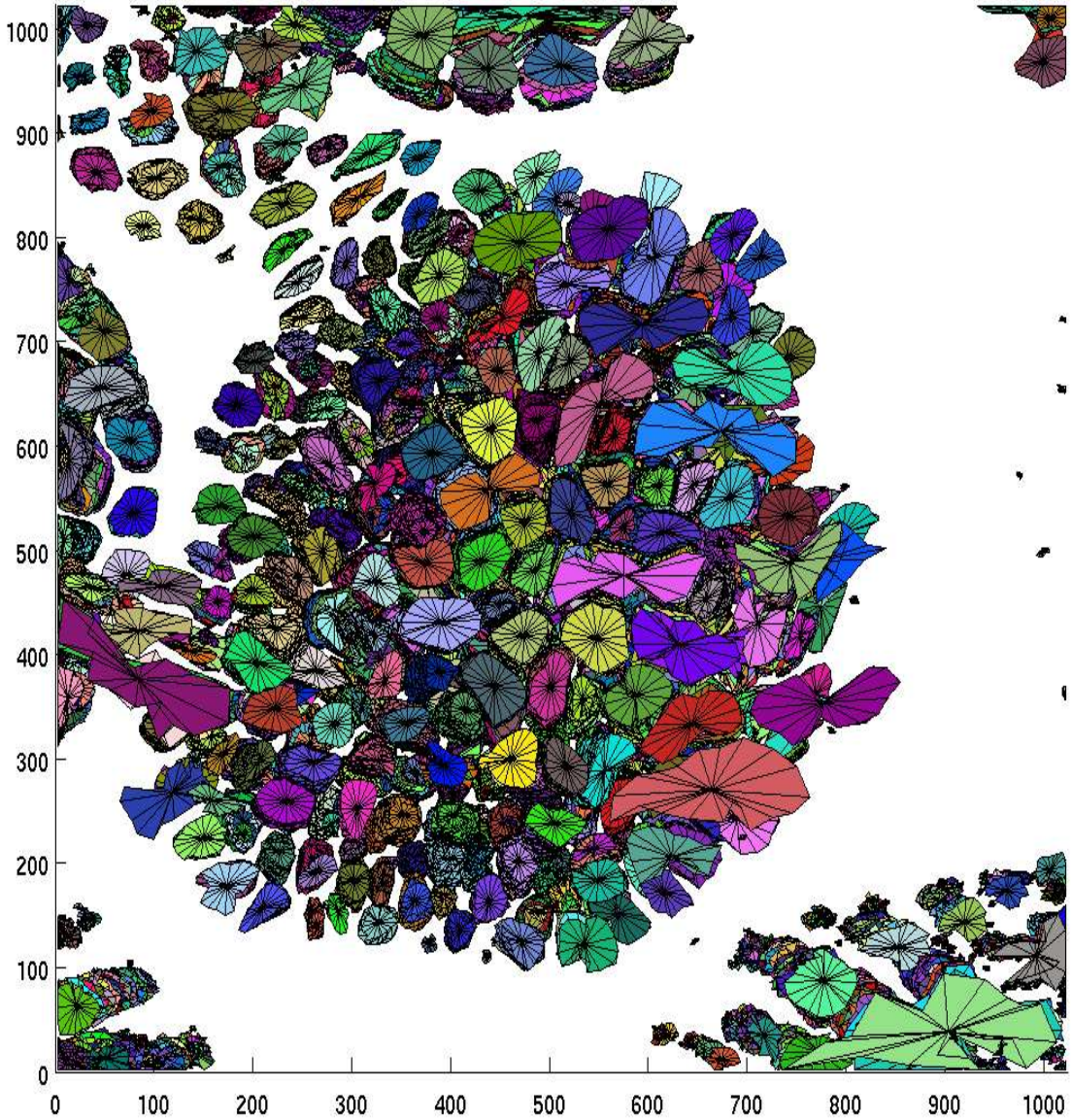

Figure 14: Combination of 2D cell compartment segmentations into a collection of 3D compartment models.

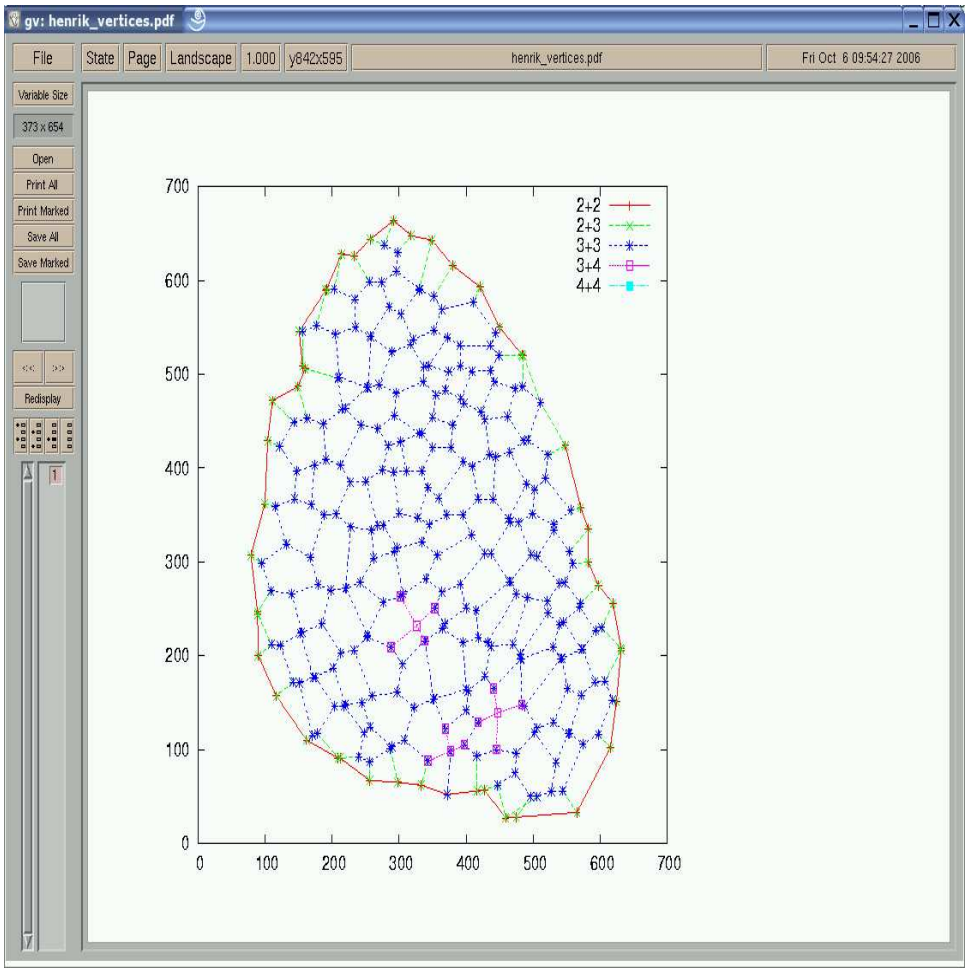

Figure 15: An alternative representation of 2D cell compartment structure is the planar junction graph in which cell walls are edges and vertices between walls are nodes. This representation can be converted to a standard 2D label image ("segmentation") and then to boundary contours. Graphic courtesy H. Jonnson.

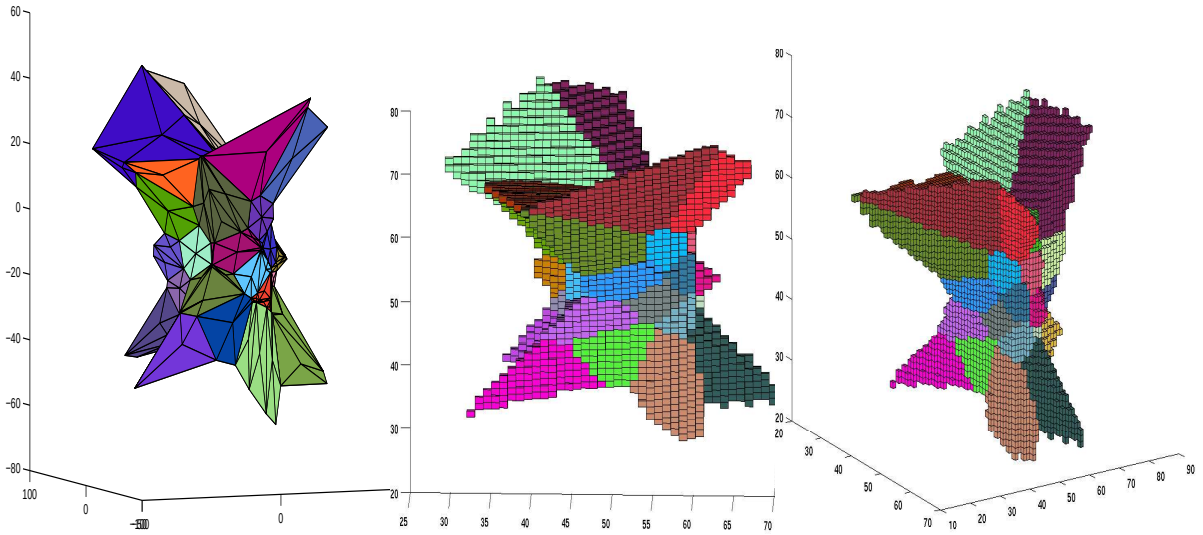

Figure 16: A synthetic 3D cell assembly created by applying the Voronoi construction to a set of 3D nucleus positions. (a) Triangulated representation. Each color represents a distinct cell compartment. (b) Voxelized representation from one viewpoint. (c) Voxelized representation from another viewpoint.

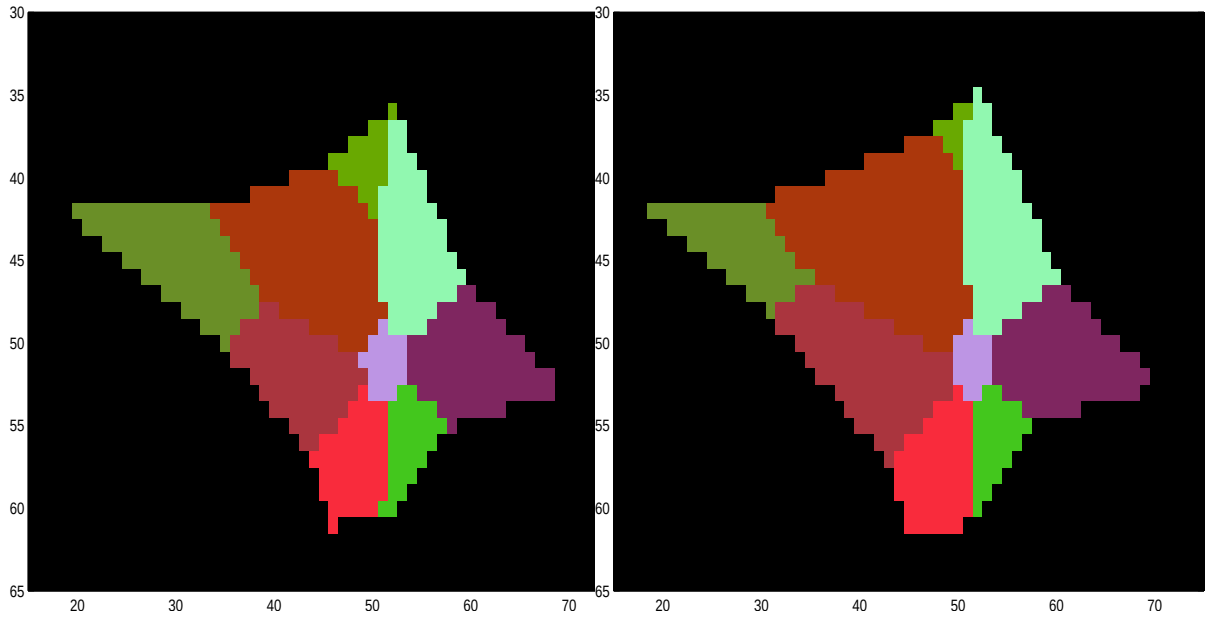

Figure 17: Transecting the 3D Voronoi cell assmebly with planes at different slice levels gives 2D slice images. (a) Slice 56. (b) Slice 57. Note that here the regions are consistently colored between slices according to the color of the 3D parent compartment; hence, the colors represent ground truth. This consistent coloring would not exist if the 2D regions were independently identified in each slice using a real 2D segmentation algorithm.

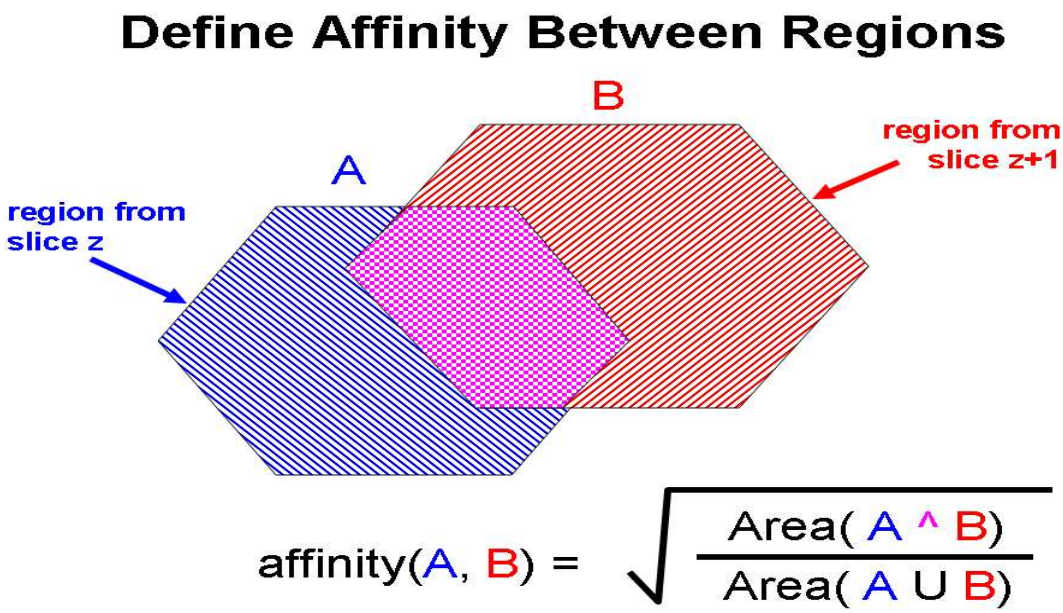

Figure 18: Our definition of affinity between two regions from adjacent stack levels is the square root of the area of intersection divided by the area of union.

incorrect. Such links are often the result of what we call the "snowman problem" illustrated in Figure 21; basically, two cell compartments are stacked directly on top of each other with their separating wall parallel to the slice planes. A *false negative* indicates that a true link was pruned. Usually, these occur only at the very top or very bottom slice level where the tapering shape of the cell compartment may result in a small 2D region in the slice image that is not well-linked to anything according to the affinity criterion.

Assuming we are able to link the individual 2D regions together across slice levels, the next step is to identify separating planes between objects that have contact with each other. Contact can be easily determined from a voxelized representation by looking in some neighborhood of each voxel that is marked as belonging to object k to see what other voxel labels appear nearby. Figure 22 shows a situation in which two objects are adjacent and we would like to recover a planar separating surface. Two approaches were tried. The first used only pixels at the contact region and tried to robustly fit a plane based on those pixels using a variation of the RANSAC algorithm [7]. This method sometimes gives spurious results, especially when the contact region was relatively small (unstable fit). A more reliable method was obtained by using *all* of the voxels from object k and object $k+1$ in a supervised machine learning framework in which a maximum margin linear separating hyperplane was found using the support vector machine approach. Figure 22b shows the separating plane recovered by linear SVM. Combining the set of planes bounding each object into a set of inequalities gives a representation of the cell compartment as a *polytope*. The polytope form can then be converted into a *facets form* as discussed ahead in Appendix 2.

There are several important limitations with this technique. First, it assumes that the cell compartments can be well represented by polytopes, which means the cell compartments must be convex. If we attempt to apply this approach to a "super cell" that resulted from a snowman situation, the results are likely to be poor as the polytope is defined by the intersection of half-spaces from all the planar facet constraints. Another problem is with cell compartments that border the background as they may have multiple planar contacts with the background, but the SVM tries to fit a single plane that best separate the object from all voxels that are marked as background.

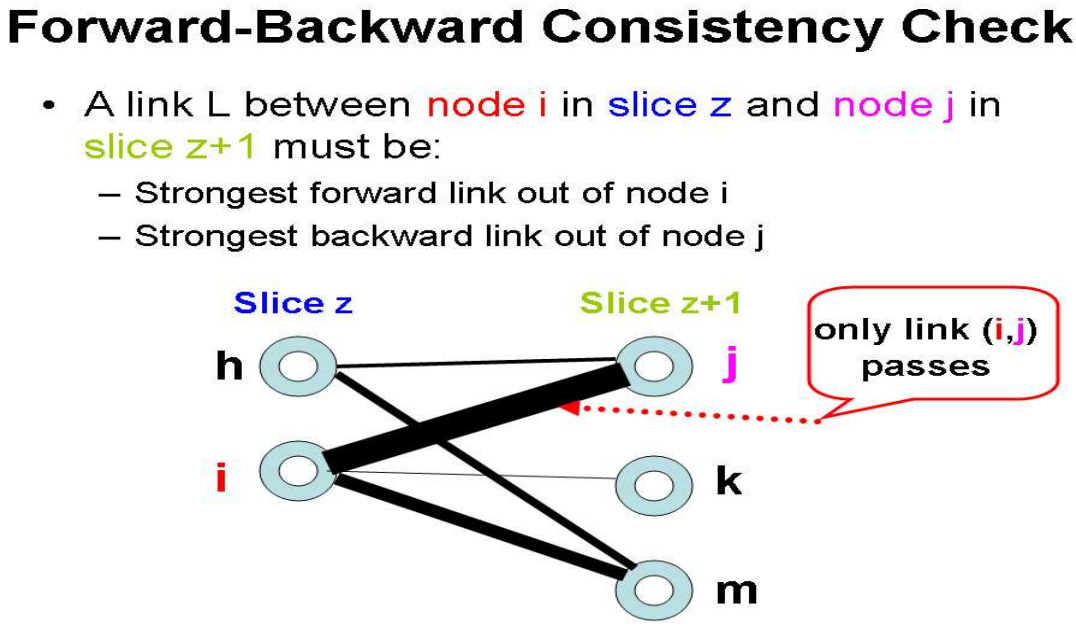

Figure 19: The forward-backward consistency check prunes links between slice layers that do not represent a mutual nearest neighbor relationship between nodes. In the illustration only the link between node i in slice z and node j in slice $z+1$ survives the consistency check.

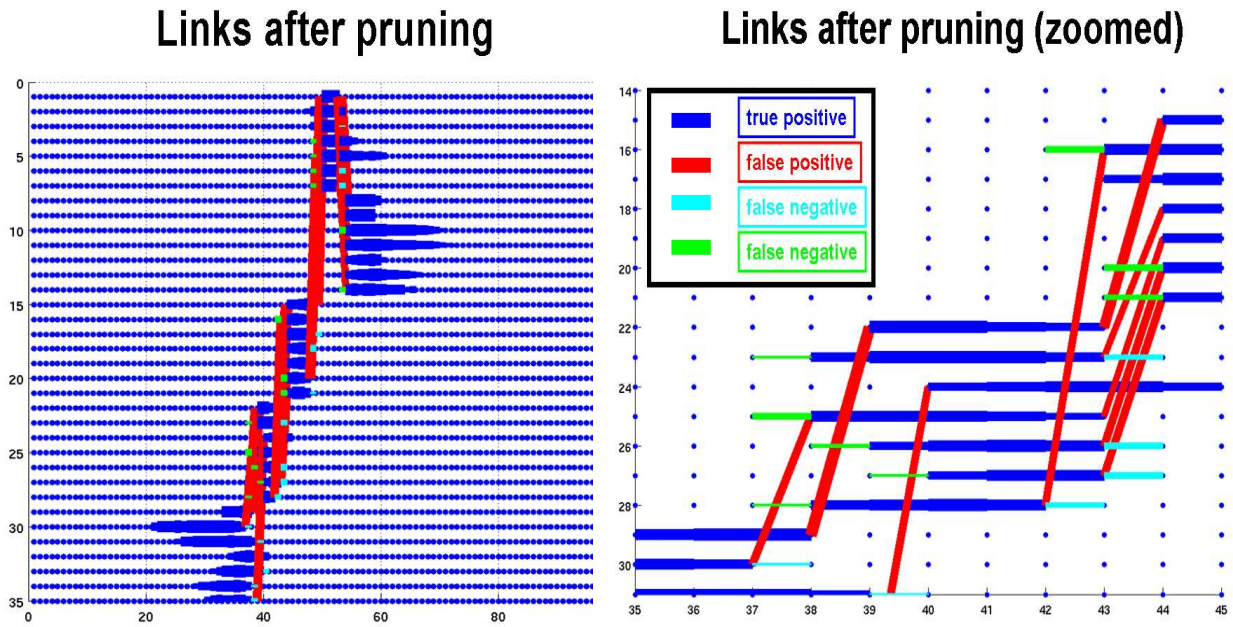

Figure 20: The affinity graph between regions in adjacent slice images after pruning using the forward-backward (mutual nearest neighbor) consistency check. (a) View showing all slice levels. (b) Zoomed in view of the affinity graph for slice levels 35 to 45.

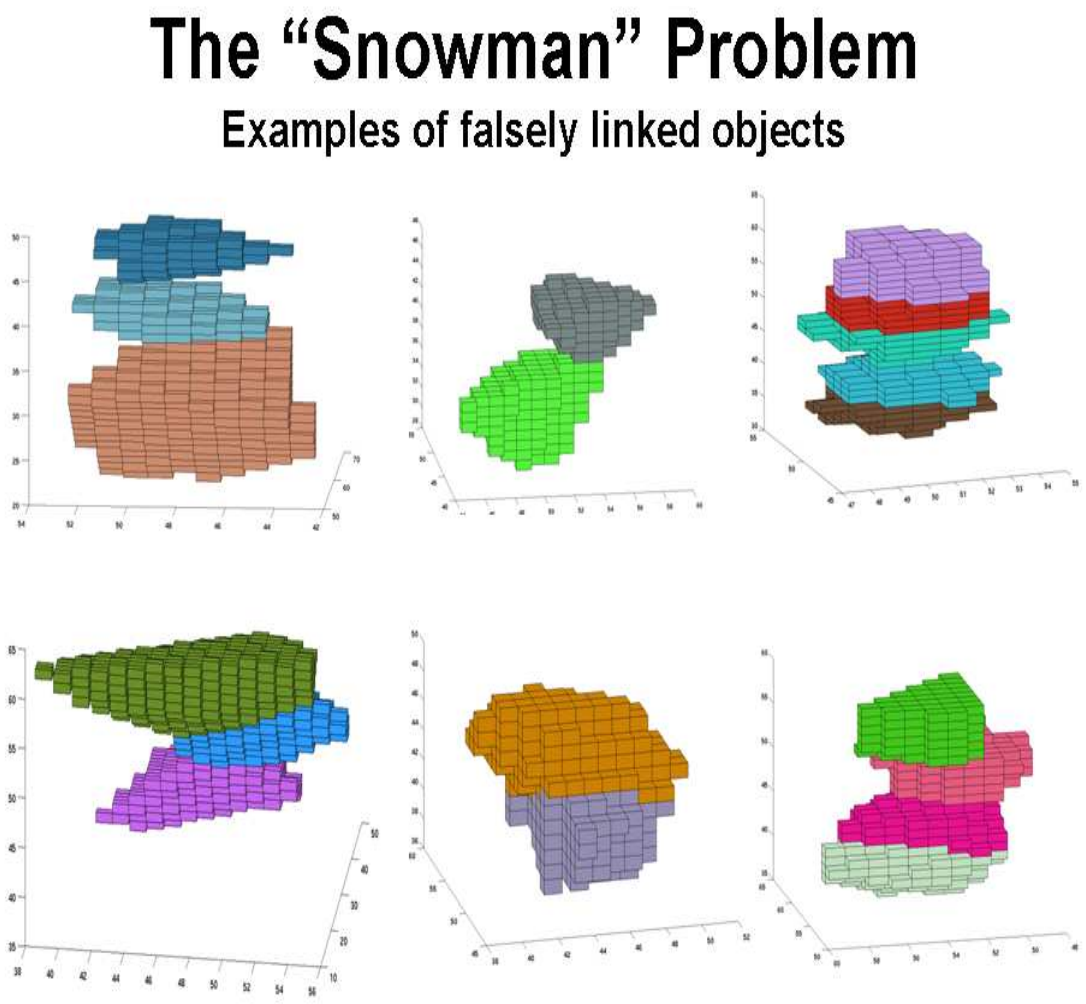

Figure 21: The “snowman problem” occurs when two cell compartments are stacked on top of one another with their separating cell wall parallel to the slice direction. Typically, this results in falsely linked objects. Using additional information, such as nucleus position, might offer some defense against these situations but has not been evaluated in our work to date.

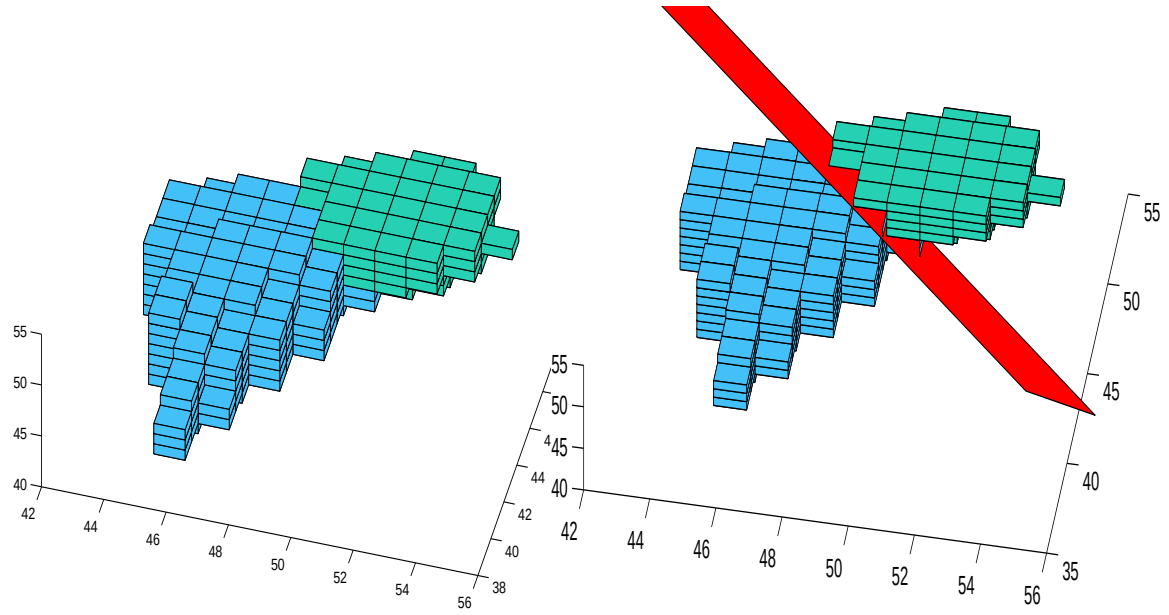

Figure 22: (a) Two objects with contact. (b) Planar separating surface. In this case, both RANSAC and the linear SVM give essentially the same answer.

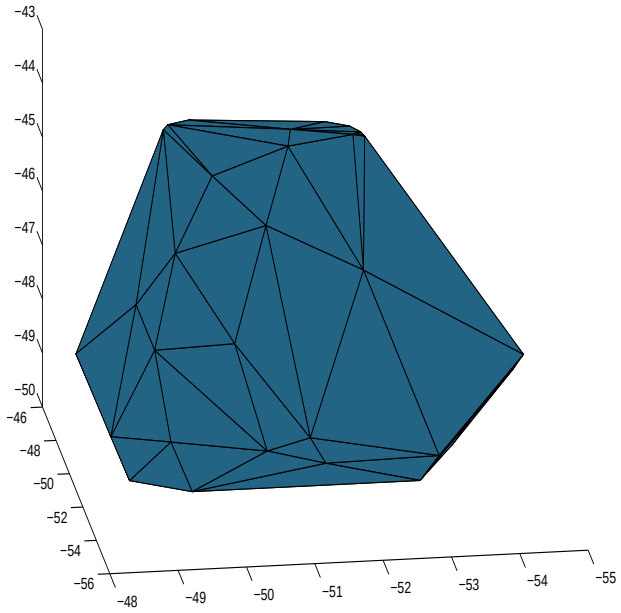

Figure 23: Recovered polytope for one cell compartment. Each planar facet is determined by the separating plane between this object and each of the adjacent neighboring objects.

9 Conclusion

In this memo, a number of algorithms developed by the author to support the *Computable Plant* project have been described. The main areas of activity included:

- **Segmentation of Nuclei in 2D**
- **Studies of Endoreduplication**
- **Growth and Lineage Analysis**
- **Recovering Nuclear Bounding Surfaces in 3D**
- **Segmenting Cell Compartments in 2D**
- **Recovering Cell Compartment Models in 3D**

Additional detail about the various algorithms can be found by studying the Matlab software package [3] or by contacting the author.

References

[1] *Amira Users Guide*

[2] M.C. Burl, A.H.K. Roeder, C.K. Ohno, E.D. Mjolsness, E.M. Meyerowitz, "Automatic Extraction of 3D Nuclear Bounding Surfaces from CLSM Imagery of Developing Arabidopsis Flowers", *Workshop on Multiscale Biological Imaging, Data Mining, and Informatics*, Santa Barbara, CA, (Sep 2006)

[3] M.C. Burl, Computable Plant Software Delivery (`burlcode`), (2008/09/03)
URL: `http://computableplant.caltech.edu/ burl/20080903/delivery_20080903.tar.gz`

[4] M.C. Burl, "Toward Recovery of 3D Structural Models of Plant Cell Assemblies", Presentation at Computable Plant Group Meeting, `20070321_PLANT.ppt`, (2007/03/21).
URL: `http://computableplant.caltech.edu/ burl/20070321/20070321_PLANT.ppt`

[5] J. Canny, "A Computational Approach to Edge Detection", *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 8. pp. 679-698, (1986).

[6] E. Mjolsness, E. Meyerowitz, "The Computable Plant", (2008)
URL: `http://computableplant.org/`

[7] M. Fischler, R. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography", *Comm. of the ACM*, (1981).

Appendix 1: Microscope Coordinate Frames

Figure 24 shows several coordinate systems that are relevant to the imaging experiments. The *metric microscope coordinate system (M-frame)* defines positions in space relative to an origin point O_M, which can be taken to be any convenient point along the optical axis of the microscope, e.g., at the "center" of the microscope or at the point where the optical axis exits the microscope. Positions are specified in appropriate metric units (m, mm, or μm). The *metric stack coordinate system (S-frame)* defines positions relative to an origin point O_S which is taken to be at the upper left corner of the top image in the stack. Positions are again specified in convenient metric units. The S-system is clearly related to the M-system by a rigid transformation (translation and rotation) that can be described as follows:

$$ {}^{\mathrm{S}}\mathbf{T}_{\mathrm{M}} = \begin{bmatrix} {}^{\mathrm{S}}\mathbf{R}_{\mathrm{M}} & \vdots & {}^{\mathrm{S}}\mathbf{t}_{\mathrm{M}} \\ \cdots & & \cdots \\ 0 \quad 0 \quad 0 & \vdots & 1 \end{bmatrix} \tag{15} $$

where ${}^{\mathrm{S}}\mathbf{R}_{\mathrm{M}}$ is the (3×3) rotation matrix between the two coordinate systems and ${}^{\mathrm{S}}\mathbf{t}_{\mathrm{M}}$ is the (3×1) translation vector. More precisely, the columns of ${}^{\mathrm{S}}\mathbf{R}_{\mathrm{M}}$ express the coordinate axes of the M-frame with respect to the S-frame. Thus, the first column expresses the X_M axis with respect to the S-frame and so on. The translation vector expresses the origin of the M-frame, O_M, with respect to the S-frame. Note that ${}^{\mathrm{S}}\mathbf{T}_{\mathrm{M}}$ is a (4×4) transformation matrix between homogeneous coordinates, so given a point $\mathbf{P}$ whose coordinates with respect to the M-frame are ${}^{\mathrm{M}}\mathbf{P}$, we can find its coordinates with respect to the S-frame as follows:

$$ \begin{bmatrix} {}^{\mathrm{S}}\mathbf{P} \\ 1 \end{bmatrix} = {}^{\mathrm{S}}\mathbf{T}_{\mathrm{M}} \cdot \begin{bmatrix} {}^{\mathrm{M}}\mathbf{P} \\ 1 \end{bmatrix} \tag{16} $$

Similarly, we can transform in the reverse direction using ${}^{\mathrm{M}}\mathbf{T}_{\mathrm{S}}$, which is given by:

$$ {}^{\mathrm{M}}\mathbf{T}_{\mathrm{S}} = \left({}^{\mathrm{S}}\mathbf{T}_{\mathrm{M}} \right)^{-1} \tag{17} $$

$$ = \begin{bmatrix} {}^{\mathrm{M}}\mathbf{R}_{\mathrm{S}} & \vdots & {}^{\mathrm{M}}\mathbf{t}_{\mathrm{S}} \\ \cdots & & \cdots \\ 0 \quad 0 \quad 0 & \vdots & 0 \end{bmatrix} \tag{18} $$

Note that

$$ {}^{\mathrm{M}}\mathbf{R}_{\mathrm{S}} = \left({}^{\mathrm{S}}\mathbf{R}_{\mathrm{M}} \right)^{T} \tag{19} $$

$$ {}^{\mathrm{M}}\mathbf{t}_{\mathrm{S}} = -\left({}^{\mathrm{S}}\mathbf{R}_{\mathrm{M}} \right)^{T} {}^{\mathrm{S}}\mathbf{t}_{\mathrm{M}} \tag{20} $$

For the M-frame and S-frame, there is typically no rotation, so ${}^{\mathrm{M}}\mathbf{R}_{\mathrm{S}}$ will be the (3×3) identity matrix. Hence, the M-coordinates and S-coordinates are related by a simple translation.

Although it is not shown in the figure, the *index-based stack coordinate system (I-frame)* is very important and closely related to the S-frame. Positions in the I-frame are specified based on integer indices for the pixels (or voxels). The I-frame and S-frame are related through scaling and possibly translation:

$$ \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = {}^{\mathrm{I}}\mathbf{T}_{\mathrm{S}} \cdot \begin{bmatrix} X_S \\ Y_S \\ Z_S \\ 1 \end{bmatrix} \tag{21} $$

$$= \begin{bmatrix} \frac{1}{\delta X} & 0 & 0 & -t_X\,\delta X \\ 0 & \frac{1}{\delta Y} & 0 & -t_Y\,\delta Y \\ 0 & 0 & \frac{1}{\delta Z} & -t_Z\,\delta Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X_S \\ Y_S \\ Z_S \\ 1 \end{bmatrix} \tag{22}$$

$$= \begin{bmatrix} (X_S - t_X)/\delta X \\ (Y_S - t_Y)/\delta Y \\ (Z_S - t_Z)/\delta Z \\ 1 \end{bmatrix} \tag{23}$$

For a 0-based indexing scheme (as in the C programming language), the integer coordinates of the first voxel are $(0, 0, 0)$), so the translations t_X, t_Y, and t_Z must all be zero making the S-frame and I-frame related purely by scaling. For a 1-based indexing scheme (as in Matlab), the integer coordinates of the first voxel are $(1, 1, 1)$, so the translation parameters must be $t_X = -\delta X$, $t_Y = -\delta Y$, and $t_Z = -\delta Z$.

Finally, there is a *metric object coordinate system (O-frame)* in which a macroscopic coordinate system is attached to the particular specimen being imaged. For the (hypothetical) leaf example shown in Figure 24 we define the O-frame to have its origin at some convenient landmark (intersection of two lateral veins with the medial vein), the X_O axis to be along the medial axis, the Z_O axis to be an outward normal to the plane of the leaf, and the Y_O axis to be $Z_O \times X_O$ to form a right-handed coordinate system.

For some applications, it may also be convenient to define a metric staging platform coordinate system (P-frame). The procedure for describing this frame relative to the other frames already discussed is straightforward.

The *pose* of a specimen is defined by the relationship between the O-frame and one of the other metric coordinate frames. Depending on the situation, the M-frame, S-frame, or P-frame could be used as the reference frame. Assuming the M-frame is used, the pose would be represented as ${}^{\mathrm{M}}\mathbf{T}_{\mathrm{O}}$ or equivalently as $\left\{ {}^{\mathrm{M}}\mathbf{R}_{\mathrm{O}},\, {}^{\mathrm{M}}\mathbf{t}_{\mathrm{O}} \right\}$.

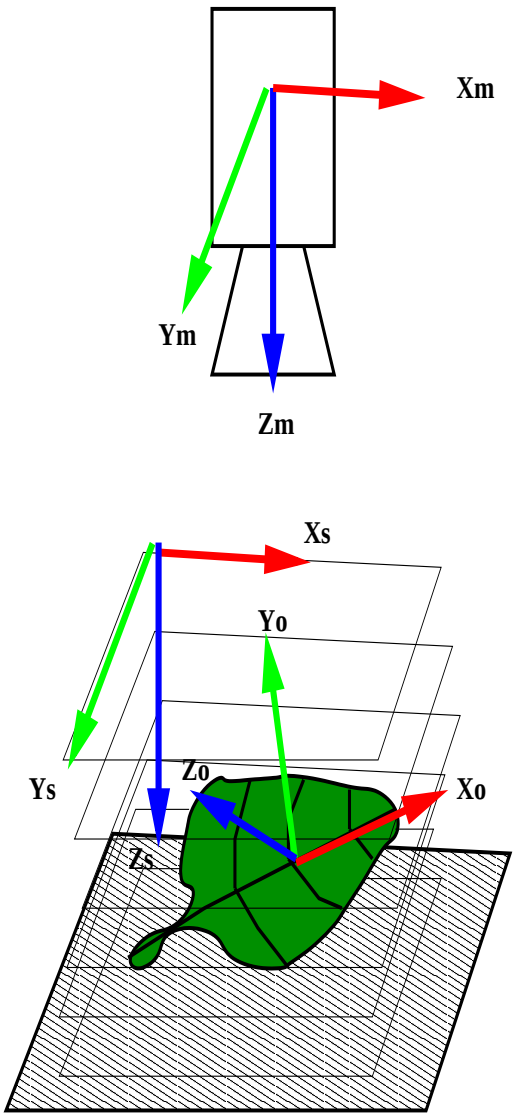

Figure 24: Various coordinate systems relevant to the imaging experiments: the metric microscope coordinate system (M-frame), the metric stack coordinate system (S-frame), and the metric object coordinate system (O-frame). The index-based stack coordinate system (I-frame), which uses integer-valued indices to describe the locations of pixels and voxels within the image stack, is not explicitly shown, but is very important and closely related to the S-system. For some applications, introducing a staging platform coordinate system (P-frame) may also be useful.

Appendix 2: Polyhedral Representations

There are several forms for representing polyhedrons. A number of software functions were developed for manipulating and converting between the various forms.

- **facets:** In the facets representation, the bounding polygon for each planar facet is given. The vertices of a facet must be ordered according to a consistent convention to indicate the direction of the outward surface normal. The facets form can be used to represent polyhedrons with concavities.
- **triangulated:** In the triangulated representation, the surface of the polyhedron is described by a set of triangles, which are in turn expressed through a set of vertices (taken over all triangles) and a set of index triples (indicating which three vertices form each particular triangle). Each index triple defines one triangle. As with the facet representation, the ordering of the indices in the triple is important as it signals the direction of the outward surface normal.
- **normals (or polytope form):** In the normals representation, which can only be used for convex polyhedrons, the outward surface normal (unit vector) and distance from the origin along the normal direction is given for each facet. A point $[x; y; z]$ on the facet satisfies:

$$0 \quad = \quad \mathbf{a}^T \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{24}$$

  where $\mathbf{a}$ consists of the unit normal and the (negative) distance from the origin. Combining the constraints of all the facets into a matrix $\mathbf{A}$ allows the polytope to be expressed as a matrix inequality:

$$\mathbf{Ap} \quad \leq \quad 0 \tag{25}$$

  where $\mathbf{p}$ is a point in homogeneous form as in Equation 24
- **voxels:** In the voxel representation, a 3D volume is broken up into discrete elements (small cubes or rectangular prisms) known as voxels. Each voxel contains an integer-valued ID indicating the identity of the object containing that voxel. This representation is analogous to the *label image* used to represent 2D segmentation results. It can be used to represent objects with concavities and even objects with disconnected pieces.