Chao, J.; Long, P.; Ward, E.S.; Ober, R.J., "Design and application of the Microscopy Image Analysis Tool," in *Engineering in Medicine and Biology Workshop, 2007 IEEE Dallas* , vol., no., pp.94-97, 11-12 Nov. 2007
doi: 10.1109/EMBSW.2007.4454182
keywords: {image processing;mathematics computing;object-oriented methods;random-access storage;MATLAB;MIATool;RAM;image editing tools;image pointer arrays;image processing;microscopy image analysis tool;object-oriented design;Chaos;Displays;Hardware;Image analysis;Image processing;Image storage;Instruments;MATLAB;Microscopy;Software tools},
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4454182&isnumber=4454150

Design and application of the Microscopy Image Analysis Tool

Jerry Chao, Palmer Long, E. Sally Ward and Raimund J. Ober*

Abstract— **Advancements in microscopy instrumentation have resulted in larger volumes of acquired image data and, consequently, increased memory and space requirements for the processing and storage of the data. To address the issue in software, the Microscopy Image Analysis Tool (MIATool) was created to support processing of large image sets that makes efficient use of available resources. Implemented in MATLAB using object-oriented design, MIATool works with image pointer arrays to utilize RAM effectively and to support the analysis of different interpretations of data. Furthermore, the software provides image editing tools which operate on parameter objects that are saved in lieu of processed images to exploit the available disk space. A detailed image analysis example is given to illustrate the design and features of MIATool.**

I. Introduction

Recent advances in microscopy imaging hardware have given investigators the ability to acquire images at higher speeds, an improvement which often translates to a significantly larger number of acquired images. A larger amount of data in turn imposes higher memory and disk space requirements for data processing and storage. While memory and storage limitations can be addressed by upgrading the necessary computer hardware, an alternative is to design and use image processing software applications that make more efficient use of the existing computer hardware.

We present in this paper some general principles behind the design of the Microscopy Image Analysis Tool, or MIATool (www4.utsouthwestern.edu/wardlab/miatool). In particular, we discuss the use of pointer arrays to effectively deal with large image data sets as well as differing interpretations and analyses of such data sets. For a synopsis of the various capabilities of MIATool, see Table I. We begin with a discussion in Section II on the choice of the MATLAB programming language and the object-oriented design methodology for the implementation of MIATool [1], [2]. We then give in Section III a microscopy image analysis example that illustrates some typical tasks that MIATool was designed to be able to address. This is followed in Section IV by a discussion of the role that RAM-efficient pointer arrays play in MIATool and a description of the viewer that provides the basics for working with pointer arrays. In Section V, focus is given to image editing tools and to parameter objects which provide a space-efficient alternative for the disk storage of processed images. In Sections VI and VII, attention is turned to the storage management of images and objects and to MIATool's expandability. We conclude our presentation in Section VIII with a continuation of the example from Section III that shows in detail how MIATool can be used to accomplish its various analysis objectives.

TABLE I
Current Capabilities of MIATool

General	Support for different interpretations of an image data set via RAM-efficient N-dimensional image pointer arrays.
	Automatic organization of image data and associated pointer arrays and modifying parameters on disk.
Viewer	Traversal of N-dimensional set of images using sliders.
	Display of images individually, or concurrently in separate windows or as RGB overlays.
	Display of images as mesh or contour plots.
Image Processing	Tools for pixel intensity adjustment, cropping, labeling, and manual segmentation.
	Flexibility to apply different image modifying parameters to different images in an image set.
	Storage of image modifying parameter values in space-saving parameter objects.

II. MATLAB and Object-Oriented Design

The MATLAB technical computing language was chosen for the implementation of MIATool primarily for the fact that its computing engine is optimized for array operations. It is thus suitable for the processing and analysis of digital images as well as pointer arrays which provide the basic framework for image analysis in MIATool. Importantly, MATLAB supports object-oriented programming, the design philosophy adopted by MIATool for code manageability and expandability [1], [2]. Just about everything in MIATool, including core components such as the image viewer and the various image editing tools, is implemented with objects. In addition to providing an intuitive mapping to the entities they represent, objects facilitate the organization of code, and as we will see in Section V, serve as intrinsic containers for the storage of image modifying parameters.

III. A Microscopy Image Analysis Example

We now introduce an example of an image analysis task that is often encountered in our laboratory. This example serves to illustrate the design specifications for MIATool. In a live cell experiment, two differently labeled organelles are imaged in a cell. For example, fast moving tubules are

The work was supported in part by NIH grants RO1 GM071048, RO1 AI050747, and RO1 AI039167. *Corresponding author, email: ober@utdallas.edu

J. Chao and R. Ober are with the Department of Electrical Engineering, University of Texas at Dallas, Richardson, TX 75080, USA, and the Department of Immunology, University of Texas Southwestern Medical Center, Dallas, TX 75390, USA jchao3@hotmail.com, ober@utdallas.edu

P. Long and E. Sally Ward are with the Department of Immunology, University of Texas Southwestern Medical Center, Dallas, TX 75390, USA palmerlong1@yahoo.com, sally.ward@utsouthwestern.edu

978-1-4244-1626-4/07/$25.00 ©2007 IEEE.

labeled with a ("green") GFP tag, whereas slow moving sorting endosomes are labeled with a ("red") RFP tag. The imaging is carried out with one camera which alternately takes nine images of "green" labeled tubules followed by one image of the "red" labeled endosomes. This sequence is repeated 1200 times to give a total of 12000 images of 0.85 MB each. The images are streamed directly to the hard drive, giving a data set of around 10 GB. The full size of the data set already shows that not all images can be stored in RAM for a typical PC. From this we see that one design criterion of MIATool has to be the efficient analysis and viewing of an image set whose size exceeds the RAM of a typical PC.

When analyzing this data, the microscopist typically wants to examine different aspects of the image set. As a first verification of the success of the experiment, often the "red" channel and the "green" channel will be viewed separately. In a conventional design of an analysis software package, the user would achieve this by creating two arrays of images, one containing the "red" images and the other containing the "green" images. After this initial check the experimentalist would typically want to study the interactions between the "green" tubules and the "red" endosomes. For this purpose overlays would be created by reading a "red" image into the "R" channel of an RGB image format and a "green" image into the "G" channel of the same RGB image. Since the "red" images are acquired at a much slower rate, each "red" image would have to be paired with each of its preceding nine "green" images to make the overlays. This time alignment by repetition of the "red" images is justified by the relative immobility of the endosomes with respect to the tubules.

This example shows that for a given experimental data set different forms of presentation can be used to display and analyze the data. In this case, for one experimental data set two arrays of grayscale images and one array of RGB images are created. Using a conventional software design this would lead to a significant increase in the need for RAM/disk space and would pose potential problems when switching from one data interpretation to another.

IV. Image Pointer Arrays and Viewer

In an effort to make more efficient use of the available RAM, MIATool uses arrays of image pointers rather than arrays of actual images. Image pointers are nothing more than addresses of the physical locations of the images and are therefore much smaller in size and occupy significantly less memory than the images they reference. In addition, one would expect that the creation and manipulation of pointer arrays in RAM be faster than doing the same with actual image arrays. A pointer in MIATool is simply the path to an image's location on disk, and the basic unit of processing in MIATool is just an N-dimensional array consisting of such pointers. As we saw in Section III, the value of N is dependent on the particular analysis task.

Given an N-dimensional pointer array, perhaps the most basic of requirements is to be able to view the images referenced by the pointers. For this purpose MIATool provides a viewer that opens with as many sliders as there are dimensions in the input pointer array. By scrolling a slider, the dimension the slider corresponds to is linearly traversed and the images referenced by the traversed pointers are displayed one after another. Referring to the example in Section III, if we were to construct a 2-dimensional array containing pointers to all the "red" images in the first row and pointers to all the "green" images in the second row, then the "red" images would be displayed by setting the first slider to the value 1 (for row 1) and scrolling the second slider. Fig. 1 shows an instance of the MIATool viewer that was opened for the viewing of an image set referenced by a 2-dimensional pointer array.

A pointer-referenced image must always be loaded from disk before it can be displayed or processed. To make the MIATool viewer a reasonable choice for image display, the speed of image retrieval must be sufficiently fast. The on-the-fly loading of images in MIATool is therefore accomplished with custom C language image retrieval routines specifically optimized for speed.

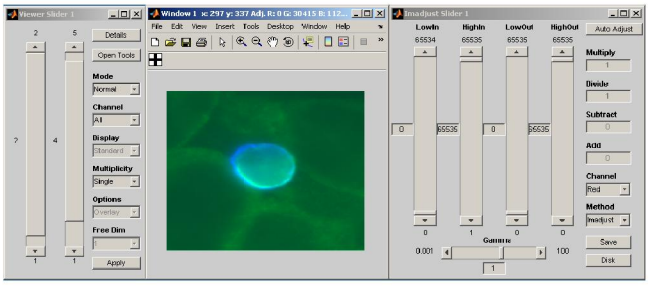

Fig. 1. MIATool viewer and intensity adjustment tool.

V. Image Editing Tools and Parameter Objects

In addition to the viewer which allows us to traverse a pointer array and display the referenced images, MIATool provides various editing tools for modifying the pointer-referenced images. The crop tool, for instance, is used to crop an image, and the intensity adjustment tool is used to modify the pixel intensities of an image. All tools in MIATool are user-interactive and operate on the image that is currently displayed in the viewer. When modified, the displayed image is immediately refreshed to give visual feedback to the user. An instance of MIATool's intensity adjustment tool is shown next to its associated viewer in Fig. 1.

Images referenced by the pointers in a pointer array may need to be modified differently from one another. Therefore, as the pointers are traversed in the viewer and modifications are made to one image after another using some tool, some form of storage is needed to retain for each pointer a potentially unique description of how its referenced image should be modified. To address this necessity, a pointer array is associated with parameter objects. A parameter object is nothing more than a data container that stores a set of modifying parameter values for each pointer in a pointer array. Modifying parameter values are simply data that concisely describe a particular modification. In the case of cropping, for instance, the modifying parameter values are the coordinates

of the two pixels that define the rectangular region to extract from an image. Every tool in MIATool is designed to work with a specific type of parameter object. The crop tool, for instance, stores user-specified crop parameter values to a crop parameter object that is associated with the pointer array being traversed in the viewer.

Users of MIATool have the option to save the modifications they make in the form of actual modified images. MIATool, however, provides the alternative of saving just the parameter objects instead. A parameter object is much smaller than a corresponding set of modified images and hence requires significantly less disk space to store. Additionally, by saving a parameter object, a record is kept of the processing that has been performed on a set of images. When necessary, the modifying parameter values stored in parameter objects can always be used to generate actual modified images.

VI. Disk Storage in MIATool

Using MIATool to process a set of images can potentially generate many objects. A single set of images may be associated with multiple image pointer arrays (which are themselves implemented as objects), and each pointer array may be associated further with multiple types of parameter objects. The number of objects can increase even more if multiple parameter objects of the same type are associated with some or all of the pointer arrays. That is, there may well be reasons for cropping, intensity-adjusting, or otherwise modifying the same set of pointer-referenced images more than one way, and each different way of modifying the images can be stored in a parameter object.

To help users with the management of large numbers of objects, MIATool automatically saves to disk all objects associated with a set of images to a specific directory structure. The directory structure also contains the set of images. By storing the images in their own subdirectory, it provides a clear separation of a set of images from its associated objects. For the subdirectory containing the associated objects, the directory structure employs a hierarchical structure that reflects directly the relationship between the various objects. All parameter objects associated with a particular image pointer array, for example, would reside in subdirectories below the one that stores that pointer array object.

A particular instance of a directory structure is itself represented by an object. A directory object is essentially a table of contents that faithfully records the numbers and types of objects contained in a directory structure as well as the hierarchical relationships conveyed by a directory structure. As such, directory objects are used as a standard interface for the saving as well as the retrieval of objects to and from the physical directory structures.

VII. Getting More Out Of MIATool

To a large extent MIATool is intended to be used in a research environment (see [3], [4] for works that have made use of MIATool) and therefore needs to be expandable in order to address new requirements that arise as research projects evolve over time. Towards this end, the design of MIATool has focused on the development of uniform interfaces. A good example is tool and parameter object expandability. MIATool enforces a uniform way for all image editing tools to interface with the viewer. Similarly, all parameter objects share a common interface through which parameter values are stored and retrieved. When the need arises, new interactive tools and corresponding parameter objects can be added to the existing set of tools and parameter objects relatively easily provided that they conform to their respective interfaces. As an example, a prototype of a tool for manual image segmentation and a corresponding segmentation parameter object were developed in our lab for the false-color labeling of vesicles in live cell images. The tool works in conjunction with the viewer to allow user-interactive identification of vesicles in images and stores the pixel coordinates of the selected vesicles in segmentation parameter objects. By adhering to the uniform interfaces, the incorporation of the new tool and parameter object took relatively little effort, and the result was used extensively for data presentation in [5]. MIATool is thus not limited to its existing set of capabilities, but rather can be used as an environment for the development of new image processing capabilities. In our lab, efforts are under way to add new capabilities to MIATool (e.g., single molecule tracking).

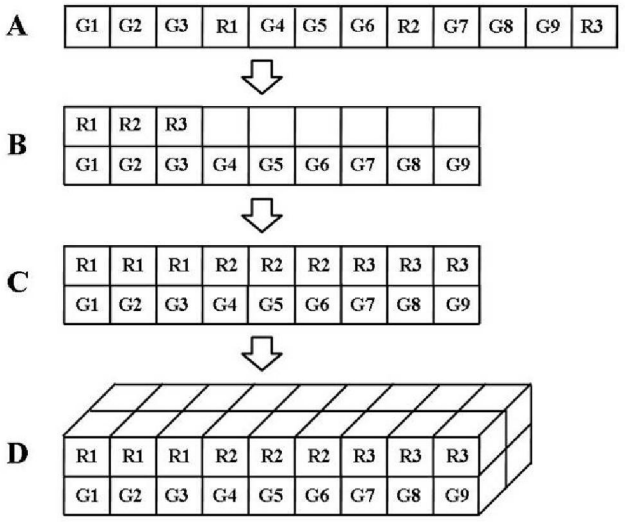

Fig. 2. Pointer array manipulation described in image analysis example, shown for a small but analogous data set. **A**. 1D array of pointers sorted by image acquisition time. **B**. 2D array of pointers separated by color. **C**. 2D pointer array time-aligned with repeated pointers. **D**. 3D pointer array consisting of two copies of the time-aligned 2D array. (G=green, R=red)

VIII. Example Continued

We now return to the example in Section III and describe in detail how we can use MIATool to perform the necessary analysis. Assume that all 12000 images are saved to a single directory on disk and are named 1 through 12000 to indicate the order in which they were acquired. We begin by importing them into a brand new MIATool directory structure. As part of the import process, MIATool generates by

default a 1-by-12000 pointer array containing in sequential order a pointer to each image. That is, we are given a 1-dimensional array containing 10800 "green" pointers and 1200 "red" pointers arranged in a repeating pattern of nine "green" followed by one "red" (see Fig. 2A). MIATool also automatically saves a copy of the 1-dimensional array to the newly created directory structure as a pointer array object.

To view our "red" and "green" images separately, we construct a 2-dimensional pointer array consisting of two rows. The first row contains only the 1200 "red" pointers which we can extract from positions 10, 20, 30, ..., and 12000 of the 1-by-12000 array. The second row contains only the 10800 "green" pointers which we can obtain from positions 1 through 9, 11 through 19, 21 through 29, ..., and 11991 through 11999 of the 1-by-12000 array. The result is a 2-by-10800 array that has 9600 empty slots in its first row (see Fig. 2B). We then open the MIATool viewer with the 2-dimensional array and find two sliders. To view the sequence of "red" images in the first row, we set the first slider to the value 1 and scroll the second slider. On the other hand, to view the sequence of "green" images in the second row, we set the first slider to the value 2 and scroll the second slider.

If we need to adjust the pixel intensities of the images for better display, we can associate the pointer array with a pixel intensity parameter object and use the MIATool intensity adjustment tool to make the appropriate modifications. As all tools in MIATool are equipped with support for propagating a set of modifying parameter values to all or a subset of the pointers in a pointer array, we can specify the same intensity settings for all images, different settings for the "red" and "green" images, or different settings for various arbitrary subsets of the pointer array. If desired, we can save the pixel intensity parameter object containing a record of our modifications for retrieval at a later time for viewing as the MIATool viewer is capable of extracting values from parameter objects to create modified images on-the-fly for display. The pixel intensity parameter object will be saved to the directory structure in a subdirectory below the one that stores the pointer array object for the 2-by-10800 array.

To view overlays of the "red" images with the "green" images, we take the same 2-by-10800 array, but completely fill the first row with nine pointers to each "red" image. To be more explicit, the new 2-by-10800 array has the same second row of "green" pointers, but has a first row that consists of nine pointers to image 10, followed by nine pointers to image 20, followed by nine pointers to image 30, ..., and finally nine pointers to image 12000 (see Fig. 2C). We then open the viewer with our time-aligned 2-by-10800 array and view the images in overlay mode. By scrolling the second slider in overlay mode, the viewer displays for each value of the second slider an RGB overlay of all images given by all possible values of the first slider. That is, the viewer displays for each column index of our array the "red" image in row 1 and the "green" image in row 2 together as an overlay.

To illustrate the flexibility of MIATool in dealing with different analysis requirements, we now expand on our example. Suppose the microscopist would also like to study the interaction of the "green" tubules with other RFP-tagged organelles that are much more weakly-labeled than the endosomes. Due to the significant difference in the distribution of RFP, the intensities of pixels corresponding to the weakly-labeled organelles cannot be increased to acceptable levels for viewing without saturating the pixels corresponding to the strongly-labeled endosomes. A solution is to overlay the "green" images separately with two sets of "red" images, one set that is intensity-adjusted for suitable display of the endosomes, and another that is intensity-adjusted for suitable display of the other organelles. To accomplish this with one pointer array, we take two identical copies of the time-aligned 2-by-10800 array which we created for overlay viewing, and simply put one copy on top of the other to form a 2-by-10800-by-2 array (see Fig. 2D). As we will see, using this 3-dimensional array as opposed to two separate 2-by-10800 arrays provides the advantage of being able to quickly switch back and forth between the two sets of overlays with a click of a viewer slider.

Given our 2-by-10800-by-2 pointer array, we use the intensity adjustment tool to adjust both sets of duplicate "green" pointers (i.e., the subset of pointers given by a first dimension value of 2) uniformly, but adjust one set of duplicate "red" pointers (i.e., the subset of pointers given by a first dimension value of 1 and a third dimension value of 1) with relatively low settings to properly display the strongly-labeled endosomes, and the other set of duplicate "red" pointers (i.e., the subset of pointers given by a first dimension value of 1 and a third dimension value of 2) with high settings to properly display the weakly-labeled organelles. Then with the viewer set to overlay mode, we scroll the second slider as in the case of the 2-by-10800 array to view the overlays. However, with our 3-dimensional array, we can use the third slider to specify which set of overlays to view. Specifically, overlays with the endosomes (weakly-labeled organelles) are viewed with the value of the third slider set to 1 (2). This example clearly shows that with multiple pointers referencing the same image, a single image can be associated with different modifying parameter values of the same type.

Note that throughout this entire sequence of analysis steps, no additional images are saved to disk. Instead, much smaller parameter objects are saved when desired.

REFERENCES

[1] The MathWorks, Inc., www.mathworks.com, *MATLAB Programming Ver. 7.1*, September 2005.

[2] E. J. Braude, Ed., *Object oriented analysis, design and testing: selected readings*, IEEE, 1998.

[3] R. J. Ober, C. Martinez, X. Lai, J. Zhou, and E. S. Ward, "Exocytosis of IgG as mediated by the receptor, FcRn: An analysis at the single molecule level," *PNAS*, vol. 101, pp. 11076–11081, 2004.

[4] P. Prabhat, S. Ram, E. S. Ward, and R. J. Ober, "Simultaneous imaging of different focal planes in fluorescence microscopy for the study of cellular dynamics in three dimensions," *IEEE Trans. Nanobioscience*, vol. 3, pp. 237–242, 2004.

[5] P. Prabhat, Z. Gan, J. Chao, S. Ram, C. Vaccaro, S. Gibbons, R. J. Ober, and E. S. Ward, "Elucidation of intracellular recycling pathways leading to exocytosis of the Fc receptor, FcRn, by using multifocal plane microscopy," *PNAS*, vol. 104, pp. 5889–5894, 2007.