Development of Nonlinear 3-D, and Novel 2-D Optical Microscope Imaging Systems for Time-lapse Imaging.

By: Robert J. Girardin
Lakehead University
December 3, 2004

Thesis Supervisor: Dr. Werden J. Keeler

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

Abstract

The design and development of a nonlinear two-photon, 3-D microscope imaging system, software for manipulation of 2-D and 3-D images, and the operation of a novel 2-D imaging system used for monitoring cultured live cell activity for many hours, is reported.

Included in the first portion of the thesis is the description of the final assembly of a mode-locked Ti:Sapphire laser producing 10 – 100 femtosecond pulses at high repetition rate. Also described is the construction and computer automation of a laser microscope with digitally controlled XY mirror laser beam scanning and with linear motor stage Z-stepping. This system was later used to image GFP tagged cultured live cancer cells.

As a companion instrument and 2-D reference system for comparative time-lapse imaging studies of cancer cells, a Nikon E400 fluorescence microscope and cooled CCD detector were purchased and integrated. Central objectives of the live cell studies were (a) to find a method of readily imaging untagged cells which can be cultured from needle biopsies taken from patients, and (b) to develop a method for semi-quantitatively ascertaining live cell vitality over time frames extending from minutes to hours. The short time scale is particularly useful as it provides information that cannot be obtained from monitoring the cell division process, which is measured in many hours. The ability to monitor cell viability is important when monitoring the effects of external treatments such as radiation, intense light, chemicals, or any combination of these.

The Nikon fluorescence microscope permitted automated collection of incubated, untagged live cell image data for many hours time duration via a technique called 'oblique illumination microscopy' (OIM), also referred to as 'oblique incidence reflection imaging'. Untagged cells are very difficult samples to see by means of commonly used imaging methods such as transmitted, fluorescence, or epi-reflection illumination methods. The ability of OIM to avoid the necessity of fluorescence tagging of the cell samples, its use of very low intensity illumination and hence low optical toxicity, made it the central technique used for many of the live cell time-lapse studies carried out in this work and presented as part of this thesis.

Acknowledgements

It gives me great pleasure to thank my research supervisor Dr. Werden J. Keeler for his counsel throughout this work. In addition to his guidance on the scientific front, I have thoroughly enjoyed our invaluable conversations on computers, finance, travelling, politics, and all the other things life has to offer and I am grateful for his friendship.

I would also like to thank the faculty and staff of the Lakehead University's Physics Department for their assistance and guidance throughout the years, with special thanks to Mr. George C. Anderson for machining several instrument components for this work.

I would like to extend my thanks and appreciation to Dr. John Th'ng and Dr. Sylvie Landry from the Northwestern Ontario Regional Cancer Centre (NWORCC) for providing numerous cultured live cell samples, and Ms. Marrisa Charboneau for the tissue thin-sectioning and characterization.

I would like to thank Graduate Studies at Lakehead University for the financial support it has provided in the form of Graduate Teaching Assistantships and Department of Physics for the Ontario Graduate Scholarship in Science and Technology (OGSST). I also thank NSERC for financial support obtained from grant funding through my supervisor.

Finally, I would like to thank those closest to me. Ma, Dad, Glen, Uncle Frank, and Dzedo, thank you for all your support and encouragement throughout the years. I would not have gotten this far without you. To Jen, Cory, and all my other dear friends, thank you for making sure that I remembered to have fun along the way. Lastly, to my wife Melissa, thank you for being there whenever I needed you, and still marrying me after all of this.

Table of Contents

List of Figures & Tables

About the CD

The accompanying CD contains the video animations referred to in Chapters 2 and 3. Some of the movies are stored in MPEG-4 format while the others are stored in Apple's QuickTime Movie format. The MPEG-4 format was chosen to compress excessively large movie files, which in uncompressed form can reach hundreds or thousands of megabytes. Since the format provides superior compression (an order of magnitude or more) with little loss in quality, the compression algorithm is quite processor intensive. For this reason, it is the recommendation of the author that the videos be viewed on a computer with a minimum of 256MB of RAM and a 1GHz processor. On some systems, the first time the animation is played, it may seem to hang or skip, due to the video being loaded into memory. Played the video a second time results in animation that is significantly smoother. Apple's QuickTime Player 6.0 or higher is required to view these videos and is freely available for download at:

http://www.apple.com/quicktime

Introduction

Microscope imaging systems generally fall into two categories separated by the type of excitation sources utilized. Incandescent or mercury lamp based microscopes provide intrinsically two-dimensional (2-D) images and are used in methods such as bright-field, fluorescence, differential interference contrast (DIC) and phase contrast microscopies. These are applications where the full field-of-view is illuminated simultaneously. Scanned laser sources, on the other hand, provide point-by-point illumination of the field-of-view and are used by continuous wave (CW), laser confocal, and pulsed laser nonlinear approaches such as two-photon excitation fluorescence (2PE) and second and third-harmonic generation imaging systems. The first category produces rapid wide-field 2-D images[1,2,3] while the second category requires XY scanning of the laser beam and Z-step movement of the stage supporting the sample[4,5,6].

One objective of research in this group is to investigate the use of light in the imaging and therapeutic treatment of cancer cells. While still little used in the treatment of cancer, photodynamic therapy (PDT) with or without the use of chemical sensitizers, has enormous potential because of its relative ease of delivery, localized and short-range treatment capabilities, relatively minor side effects, and low cost. Also, it is a therapy technique that can be re-applied several times if needed. In the past few years, in support of these interests, a standard fluorescence Nikon E400 microscope was purchased for reference purposes and routine 2-D imaging of cells and tissues. Additionally, much time and effort has been spent in assembling from parts, a femtosecond laser light source and a scanning three-dimensional (3-D) laser microscope. While assembly of the pulsed laser source and laser microscope system has been time consuming, the overall system costs have been kept to below one tenth that of the commercially available alternatives (in excess of $0.5M), permitting the research to go forward.

At present, several biotech and pharmaceutical companies, cancer treatment facilities and University research labs are using light in combination with photosensitive drugs to study and improve the efficacy of photodynamic therapy in disease treatment. A standard description of drug-based PDT is given at the National Institute of Health website[7]. Although it is well known that relatively intense light alone can easily kill live cells, little to no results have been reported on the use of properly directed intense light as a therapy in the treatment of cancer. While not advocating a chemical free approach to therapy *per se*, there are several advantages to a drug free approach. These include a simpler research approach, the removal of any side effects associated with the light sensitizing drugs, and circumvention of the often intrusive research restrictions imposed by the biotech companies who may be suppliers of the drugs. On the other hand, a significant disadvantage of the drug free approach for clinical applications, is the increased difficulty in guaranteeing that, in the treatment of a distributed tumour mass, all of the tissue is located and treated. But, for the *in vitro* studies of cancer cell lines, determining the efficacy of treatment with chemical free intense light alone would be of significant interest for the reasons above.

Ideally, in the study of cancer cells, one wishes to be able to image biological activity at the sub-cellular level in near real time in order to be able to monitor sample response to environmental change or the application of external stimuli. That is, one needs to be able to observe the cells with adequate signal-to-noise and resolution, and then differentially monitor the response of treated cells to those left untreated, in a field-of-view containing several cells. Because there is no prescribed method for determining long-term biological activity, a large part of this work was devoted to developing one.

In addition to the live cell studies discussed above, other objectives of this thesis work were (a) to complete the assembly of a mode-locked pulsed laser 3-D sectioning microscope, (b) to develop software that would integrate the functions of laser scanning, image collection

and 3-D rendering in a system being assembled in-house, and (c) to test the ability of the combined system by imaging samples that were primarily cancer cell lines available from collaborators at the Northwestern Ontario Regional Cancer Centre (NWORCC) in Thunder Bay.

Some time after commencing the work, unexpected new resources became available that broadened portion (c) of the work. Namely, a much wider variety of cultured cell lines than was anticipated, became accessible due to their development by Dr. Sylvie Landry and Dr. John Th'ng at the NWORCC. Also, a laser confocal microscope was established in the Lakehead University Instrument Lab by University Chair faculty member Dr. Heidi Schraft and was used briefly in this thesis.

Chapter 1

Three-Dimensional Imaging Microscope System

1.1 Two-Photon Scanning Laser Microscopy

1.1.1 Two- Photon Theory

At low light intensities, quantum mechanics predicts that an atom or molecule can only absorb a photon if the photon energy is equal to the energy difference between two of its energy levels, and the symmetry differences of the levels are appropriate. This law breaks down rapidly in the high intensity world of nonlinear optics. Multiphoton excitation occurs when an atom or molecule absorbs multiple photons whose combined energy sums to that of an allowed transition energy. This latter effect was first predicted by Maria Goppert Mayer[8] in 1931 and can be visualized through the simplified Jablonski diagram in Figure 1.1 [9,10,11].

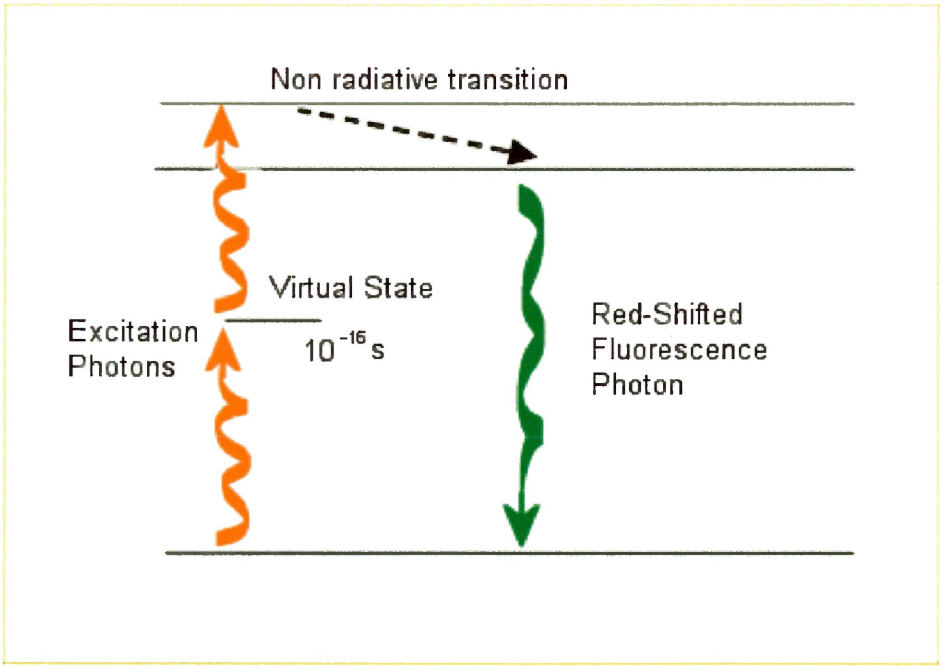


Figure 1.1: Simplified Jablonski diagram of two-photon fluorescence.

The molecule first absorbs the initial photon, which excites it into a short-lived virtual state. If more photons of combined energy equal to the remaining energy difference are absorbed in a short enough time period (T ~ 10^{-16}s), then the molecule will behave as if it had absorbed one photon of the total energy difference. The wavelengths of the multiple photons need not be the same; rather, they are governed by the relationship:

$$\frac{1}{\lambda_s} = \frac{1}{\lambda_1} + \frac{1}{\lambda_2} + \cdots + \frac{1}{\lambda_n} \tag{1.1}$$

where λ_s is the wavelength required for single photon absorption and λ_i in the wavelength of the i^{th} (i = 1, 2, ..., n) photon absorbed.

For n photon excitation, the time averaged fluorescence intensity, $\langle F(t) \rangle$, is given by:

$$\langle F(t) \rangle = \frac{1}{n} g^{(n)} \phi \eta C \sigma_n \langle I_0(t) \rangle^n \int_V dr S^n(r) \tag{1.2}$$

where $g^{(n)}$ is the n^{th}-order temporal coherence of the excitation source, ϕ is the fluorescence collection efficiency of the system, η is the quantum efficiency of the molecule, C is the molecule concentration, σ^n is the n^{th}-order cross section of the molecule, $\langle I_0(t) \rangle^n$ is the n^{th} power of the average intensity at the focal point of the objective, and $\int_V dr S^n(r)$ is the spatial distribution of the incident light.

A majority of the factors in Eq'n (1.2) are system and fluorophore dependent, and must therefore be determined for the specific system being used at the time. The important relationship is:

$$\langle F(t) \rangle \propto \langle I_0(t) \rangle^n \tag{1.3}$$

Thus, the fluorescence detected varies directly with the n^{th} power of the incident beam intensity. The nature of multiphoton imaging provides many positive advantages for imaging, with the main effect being reduced out-of-focus fluorescence, which results in higher contrast three-dimensional sectioning capabilities.

Consider the case of two-photon absorption with n in Eq'n (1.3) equal to two. The average fluorescence intensity becomes:

$$\langle F(t) \rangle \propto \langle I_0(t) \rangle^2 \tag{1.4}$$

or

$$\langle F(t) \rangle \propto \frac{\langle P_0(t) \rangle^2}{R^4} \tag{1.5}$$

where $\langle P_0(t) \rangle$ is the average power of the input beam, and R is the distance from the focal point the objective. As can be seen in Eq'n (1.5) the fluorescence output is inversely proportional to the fourth power of the distance instead of the standard squared relationship. This means that the excitation of fluorophores outside of the focal volume is dramatically reduced. Also, since the excitation wavelengths are double those of single photon excitation, the exciting photons can penetrate more deeply into the sample due to reduced Rayleigh scattering (see Figure 1.2).

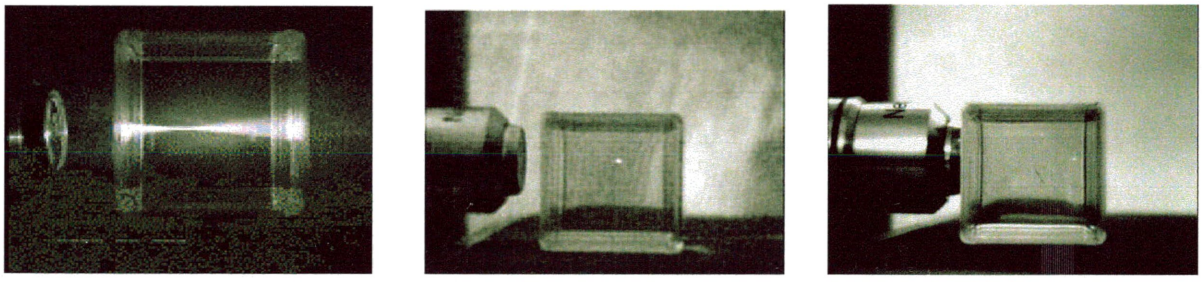


Figure 1.2: Stimulated fluorescence of a 19mm deep gelatin cube tagged with rhodamine. Single photon excitation (left) produces excitation throughout the depth of the sample resulting in significantly more sample volume to image. Two-photon excitation (center) confines fluorescence to a single sphere in the center of sample volume providing crisp resolution. The spot located at a depth of 19mm (right) illustrates the high penetration depth achievable with two-photon excitation.

As is well known[12,13], the resolution of an optical microscope is defined primarily by the numerical apertures (NA = n sin θ) of the objective and condenser lenses. For one pass through either lens:

Image brightness	$I \propto NA^2$
Focal volume depth	$D \propto 1/(NA)^2$
Resolving Power	$R = r_{Airy} \sim= 0.6\lambda/NA$

where r_{Airy} is the radius of the Airy disc that resolves two small objects.

When a microscope is used in the epifluorescence mode, the objective is both the condenser and pickup lens, so both brightness and depth of field D, scale as $NA^{\pm 4}$ and the resolving power is on the order of $0.6\lambda/2NA$. Image brightness decreases with magnification M as M^{-1}, thus fluorescence signal intensity per unit volume of sample in epifluorescence scales as NA^4/M^2. Therefore, to maximize both fluorescence image brightness and resolution, one should use a system with high overall NA and as low a magnification as is practical. Since the nonlinear two-photon effect depends on the square of the intensity, the effective Airy radius is defined by a $sinc^4$ function instead of the usual $sinc^2$ function. This provides an advantage to two-photon microscopes by effectively doubling their resolving power.

All of the reasons above permit two-photon microscopy to produce higher contrast sections at greater depths than that of traditional single-photon confocal microscopy as can be seen in Figure 1.3 [14]. This fact makes a two-photon microscope an ideal choice for 3-D imaging.

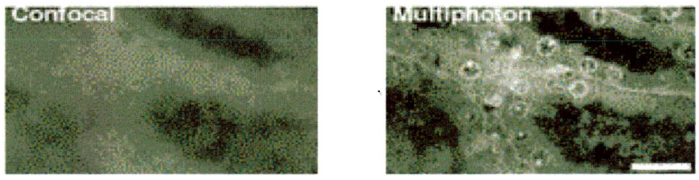


Figure 1.3: "Images of acid fucsin stained monkey kidney taken at a depth of 60 μm by confocal (left) and multiphoton microscopy (right). Laser intensities were adjusted to produce the same mean photons per pixel. The confocal image shows a significant increase in local background resulting in a lower contrast image. However, the multiphoton image maintains contrast even at significant depths within a light scattering sample."

1.1.2 Two-Photon Microscope

Pulsed Laser Stage

The internal structure of the femtosecond laser is depicted in Figure 1.4. The laser cavity is designed to support a maximum number of lasing modes (~10^6), as opposed to one or a limited number of modes in a CW laser. Since the Ti:Sapphire crystal is dispersive, with blue light traveling slower than red for low light intensities, the speed of light in the crystal for different wavelengths is given by:

$$v(\lambda) = \frac{c}{n(\lambda)} \tag{1.6}$$

with $n_{red} < n_{blue}$. This wavelength dependence requires that ancillary prisms be used to compensate for the different round-trip travel times experienced by each mode due to the presence of the Ti:Sapphire crystal in the cavity, and dispersion at mirror surfaces. With the prisms set correctly, the different mode round-trip times are equalized and can then become 'locked' (spatially and temporally) by a Kerr-lens high field nonlinear perturbation in the crystal. When the mode amplitudes combine, interferometrically, femtosecond pulses are produced with the very high peak powers characteristic of these lasers.

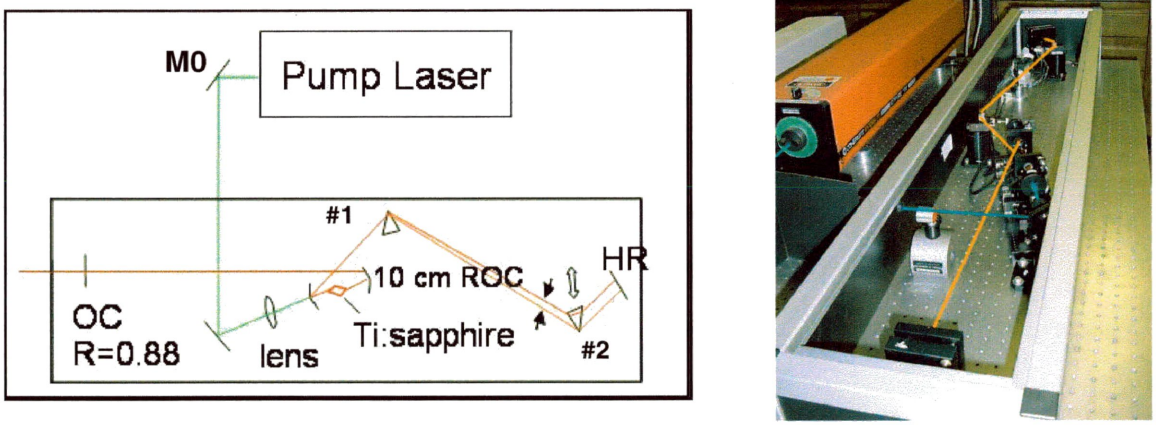


Figure 1.4: Internal schematic and output view of a KMLabs Ti:Sapphire laser with Ar+ pump.

The mode-locked laser system used in the 3-D imaging microscope was assembled in-house from a Coherent Innova 70-4 Ar^+ pump laser, a Kapteyn & Murnane (KMLabs) Ti:Sapphire laser kit, and ancillary parts. The characteristics of the KML laser have been thoroughly documented[15,16,17,18]. The pump laser is vintage 1988 and is rated nominally at 4 watts power. However, because it is kept clean with purged dry air from a Balston Purge Gas Generator, it still produces in excess of 5 watts, even after several thousand hours of use. M0 is a mirror-pair periscope that is used to rotate the polarization of the pump beam from vertical to horizontal as required by input for the Ti:Sapphire laser stage. Prism #1 disperses the pulse spectral bandwidth so that the redder end of the pulse must pass twice through ~1cm of quartz at prism #2 near the high reflector mirror (HR), per round trip in the cavity. The temporal delay associated with the glass path compensates for the higher light speed of the red portion of the pulse during passage through the Ti:Sapphire crystal. Such compensation facilitates mode-locking of the $\sim 10^6$ Fourier components to generate the femtosecond pulses. The beam is finally coupled out of the Ti:Sapphire laser at the output coupler mirror (OC), which has a reflectivity of 0.88 vs. 1.0 at the HR.

The operating mode of the Ti:Sapphire laser for the imaging experiments was always pulse mode at ~ 50-100fs pulse duration. The pulse duration is estimated from the locked bandwidth measurement and known properties of the Ti:Sapphire laser system. Bandwidth used for mode-locking was approximately 25-50nm in width, and could be centered from ~760nm to 810nm. The KMLabs laser operates at ~90 mega pulses per second (Mpps) pulse repetition rate due to the ~3 meter round-trip path in the laser cavity, and has an average output power of 300-600mW when pumped with 3.0 – 4.5 watts of pump power. This corresponds to ~ 3 - 6 nanojoules (nJ) per pulse. The average beam power must always be attenuated to a few milliwatts (pulse energies of a few picojoules) before the beam enters the microscope stage to prevent damage of the sample.

Scanning Microscope System:

A schematic of the pulsed laser, external pre-chirp assembly and microscope stage is seen in Figure 1.5. In order to monitor laser power and pulse bandwidth, a small fraction of the beam is guided via BS1 and a 400μm diameter multimode fibre into an Ocean Optics Inc., PC-1000 fibre optic spectrometer with 20μm x 400μm entrance slit. This spectrometer has a spectral resolution of about 0.5nm at the laser operating range. The major portion of the beam enters a microscope pre-chirp stage consisting of M3, M4 (New Focus 5102-NIR mirrors) and P1 and P2 quartz isosceles triangle prisms with apex angle of 69° to preserve Brewster angle incidence on both sides of each prism for the counter-propagating beams. Pre-chirping the pulses again, prior to entering the microscope, stretches the time domain of the pulse by forcing the 'blue-end' of the pulse to the leading temporal edge. Later, when passing through glass in the beam expansion lenses, the microscope objective and the dichroic mirror, the pulse again shortens spatially (hence temporally) due to the greater slowing of the 'blue-end' of the pulse in the optics. Apertures A1 and A2 are used to maintain pre-microscope alignment of the beam, and the neutral density filter, ND, provides variable attenuation of the pulse train. The X and Y scan mirrors then reflect the beam into the microscope for raster scanning of the sample during imaging. Two-photon excitation with a few picojoules of pulse energy requires the combination

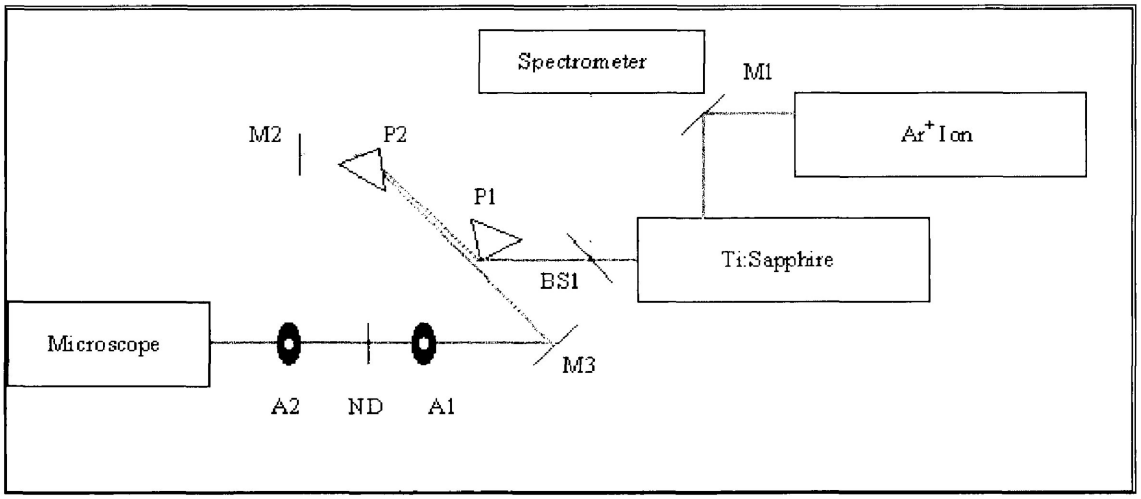


Figure 1.5: Schematic of two-photon microscope laser stage.

of ultrashort pulses and tight focusing of the pulses using an objective. Mode-locking can produce pulses as short as 20fs in this laser system but pulses in the 50-100fs range were normally used. As an example of an optical power density estimate at the sample, a 100pJ / 50fs pulse yields a peak power of ~2.0kW. If this pulse is focused to a ~0.3μm diameter spot, the power density reaches approximately 2 x 10^{12} W/cm^2 in the focal volume. By taking the product of the 90MHz pulse repetition rate and an average pulse length of 50fs, the fraction of real time that light is actually interacting with the sample is determined to be approximately ~10^{-5} of real time. The technique results in a self-confocalizing effect (light arriving at the detector is obtained only from the focal volume at the sample) without the need for confocal apertures. That is, excitation is automatically confined to the plane of focus since the intensity required for the nonlinear two-photon excitation of the sample, is only met there. In spite of the very high peak power density, the time averaged power delivered to the sample is lower than from CW confocal lasers.

Figure 1.6 shows representative spectra for the KMLabs Ti:Sapphire laser used in these studies. The central wavelengths range from approximately 740 – 830nm. If the mode-locked bandwidth is reduced by blocking part of the dispersed internal cavity beam, the pulse length increases inversely.

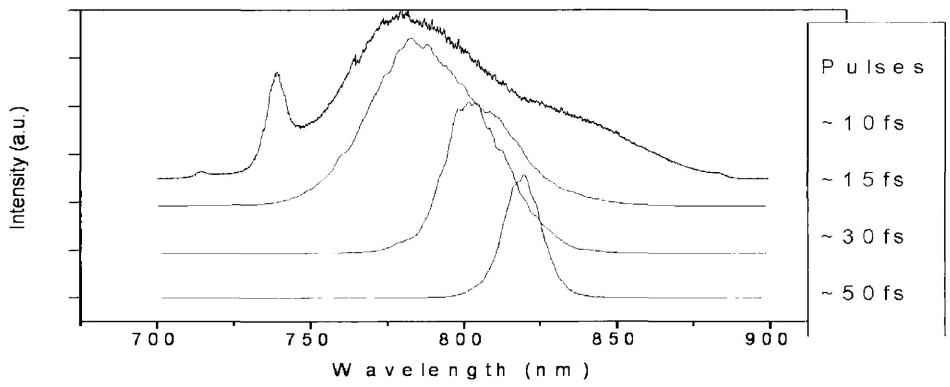


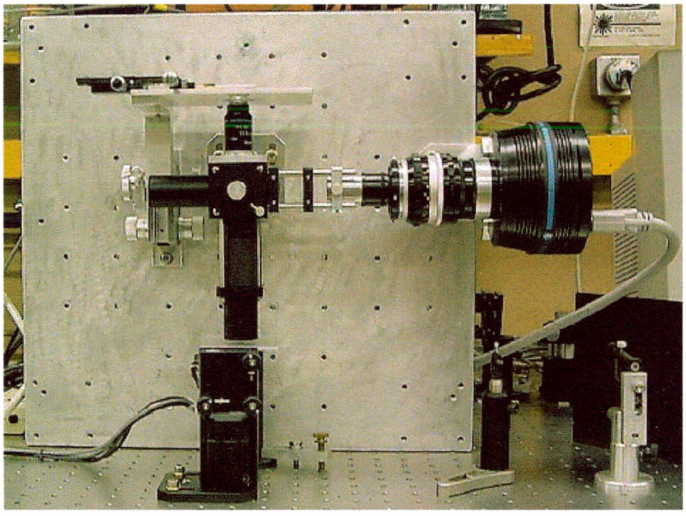
Figure 1.6: Representative spectra of the KMLabs Ti:Sapphire pulse laser.

Laser Microscope Details:

Upon exiting the secondary pre-chirp stage that follows the Ti:Sapphire laser, the beam enters the microscope optics which are arranged in an inverted microscope configuration. Shortly before beginning this work, the scanning three-dimensional microscope system shown in Figure 1.7, had been designed and partially constructed. Two Cambridge Technologies Inc. model 6210 analog galvanometer mirror units used as X and Y-axis scanners are held in a model 6102103R XY orthogonal jig. Each scan unit is driven by a model 67821 preamp-controller and two Power One Inc. model HD-28-4A high current slaves for bipolar output. The preamp and slave units, including interconnect cabling, were assembled in-house[19].

To attain uniform illumination over the sample area and maximum resolving power, the laser beam must fill the back aperture of the microscope objective at all times. To achieve this, the scan mirror assembly must act as a rotor object, and be imaged at the back aperture entrance to the objective where it also acts as a non-translating scan source. Simultaneously, the input parallel beam of light from the laser must follow the usual path of an inverted microscope. This was accomplished following the method of Kino (see pg. 76 of reference 4) by placing a lens pair in the beam path at conjugate focal positions as shown in the schematic. In addition, a beam expansion factor of $f_2 / f_1 \sim 1.5$ was incorporated to expand the beam diameter enough to fill the back aperture of the objective.

The fluorescence signal from the sample plane (red shifted from one-half the pump laser wavelength) is separated from the laser fundamental by a dichroic mirror located on the diagonal of the objective box as shown in the operational schematic. A band-pass, (370 – 570nm) 'Russian Blue', filter is then used to reject excitation light near 800nm that may have scattered into the output path leading to the imaging camera. The image light is then focused onto a cooled (-20C) Apogee Inc. KX85 camera CCD chip using a Nikon SLR lens. For locating areas of interest in the sample, manual operation of X, Y and Z micrometer translators is incorporated into the microscope stage.



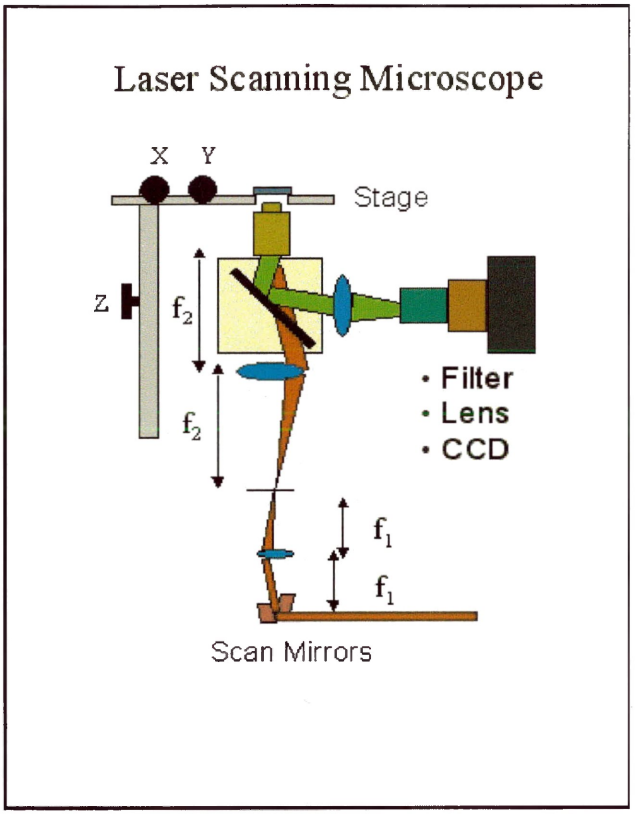


Figure 1.7: Two-dimensional, two-photon microscope optics apparatus (left), operational schematic (right).

1.2 XY Scan Computer Control:

Initially, analog function generators were used to scan the laser beam in the XY plane. However, to eventually treat certain areas of a sample while leaving other regions untouched (especially over a period of time or non-standard shapes), general computer control of the scan area(s) is required. In order to achieve this level of control over the scanning of the spot, the analog drivers were replaced in favour of an A/D/A interface card.

1.2.1 Digital Interface

The National Instruments (NI) multifunction data acquisition (DAQ) card, model NI PCI-6024E, was chosen to replace the existing analog control to the scan mirrors. The NI 6024E connects to the computer via a standard PCI slot and has two analog (D/A) output channels. It is capable of analog output in the range between ±10V with 12 bit resolution and a maximum

output rate of 10kS/s[20]. To connect the D/A outputs to the scan mirror current amplifiers, an NI CB-68LP 68-pin digital and trigger I/O terminal block was used. The X-axis scan is controlled via the card's first analog output, AO 0, likewise, the Y-axis scan connects to the second output, AO 1.

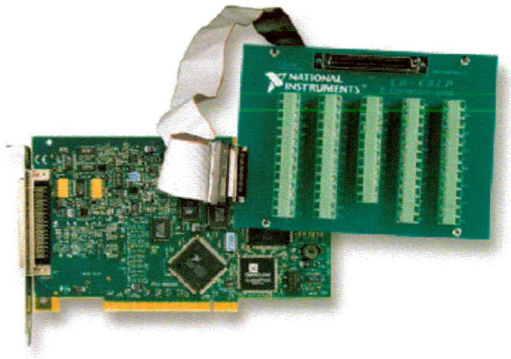


Figure 1.8: National instruments PCI-6024E DAQ card and CB-68LP terminal block.

1.2.2 National Instruments Function Calls

Software routines for the PCI-6024E are provided for many development languages including Microsoft's Visual Basic. This language was chosen because of its overall flexibility, ease of use and because each piece of hardware controlled by the interface application, had command references for Visual Basic. The NI routines are contained in the National Instruments library file *nidaq32.dll*. In this section, a description of the National Instruments output subroutines utilized (subroutine names in ***bold-italics***) and their respective arguments are provided[21].

AO_VWrite(deviceNumber, chan, voltage)

Write an analog voltage to an output channel. The routine first converts the voltage into a binary number that can be written by the card.

deviceNumber	16 bit integer	The device number to write to.
chan	16 bit integer	The analog output channel number to write to (0 or 1).
voltage	64 bit float	The analog output voltage to write.

WFM_Scale(deviceNumber, chan, count, gain, voltArray, binArray)

Converts an array of analog voltages into an array of binary values that the DAQ card uses when producing the output voltages.

deviceNumber	16 bit integer	The device number to write to.
chan	16 bit integer	The analog output channel number to write to (0 or 1).
count	32 bit unsigned integer	The number of voltages in the array.
gain	64 bit float	Multiplied to the voltage as translation is performed.
voltArray	array of 64 bit float	The analog output voltages to write.
binArray	array of 16 bit integer	The binary values returned by the function.

WFM_Op(deviceNumber, numChans, chanVectt, buffer, count, iterations, rate)

Writes a voltage waveform stored in binary value buffer arrays to analog output channels.

deviceNumber	16 bit integer	The device number to write to.
numChans	16 bit integer	The number of analog output channels.
chanVect	array of 16 bit integer	The array of channel numbers to write to.
buffer	array of 16 bit integer	The binary values the card is to write.
count	32 bit unsigned integer	The number of values in the array.
gain	64 bit float	Multiplied to the voltage as translation is performed.
iterations	32 bit unsigned integer	The number of times the array is written.
rate	64 bit float	The number of values to write in one second.

The software for the XY scan is a recent addition to the system hardware interface and is still undergoing further development and optimization. Initially the function AO_VWrite was employed for both X and Y scans, writing immediate voltage values to the card in a standard iterative loop. While this was successful in achieving a working raster scan, there were frequent 'noise spikes' in the scan due to operating system overhead interruptions and/or recalculations

during the scan. This would lead to noticeable image artefacts due to non-uniform illumination at the sample plane.

The problem was corrected by using the NI built-in buffered waveform functions WFM_Scale and WFM_Op. Prior to scanning, the whole range of X voltages are calculated and stored in an array. This array is passed to the WFM_Scale function producing the array of binary values the card uses to output voltage, and thus is stored for future identical scans. The Y scan consists of a series of steps performed at either end of the fore and back X scans and is still performed using AO_VWrite in an iterative loop. The function WFM_Op is called using the binary value array generated earlier. This adjustment resulted in a clean raster scan in X, with slight end-of-scan hot spots due to the momentary delay of the Y voltage write. Currently the user interface, depicted in Figure 1.9, allows for modification of the maximum, minimum, and increment voltages (for both X and Y), a delay between Y increments, and the number of raster scans performed.

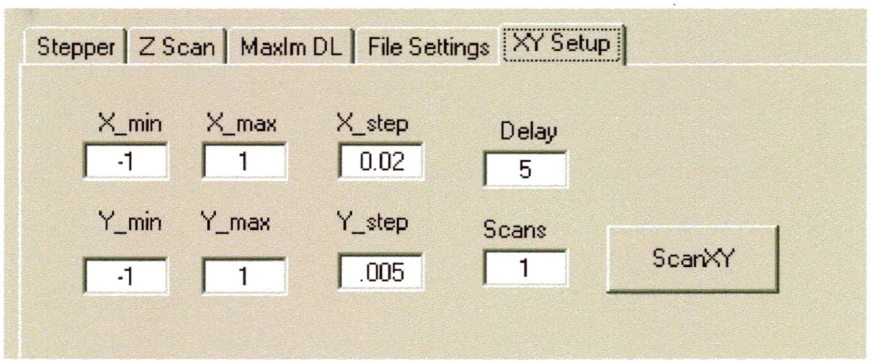


Figure 1.9: XY scan user interface.

1.3 Three-Dimensional Sectioning

The three-dimensional sectioning capability provided by the manual micrometer translators is limited in its usefulness. Accurately determining and relocating multiple positions at the micron level is difficult and very time consuming. In order to quickly and accurately image

samples three-dimensionally, over an extended period of time, an automated sectioning system is required.

1.3.1 Zaber Linear Actuator

The Zaber T-LA28A linear stepping actuator was the device chosen to perform the third (Z) dimension of sectioning. The actuator, with embedded microprocessor and ROM, is controlled by transmitting ASCII (American Standard Code for Information Interchange) byte commands to it via an RS-232 (Comm.) computer port. The T-LA28A has a maximum extension of 28mm with a 0.1μm resolution. The unit has a maximum thrust of 50N and can be controlled by either the RS-232 port, or a manual control knob located at the base of the device. Multiple instances of the device can be connected together in a daisy chain and accessed individually in software by their respective unit numbers. The Zaber TSB28-M translation stage is 4in x 3in x1in and was chosen to provide the Z-platform to which the full microscope sample stage is connected.

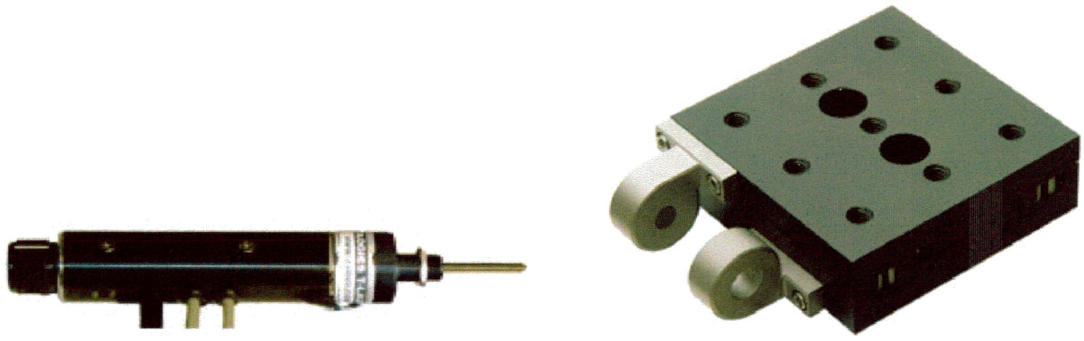


Figure 1.10: Zaber T-LA28A linear actuator (left), TSB28-M translation stage (right).

1.3.2 Zaber Visual Basic Class

A command to the T-LA28A unit is a 6-byte ASCII string consisting of a 1-byte unit number, 1-byte command number and 4-byte data string (least significant byte first). While this may be sufficient for many applications, working solely with command codes becomes

impractical when designing a complex graphical user interface that is to provide future development opportunities[22].

Visual Basic was chosen because it allows the creation of classes. A Visual Basic class is a custom set of routines and properties for a user-designed software object. Therefore, using this approach, it was possible to create procedures for the actuator in a self-contained Zaber T-LA28A class. This class can then be imported into any number of Visual Basic projects and has the advantage of maintaining the hardware control routines and the graphical interface as separate entities.

Provided below is a list of the more commonly used properties and routines available to a Zaber T-LA28A object. Function and subroutine names will be in ***bold-italics*** and followed by their respective descriptions. If a routine requires arguments, the argument names, data types and description will be given immediately following the routine description For the full source code, refer to Appendix A.

Properties of the T-LA28A Object:

Name	Data Type	Description
Unit	Byte	unit number of the actuator
Position	Double	extension in microns of the actuator
Comm	Object	comm object the actuator is connected to

Function move(New_Position, Relative_Change) returns Integer

Move is used to change the extension length of the actuator. Returns -1 if the desired position is out of the range. Returns 0 if the new position is within range.

New_Position	Double	new desired extension length of actuator in microns
Relative_Change	Boolean	indicates if the change to be made is relative to the current position

Function positionToBytes(New_Position, data_array) returns Variant

Converts the given position from a double precision number into a four-byte array of microsteps (whole number) to be sent over the Comm. port. Due to rounding, the uncertainty in the position is +/- 0.5 microsteps or 0.05 microns. The function then returns this array.

New_Position	Double	the double precision number to be converted to be converted to microsteps
data_array	Byte	the array to store the microstep bytes in

Function bytesToPosition(position_bytes) returns Double

Converts the data bytes received over the Comm. port into a double precision position value in microns. Due to rounding the uncertainty in the position is +/- 0.5 microsteps or 0.05 microns.

position_bytes	Byte	four byte microstep array to be converted into microns

Subroutine sendCommand(act_unit, command, data)

Sends a specific command with data to a specified actuator via the communications port.

act_unit	Byte	the actuator unit number the command is to be sent to
command	Byte	the reference number or the zaber command to initiate
data	Byte	four byte array containing binary data for the command

Subroutine homeUnit()

Sends the "home" command to the actuator, which returns it to a position of zero microns / microsteps.

Subroutine renumberUnits()

Sends the "renumber" command to all the actuators, which resets the unit numbers of each to their position in the chain from the communications port.

Subroutine returnPosition()

Sends the "return position" command to the actuator, calling the routine which causes it to return its current extensions (in microsteps) to the system.

1.3.3 MaxIm DL Function Calls

If automated three-dimensional images are to be taken, the software controlling the linear actuator must also be able to activate a CCD camera exposure. All CCD images taken in this work were captured using Diffraction Limited's MaxIm DL software. The program allows access to a collection of its functions through ActiveX scripting. ActiveX objects provide properties and methods, which can be accessed by an external application such as Visual Basic. Unlike the Zaber class object, the MaxIm DL application must be installed on the system for ActiveX scripting to function because it is not compiled into the calling application. Provided below is a list of the MaxIm DL ActiveX function calls utilized in the 3-D imaging application. The conventions of the above section are used[23].

Properties of the CCD Camera Object:

Name	Data Type	Description
ImageReady	Short	Indicates if the image buffer is ready following an exposure
LinkEnabled	Boolean	Used to enable or disable the link between the CCD and the calling application

Function Expose(Duration, Light [, Filter]) returns Boolean

Starts and CCD exposure and returns true if successful. ImageReady property can be used to determine when the exposure is completed.

Duration	Double	Duration of exposure in seconds
Light	Short	1 for light frame, 0 for dark frame (ignored if no shutter)
Filter	Short	(optional) Index of filter to use; first filter is 0 (ignored if no filter wheel)

Function SaveImage(FilePath) returns Boolean

Saves the last image in the CCD buffer to the specified path using the Flexible Image Transport System (FITS) format.

FilePath	String	path to save file in

The software result of the above work is presented in Figure 1.11. A graphical user interface allowing for three-dimensional sectioning was produced. It provides a means of connecting to the unit, homing, moving the actuator in arbitrary or fixed increments, and automated Z-scan control.

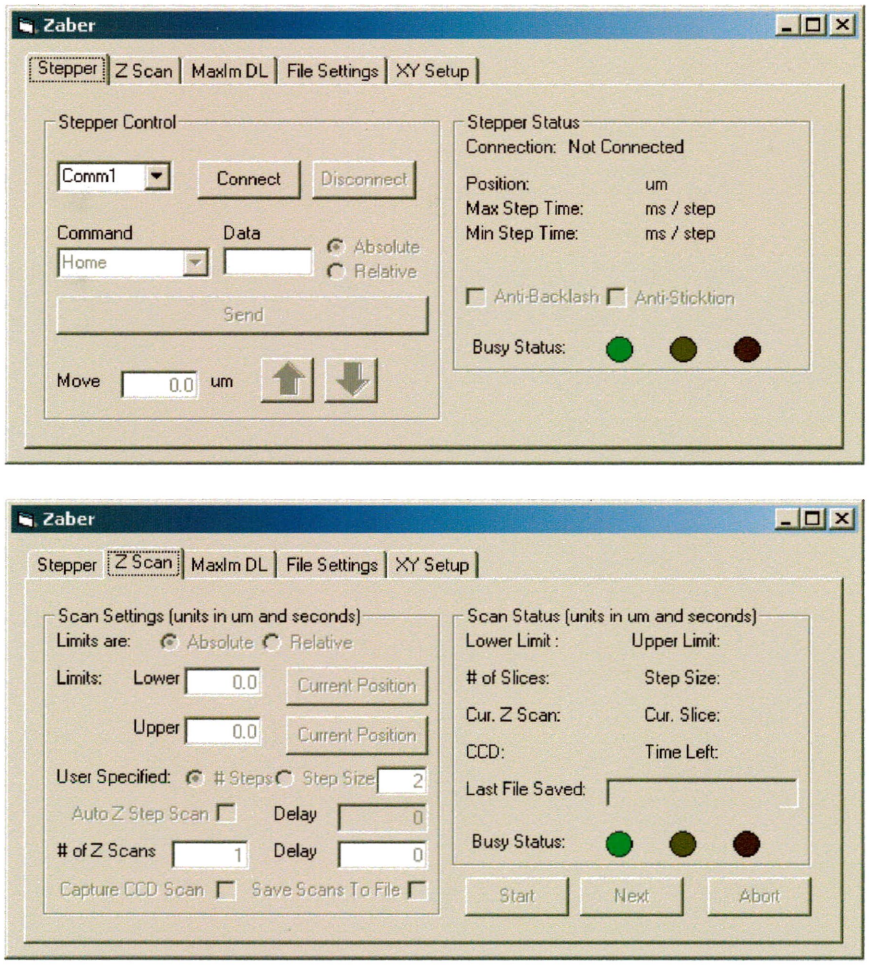


Figure 1.11: 3-D imaging control graphical interfaces. Manual actuator control (top) automated sectioning control (bottom).

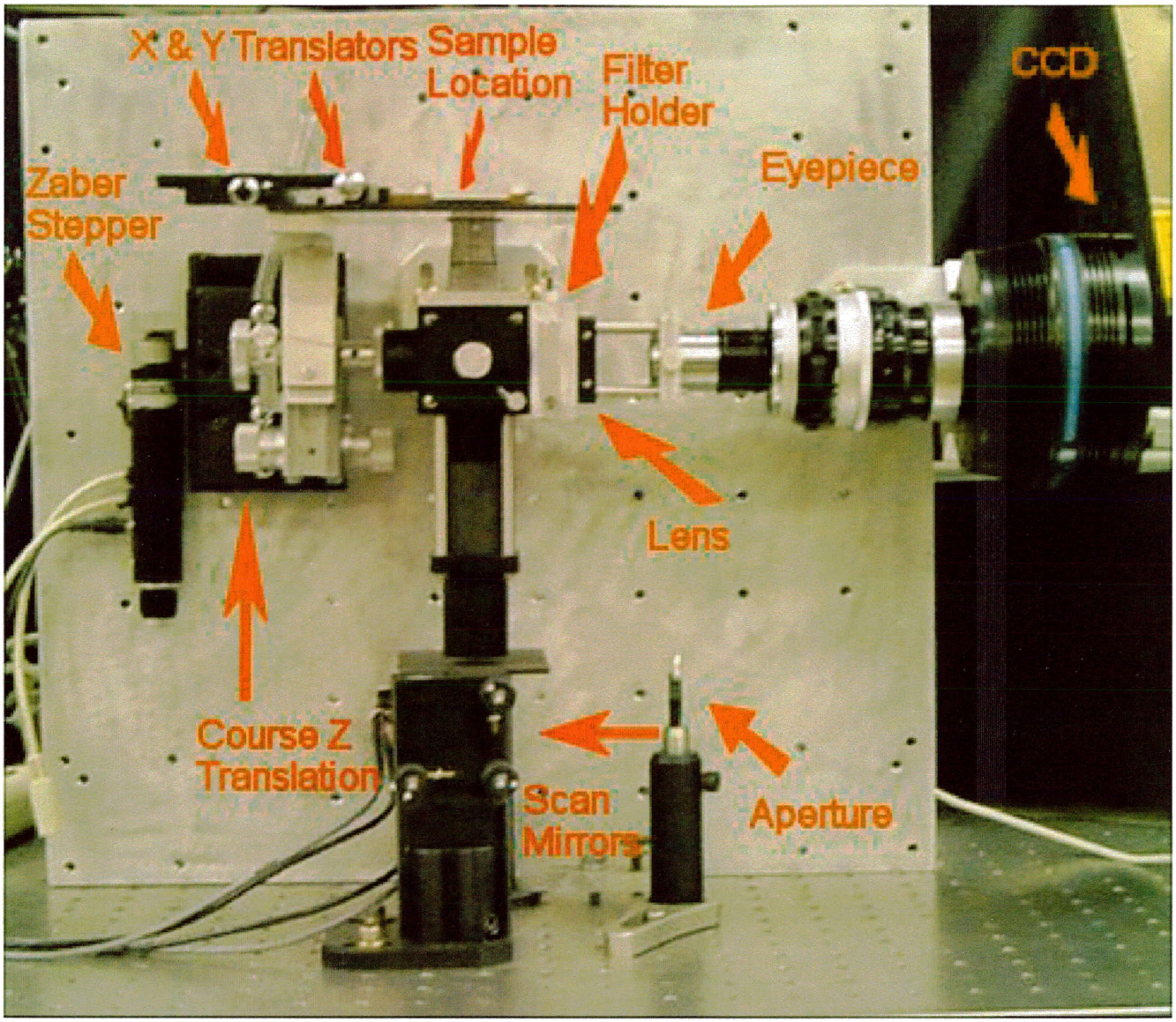


Figure 1.12 Complete 3-D Microscope with Zaber z-Stage

The hardware result is presented in Figure 1.12. The Zaber Z-stepper is shown connected to left side of the manual translation stage and the spring attachment reduces the load on it. The CCD camera is the same unit used by the Nikon E400, but includes a SLR 35mm focusing lens to take infinite conjugate light from the eyepiece to the CCD chip. A sliding filter holder on the camera side of the dichroic mirror cube, which moves perpendicular to the page, can be used to insert an appropriate band pass emission filter into the imaging pathway.

Chapter 2

Two-Dimensional Tissue and Live Cell Imaging

As previously mentioned, an objective of research in this group is to promote the use of light in the detection and treatment of cancer. While PDT treatment techniques are clinically used quite extensively in Australia, for example, where there are high incidences of skin cancer, the method is largely in the research phase in most other regions due to the overwhelming use of radiation and chemotherapy treatments. For PDT treatment of cancer to become more commonly used, further demonstrations of its relative effectiveness must be realized and live cell samples *in vitro* are the most convenient working platform for studies where lab animals are unavailable. Prior to clinical trials, the efficacy of laboratory-based treatment must first be demonstrated and high quality near real-time imaging is a useful tool in this pursuit. To determine the effectiveness of lab experiments, images of the sample must meet these criteria:

1) Images must display a distinction between normal and cancerous regions (if both are present) to permit targeting of a treatment.

2) The images must have good contrast, be rapidly obtainable, and be available for an extended period of time (several hours) to effectively monitor cell viability and response to treatment methods.

3) The large data sets accumulated during the monitoring process must be in a format that allows rapid playback and easy visualization for proper interpretation.

This chapter describes the steps taken in developing a system that meets the above criteria. Section 2.1 describes the microscope setup used to carry out this investigation. 2.2 presents results pertaining to the first criterion using fixed samples at both the tissue and cellular levels. 2.3 addresses criteria in 2) via imaging technique and sample incubation. Finally, section 2.4 deals with software developed to manipulate results to meet criterion 3.

2.1 Nikon E400 Fluorescence Imaging System

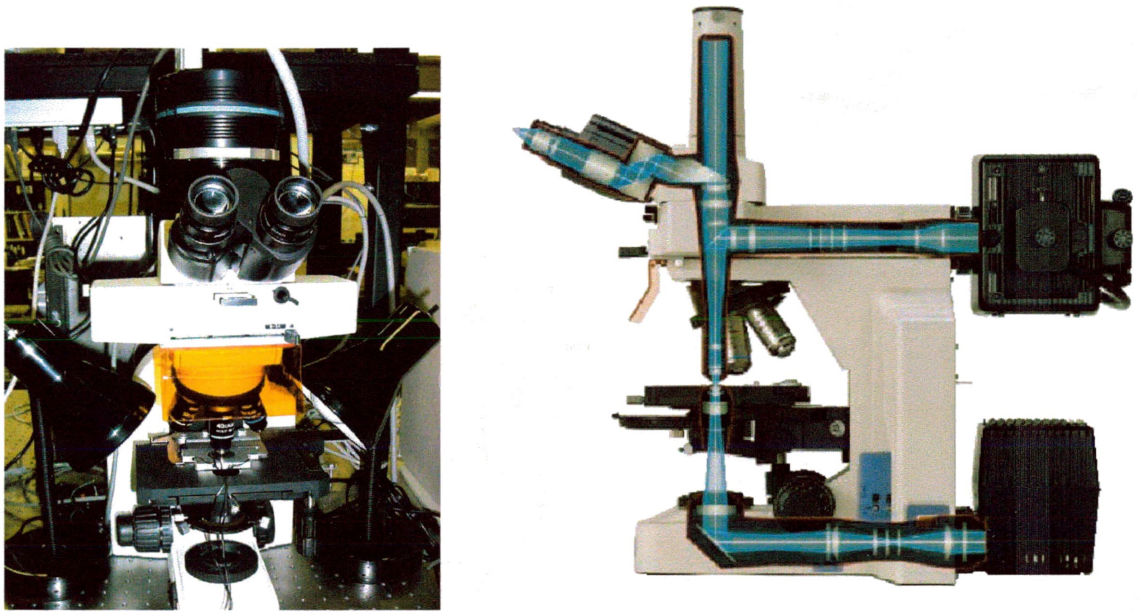


Figure 2.1: Nikon Eclipse E400 Imaging System experimental apparatus (left), optics schematic (right). Normally only the upper or lower optical source lamp is used at one time.

An imaging system consisting of a Nikon Eclipse E400 Fluorescence Microscope in conjunction with an Apogee KX85 thermoelectrically cooled CCD camera was used for most of the 2-D investigations. The Nikon E400 employs Nikon's CFI60 optics system and uses a 6V, 100W halogen lamp for illumination. The fluorescence attachment head contains slots for four filter blocks for epifluorescence illumination and a 25mm diameter filter holder. Focusing is achieved through a course focus knob of 12.7mm per rotation and a fine focus of 0.1mm per rotation. The minimum possible Z-axis focal adjustment is ~1μm[24].

The right image of Figure 2.1 presents a ray optics schematic of the E400. During the course of these investigations, only the upper lamp was used for fluorescence excitation. Light emerges from the halogen lamp and passes through the excitation filter of the filter cube. For a schematic of the filter cube see Figure 2.2. The filtered light reflects off the dichromatic mirror and is focused onto the sample by the objective. The fluorescence is then collected by the objective

and passes through the mirror and the emission filter. Finally the rays pass through the optional filter and exit into the imaging optics (eyepiece or phototube and CCD)[25].

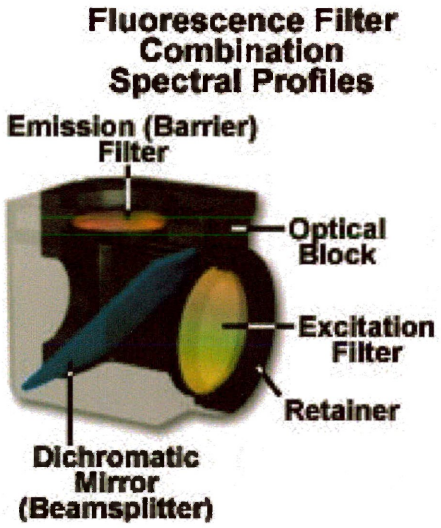


Figure 2.2: Schematic of filter cube.

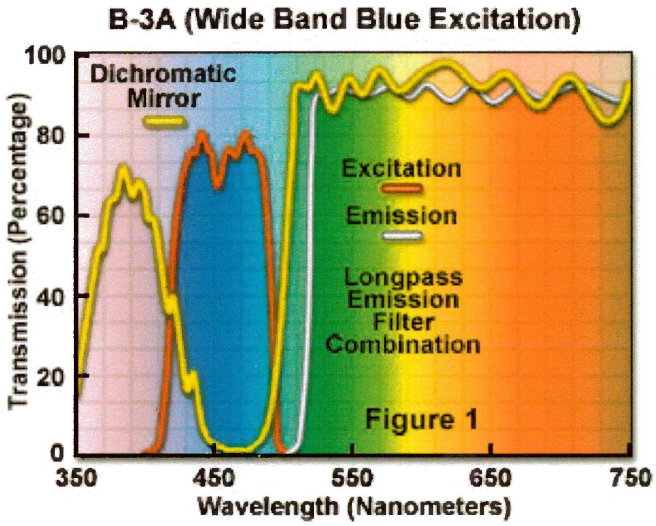


Figure 2.3: Nikon B-3A filter cube spectral curves.

When the E400 was used for fluorescence imaging of tagged samples, the Nikon B-3A filter cube was utilized. This filter cube contains a 420-490nm band-pass excitation filter, a long-pass dichromatic mirror with a cut-on wavelength of 505nm, and a longpass barrier filter with a cut on wavelength of 520nm. Refer to Figure 2.3 for full spectral curves[26].

The objectives used with the E400 in these studies were Nikon CFI Achromat Flatfield objectives. Table 2-1 gives the properties of the individual objectives. Listed are their respective product numbers, magnifications, numerical apertures, working distances[27], and XY resolutions, calibrated as CCD pixel resolutions. The XY camera pixel resolutions represent the width and height of an individual pixel in a captured image. They were determined by measuring a known length on an objective micrometer and then dividing that length by the number of pixels illuminated.

Nikon CFI Achromat Flatfield Microscope Objectives				
Nikon Product No.	Magnification	N.A.	W.D.	Pixel Resolution [µm / pixel]
93161	10x	0.25	6.10	1.095
93163	40x	0.65	0.65	0.2538
93159	60x	0.8	0.25	0.1692
93164	(oil immersion) 100x	1.25	0.18	0.1095

Table 2-1: List of objectives used in the Nikon E400

The second component of the imaging system is the afore-mentioned Apogee KX85 CCD camera. The camera utilizes a shutterless Sony ICX085 interline CCD chip with a resolution of up to 1300 pixels horizontal and 1030 pixels vertical with a grey-scale colour depth of 12 bits per pixel[28]. The quantum efficiency of the chip is represented by the blue curve in Figure 2.4[29]. Prior to acquiring images, the CCD is forced air thermoelectrically cooled to -20°C. This camera was selected because of its enhanced 'blue' sensitivity in the region

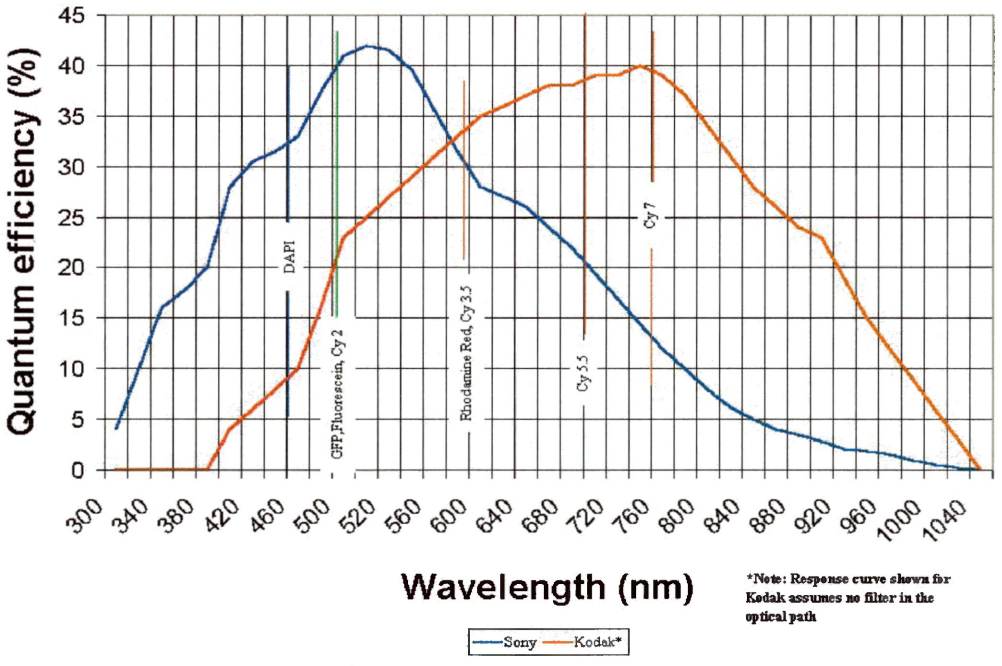


Figure 2.4: Quantum Efficiency of the Apogee KX85 CCD Camera (blue curve).

between 380nm and 520nm. This range corresponds to the wavelength range associated with two-photon excitation by the ~780nm Ti:Sapphire laser pulses and subsequent fluorescence that may be induced in various samples. Reduced sensitivity in the red also improves the signal-to-noise ratio where the principle optical noise is due to the laser fundamental near 780nm. The red curve represents the response of a more typical CCD detector.

2.2 Cancerous vs. Healthy Tissue

2.2.1 Tissue Samples

At the May 2001 CLEO conference, S.G Demos[30] of Lawrence Livermore presented data showing that in a freshly excised tissue sample, the cancerous tissue appears to have higher autofluorescence intensity in the near-infrared (NIR) than does healthy tissue surrounding it when illuminated by low power Helium-Neon (HeNe) light (See Figure 2.5). As minimal tagging is an aim of our study, a reproduction of the Demos result was attempted on smaller sections of frozen tissue taken as biopsy material.

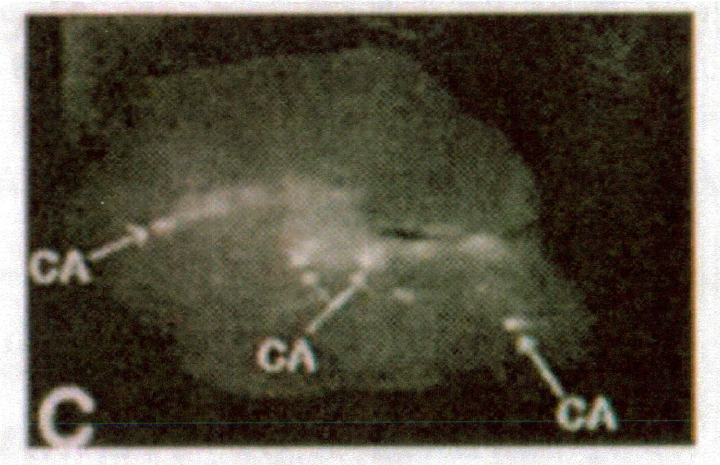


Figure 2.5: Results of NIR imaging of fresh breast tissue by Demos et al. at Lawrence Livermore. A line of cancerous tissue is clearly visible in this 4 cm x 5 cm x 1 cm thick tissue section.

Thin sections of freshly frozen breast tissue with intermingling normal and cancerous regions, were prepared for us by Ms. Marrisa Kubinec of the Northwestern Ontario Regional Cancer Centre. Four adjacent thin sections per patient biopsy were taken and fixed in ethanol. One section from each patient was stained with eosin, which turns normal breast tissue pink and haematoxylin, which turns cancerous tissue purple. One such stained section (See Figure 2.6 a) was imaged using a stereomicroscope with an overall magnification of ~40x and used as a reference for imaging the adjacent unstained sections. An unstained section of the same tissue sample was then autofluorescence imaged using the Nikon E400, its blue excitation cube, and the 10x objective followed by a 10x eyepiece. Overall image field-of-view on the monitor is the same as that of the eyepiece viewing path, even though there is no eyepiece element in front of the CCD camera, because both display the same image of the objective. Before being imaged on the Apogee CCD the light was passed through a 700nm cut on long pass filter so only the NIR autofluorescence was collected. The images produced had a narrow field-of-view, so nine images were taken in a checkerboard pattern and stitched together in software to form a larger composite image. NIR images of a representative portion of a tissue section from patient R575, are presented in Figure 2.6 along with its stained reference. Image A is the stereo microscopic image of the stained section. The circled areas are a darker purple than the surrounding tissue and have been verified as cancerous. Image B is the result of our stitched auto-fluorescence images. Image C is a pseudo coloured version of image B, providing better contrast. As can be seen, the higher intensity areas of the auto-fluorescence correspond well to the cancerous areas of the stained section. Thus, the method verified for tissue thin sections the result that Demos et al. had presented for macroscopic tissue resections[31].

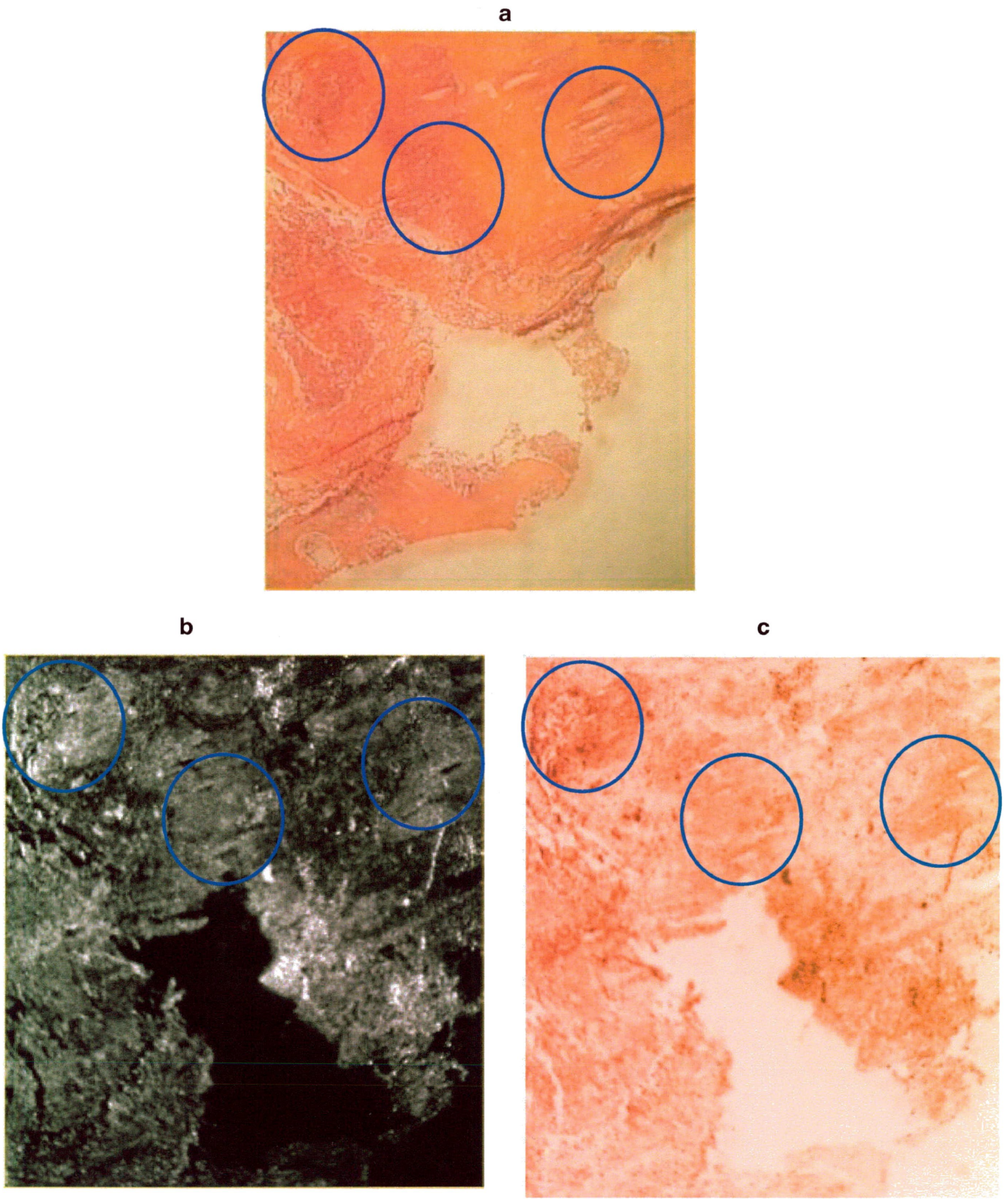


Figure 2.6: Tissue Thin Sections (circled areas are cancerous). Stained tissue section from stereomicroscope (a), autofluorescence image of unstained tissue section (b), pseudo coloured image(c).

2.2.2 Cellular Level Auto-fluorescence

The intensity variation between cancerous and healthy tissue prompted an investigation at higher magnification in an attempt to determine the cause of the difference. Using a 40x objective, individual cells or groups of cells in the cancerous region of the tissue appeared brighter and round in shape while adjacent normal tissue cells were smoothly interconnected into a homogeneous mass. This suggested that the cause of the difference might be cell density and/or morphology related. Therefore, a decision to investigate fixed cultured cells was made since a 1 cm^2 plating of cells on a cover slip can provide many configurations of the cell cycle at different locations on the cover slip.

Fixed, untagged human skin cells (cell line HSF-55) and fixed tagged breast cancer cells (cell line MCF-7) were obtained from Dr. John Th'ng at the Northwestern Ontario Regional Cancer Centre. Untagged breast cells were not available at the time. The same procedure as in 2.2.1 was employed.

Images of non-cancerous skin cells are presented in Figure 2.7 a. Cells in different phases of the cell cycle were located and it was discovered that the autofluorescence intensity was greatest in the nearly spherical cells near to, or undergoing mitosis (division). As is well known, cells roll-up during the division phase[32,33,34]. Since cancer cells divide more rapidly than normal cells in living tissue, proportionately more of them will be in the brightly scattering rolled-up geometry. Similarly, the brightest tagged breast cancer cells are those undergoing mitosis, although, as Figure 2.7 b shows, the cells are more spherical on average than the non-cancerous skin cells. Cell density also appears to be greater in regions containing many cancer cells and this will also contribute to an intensity increase. Visible spectroscopic measurements were separately made on sample regions that were normal or cancerous, but aside from the intensity differences noted, there was little change in the spectral content. Thus, it would appear that the main difference in the autofluorescence intensity observed between normal and cancerous tissue regions, is due to morphology and in particular, the faction of cells undergoing mitosis[35].

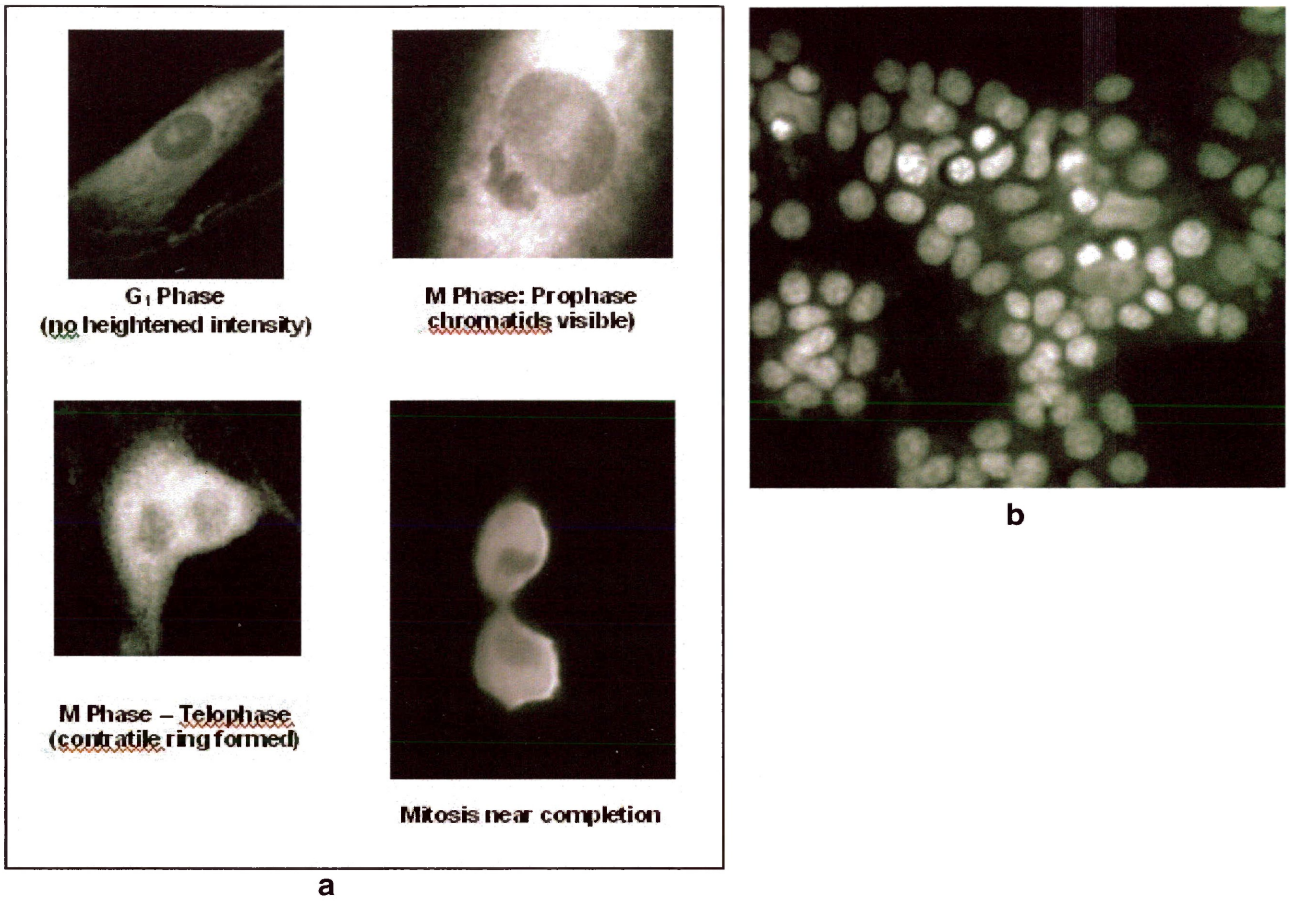


Figure 2.7: Cellular level fluorescence. Autofluorescence of untagged skin cells in different stages of mitosis (a), fluorescence image of tagged breast cancer cell line (b).

2.3 Live Cell Viability Studies and Imaging Techniques

2.3.1 Oblique Illumination

In the initial cell autofluorescence investigations, fixed cells were used. However, to provide further support for the intensity variation during cell-cycling model, it was thought that live cell samples would yield a better view of the dynamics of the effect. Live cultured untagged human skin cells (cell line HSF-55) were obtained from Dr. Sylvie Landry at the Northwestern Ontario Regional Cancer Centre. The cell-line is cultured in culture flasks for weeks at a time and when samples are required, they are 'plated' onto microscope cover slips inserted into the flasks where the cells can attach to the surface over a period of ~ 24 hours. In order to maintain live cell viability after the cells are removed from the culture flask, a microscope stage reservoir

is required that can maintain the necessary growth medium. Various reservoir designs were investigated. Initial reservoirs for growth medium were created on microscope slides by layering scotch tape around a square with an area slightly less than that of the cover slip and greasing the interior walls of the square to prevent water leakage. Each reservoir was then filled with growth medium and the cover slip and sample cells were placed over it. Nail polish was used to secure and seal the cover slip to the microscope slide's surface.

Standard epifluorescence imaging of these untagged cells was attempted, but an acceptable signal could not be detected, even with long exposure times at high excitation intensity. However, instead of using epi-illumination, it was discovered that white light illumination of the sample from the side of the Nikon E400 stage provided a good signal-to-noise image with very low excitation power density. This geometry is known as *oblique illumination microscopy* since the excitation light enters the sample at an oblique angle (See Figure 2.8). The signal appears to be a combination of reflected, refracted, and scattered light off cell components that are generally smaller than the wavelengths of the incident light. While irresolvable in the traditional sense, these structures can still offer significant group response to the electromagnetic fields of incident photons. Depending on the angle of incidence, oblique illumination images can resemble those of phase contrast, differential interference contrast, or dark field microscopy[36].

While performing oblique illumination investigations, a 25W incandescent light bulb was placed at the side of the Nikon E400. Later a pair of bulbs, one on each side of the stage, was used to improve image uniformity. The dichroic mirror and 505nm longpass filter of the BA-3 filter cube were part of the pick-up optical path, however the 700nm longpass filter was not used. The 40x objective generally provided the best magnification for the relatively large cells.

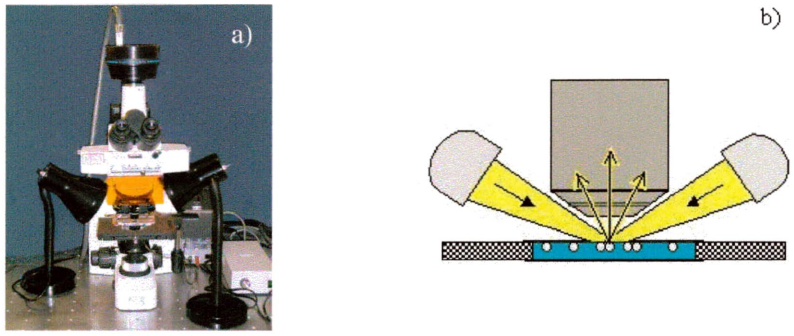


Figure 2.8: (a) Oblique Illumination apparatus and (b) lamp-sample ray schematic

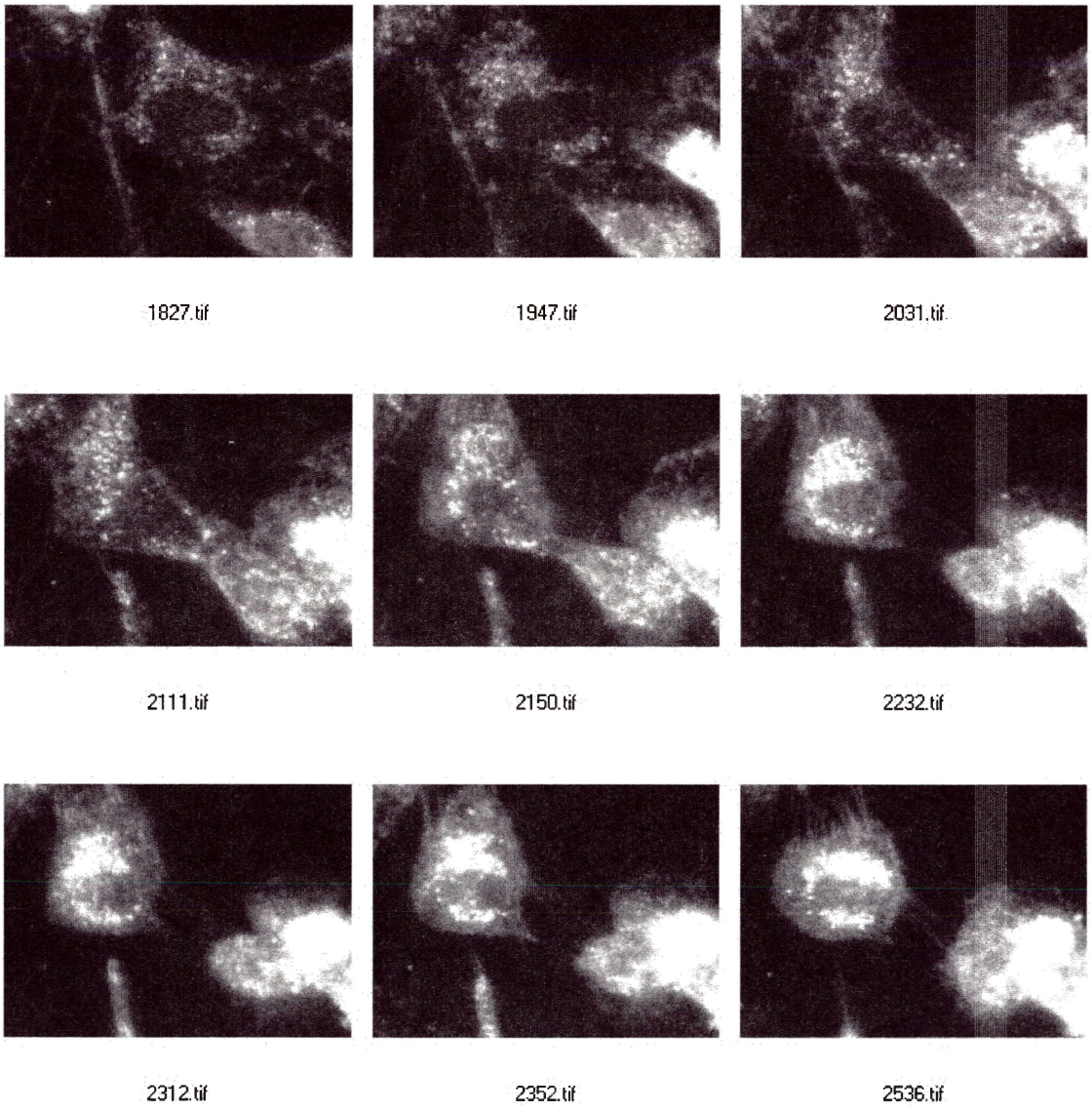


Figure 2.9: Time-lapse image sequence of live, untagged skin cells in scotch tape reservoir.

Typical reservoir sample results are presented in Figure 2.9. The complete data set is a time sequence of images taken over a period of 7 hours at intervals of 4 minutes. In the figure, the label of each frame indicates the time the image was taken in 24-hour time units relative to the starting time (refer to the video Chap_2_1.mp4 for the full animation). In this configuration, macromolecule and organelle movement both in and between cells, cell movement, and cell rounding up was observed. During the rounding process, the macromolecules and organelles collected near the nucleus and increased in number. Very good contrast was achieved but cells became progressively less mobile with time and ceased movement after an hour or so. An unexpected finding was that live cells autofluoresce less intensely than fixed cells.

2.3.2 Incubator

The short lifespan of the samples indicated a problem with the reservoir system being used. Thus, different well depths and mounting techniques were attempted, yielding no increase in cell lifetime. It became clear that the cells were dieing because the room was too cold and thus, a form of temperature control would be needed. A heated incubator was therefore designed to maintain sample temperature near 35°C and was constructed with the assistance of the department machinist Mr. George Anderson.

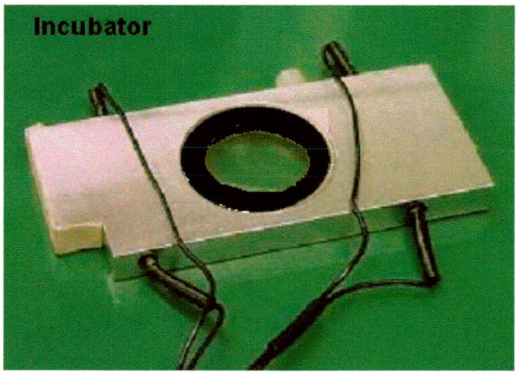


Figure 2.10: Variable Temperature Cell Incubator.

The incubator used in these investigations is pictured Figure 2.10. It consists of a Teflon chamber insert in an aluminum base. Nutrients are provided to the cells by storing growth medium between a bottom cover slip and the inverted sample cover slip mounted on the top of the chamber. Heat is provided to the sample via two 20Ω resistors located on either side of the chamber. Temperature control is provided by a variable voltage supply.

2.3.3 Oblique Illumination with Incubator

With the incubation system completed, another attempt was made to perform live cell imaging with the hope of witnessing a significant event such as mitosis or apoptosis (cell death). Live, cultured, untagged human skin cells (cell line HSF-55) were again obtained from Dr. Landry. By this time, live untagged breast cancer cells (cell line MCF-7) were also available. The cells were cultured using the same process as in section 2.3.1. A clean cover slip was placed on the bottom of the Teflon mounting chamber, the chamber was filled with growth medium and a cover slip with cells was attached to the top of the chamber. The chamber was then placed in the incubator with the resistors attached to the variable voltage power supply. After a few trials, the optimum operating conditions were determined. Many imaging sequences were taken using this oblique illumination technique.

The new incubator provided a significant improvement immediately. During the first run with cell temperature near 35°C, several cell divisions occurred in the field of view. Selected frames for one such cell are shown in Figure 2.11. The time sequence was taken over a period of 3 hours at intervals of 15 seconds. In the figure, the label on each frame indicates the frame number (refer to the video Chap_2_2.mp4 for the full animation). As can be seen, the macromolecules organelles collected around the nucleus just before and during mitosis, and the cells become much brighter. Also, a total detachment of one daughter cell from the other never occurs; rather a thin membrane filament remains as a link between the two cells as is

characteristic of skin. During this trial, a pseudo 3-D shadowing effect was observed, providing poor detail within the cytoplasm. By adjusting the position of the light source, contrast similar to the initial results was achieved in subsequent trials.

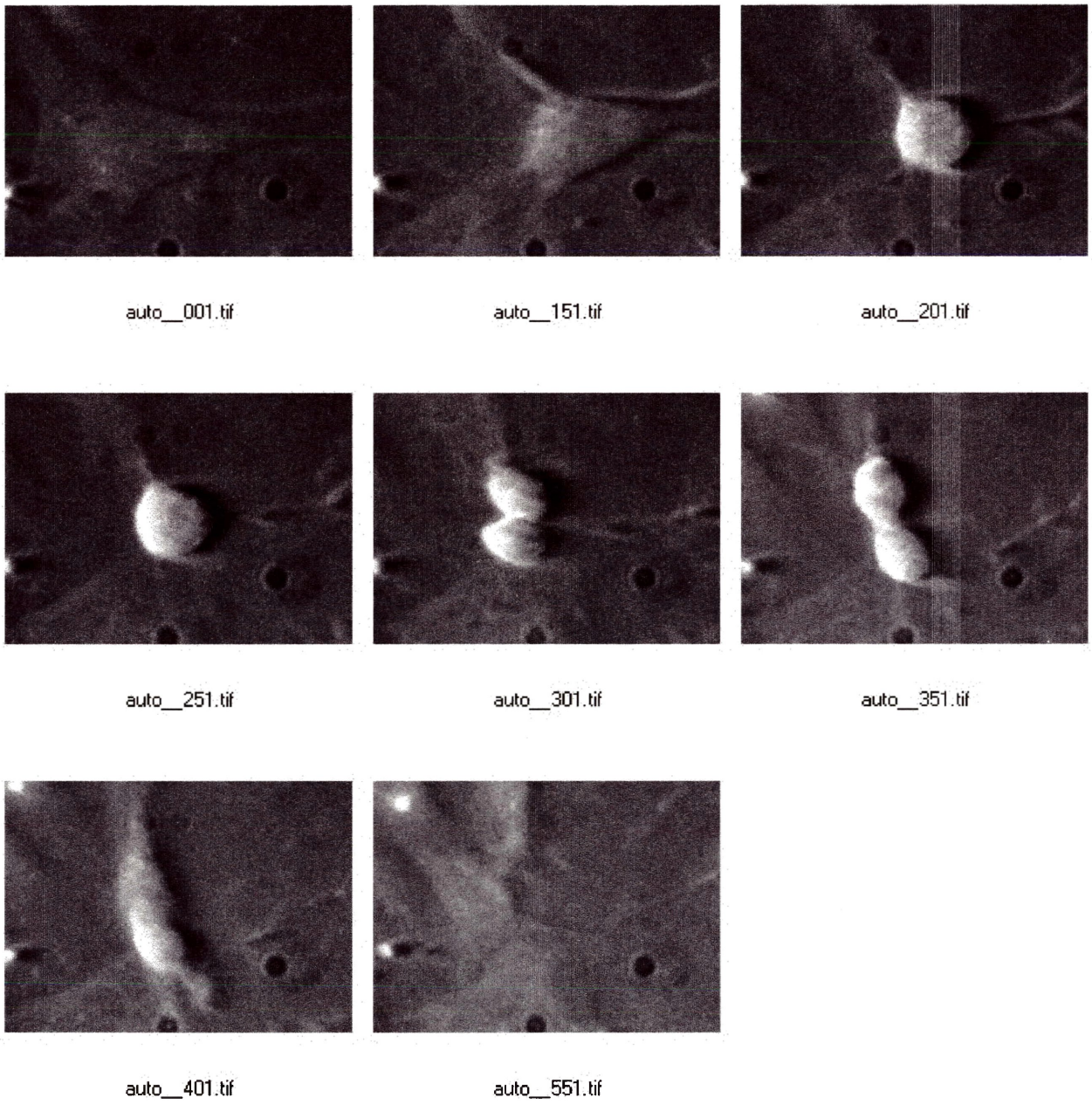


Figure 2.11: Untagged HSF-55 skin cell division using oblique illumination and an incubator.

Presented next in Figure 2.12, is a group of breast cancer cells, which were successfully imaged for over 16 hours at an interval of 15 seconds (refer to the video Chap_2_3.mov for the full animation). The label of each frame indicates the time at which the frame was taken. As can be seen, the cancer cells cluster in individual groups and are constantly rounded up, as was seen previously for tagged samples. This imaging sequence also verified the effectiveness of the incubator because the division did not occur until the latter end of the observation period.

The final result for this section is presented in Figure 2.13. It is a time sequence of a skin cell experiencing apoptosis (controlled cell death) taken over a period of 10 hours at intervals of 20 seconds. In the figure, the label on each frame indicates the frame number (refer to the video Chap_2_4.mov for the full animation). The data clearly shows the cell rounding up and breaking down into smaller parts, the typical effects of cell death. As can be seen, the macromolecules and organelles also gather around the nucleus during apoptosis.

2.3.4 Oblique Illumination Combined with Tagged Fluorescence

Fluorescence marking of live cell components using fluorophores attached to antibodies, nanoparticles, or altered protein sequences, as for example the insertion of green fluorescent protein (GFP) DNA into the genome, can yield detailed imaging information on the cell cycle behaviour of the specific cell component tagged. However, little is learned directly about other components in the cell because they are not visible. By contrast, the untagged oblique imaging method reveals motional information on many components in the cell but with little or only limited specificity as to the components being observed.

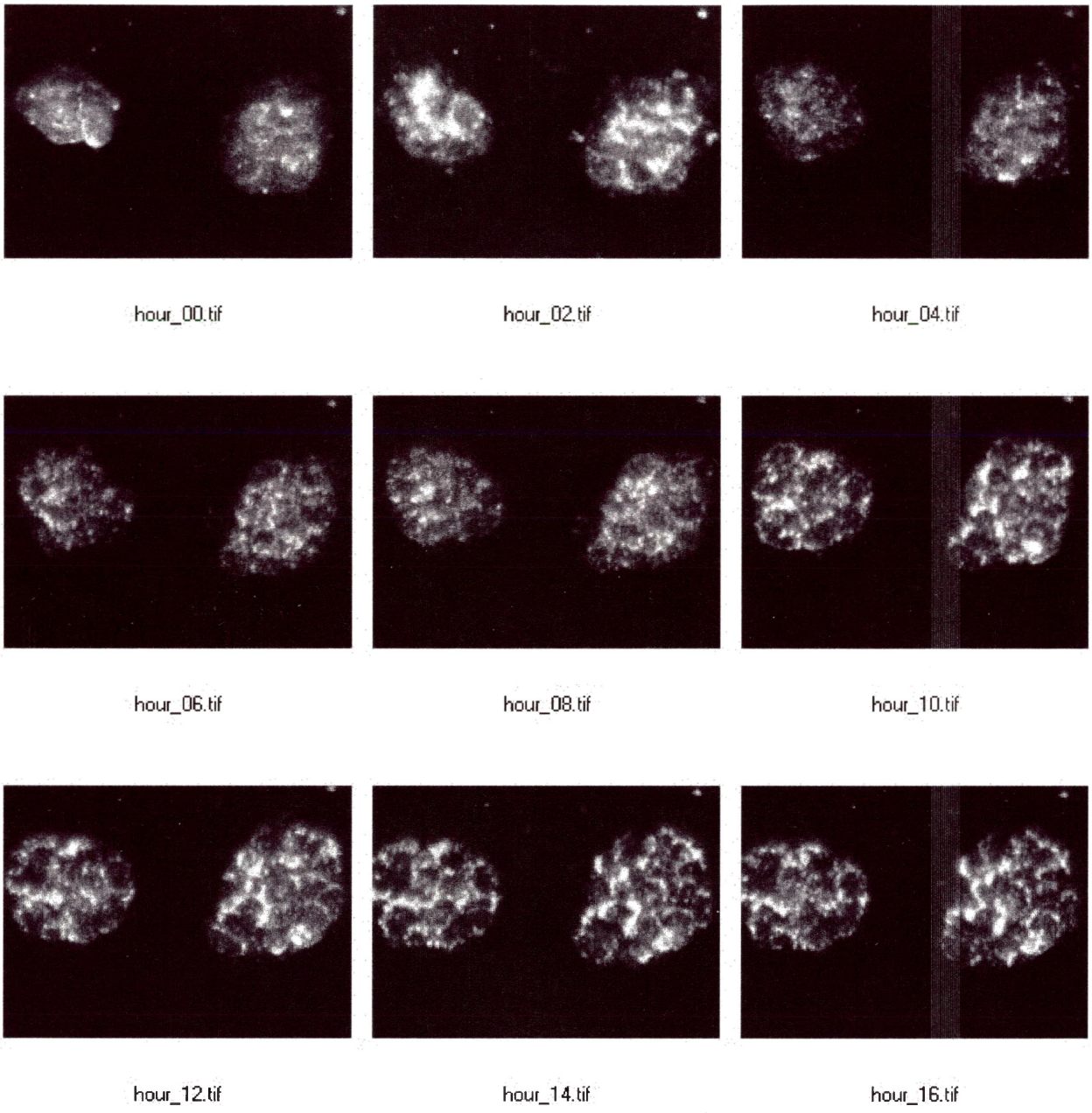


Figure 2.12: Untagged cancer cell division using oblique illumination.

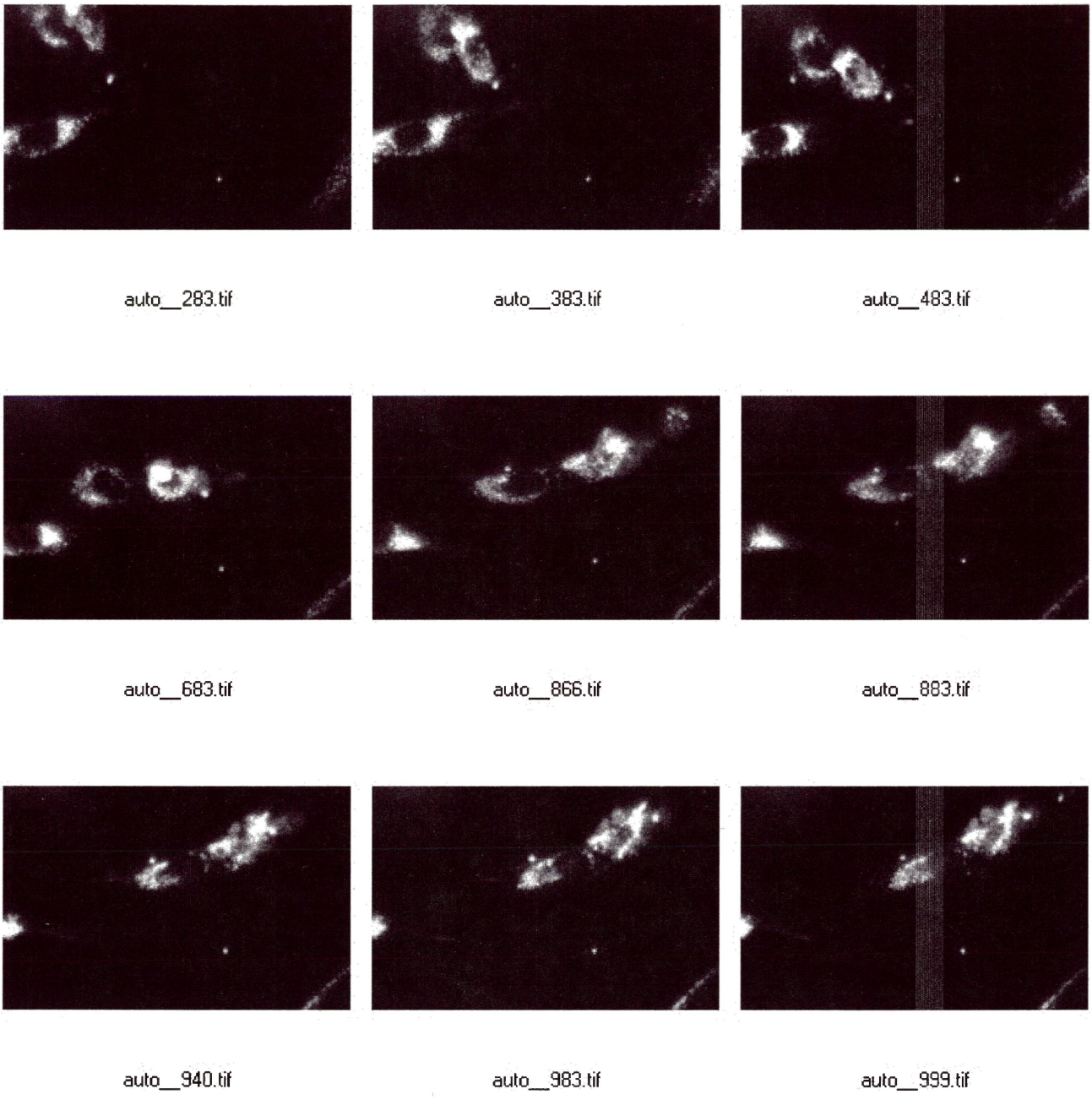


Figure 2.13: Skin cell apoptosis using oblique illumination.

It was quickly observed after a few untagged imaging runs that the oblique illumination method tended to do a rather poor job of illuminating components in the nuclear region of the cell. Since many biological studies tag nuclear components, it was an obvious question as to whether the two visualization methods could be combined simultaneously.

Commonly, to observe more than one process, two or more types of tagging must be affixed to appropriate cell components. If two tags are used for example, separate images must be taken using different filters followed by image combination in software. The disadvantage of this technique is that it takes more time to build one frame and requires a more complicated apparatus. If both the nuclear and cytoplasm detail could be imaged at the same time using a combination of oblique illumination and fluorescence tagging with lower overall toxicity to the cell, the combined methods might represent a useful new viewing method.

Live, cultured, breast cancer cells (cell line MCF-7) were again obtained from Dr. Sylvie Landry at the NWORCC. These cells had their Histone 1 material altered to express the Green Fluorescent Protein. Images were taken using the Nikon E400 with the epifluorescence lamp (set to the lowest power level for acceptable imaging) in conjunction with the incandescent oblique illumination lamp.

Figure 2.14 shows a time sequence for the dual illumination imaging setup (refer to the video Chap_2_5.mov for the full animation). The images were taken at intervals of 15 seconds. In the figure, the label on each frame indicates the frame number. The nuclear chromatin supporting Histone 1, the macromolecules and organelles in the cytoplasm, and the cell boundaries are all clearly visible in this data. In the sequence the Histone can be observed self-assembling, along with the chromatin it supports, into chromosomes that then separate as the cell divides. Simultaneously, macromolecule organelle movement primarily in the cytoplasm of the cell is captured during imaging. Thus, the combination of the two imaging methods provided a fast, simple method for obtaining images with both nuclear and cytoplasm detail. Cell viability using these combined techniques, was maintained for over 4 hours.

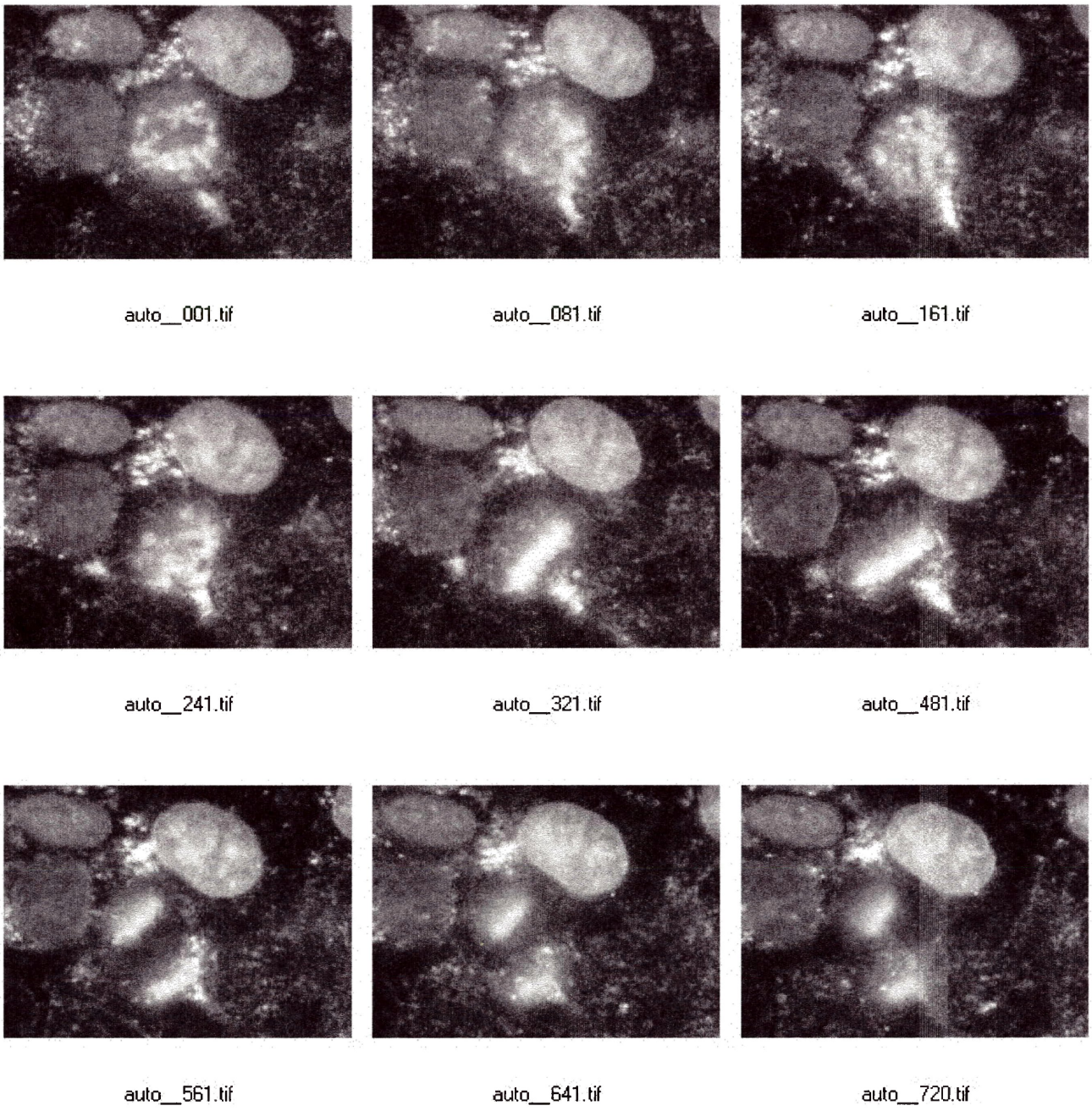


Figure 2.14: Chromatin tagged cancer cell fluorescence in conjunction with oblique illumination.

2.4 Batch Image Processing Software

As one would expect, the better the quality of an image or image sequence, the easier it is to extract meaningful information. It is now commonplace to use computer software to enhance the quality of an image, to optimize special details of interest. Typically this is done manually by the observer in an image editing program. However, once the number of images begins to approach the hundreds or thousands, as they do in these investigations, manual processing becomes impractical. Therefore, during the course of this study, several versions of batch processing software were written to apply the same enhancements to multiple images. The user interface for this software is presented in Figure 2.15.

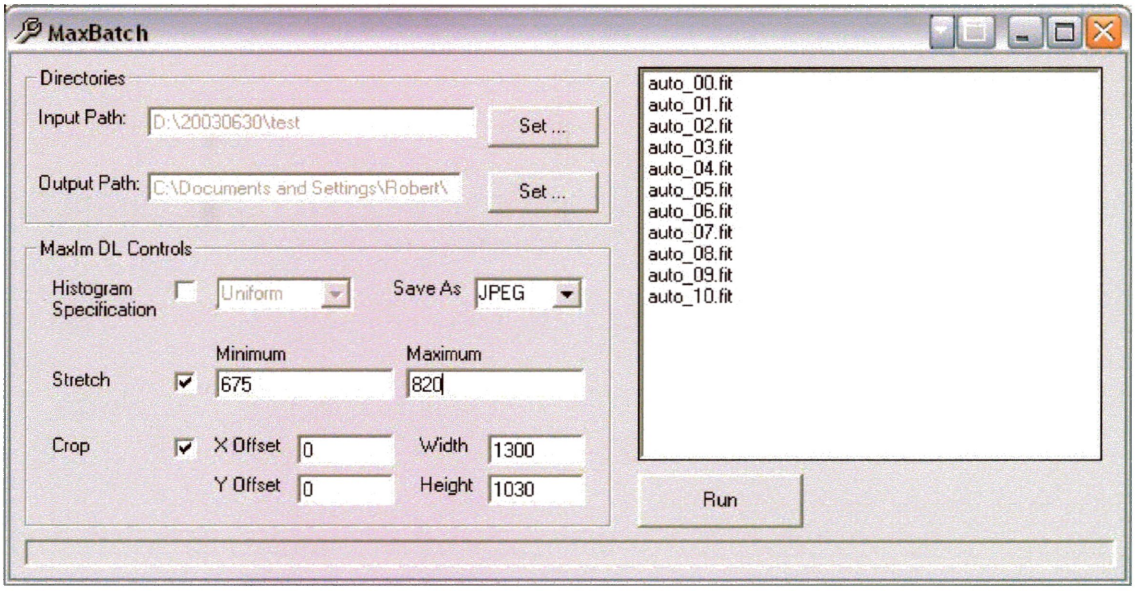


Figure 2.15: Batch processing software interface.

All CCD images taken in this work were captured using Diffraction Limited's MaxIm DL software. The program allows access to a collection of its functions through ActiveX scripting. In this section, a description of the MaxIm DL processing techniques and their subroutine calls (subroutine names in ***bold italics***) is provided. Specific arguments for each routine are given as necessary (see reference 23).

Histogram Specification

Upon capturing an image, MaxIm DL automatically generates a histogram of intensity values for that image. Using histogram specification, the software can force a specified curve onto the histogram; thereby scaling the intensity values according to that function. This option can prove valuable when a high dynamic range of intensity (14-bit camera) is to be compressed into a more viewable form.

HistogramSpec(Curve)

Curve is the argument specifying the shape of the curve to fit. The batch software provides six options for *Curve.*

Value		Curve Type
mxUniformHS	(0)	Uniform
mxExponentialHS	(1)	Exponential
mxLogNormalHS	(2)	LogNormal
mxGaussianHS	(3)	Gaussian
mxRayleighHS	(4)	Rayleigh
mxStraightLineHS	(5)	Linear

Contrast Stretching

Stretching adjusts the brightness and contrast of the image by setting the upper and lower intensity bounds for the grey levels in the image. The intensities are then stretched to fit between these values. The maximum stretch can be thought of as the saturation value and the minimum as the background value.

StretchMax(value)

Value is a floating point number indicating the maximum stretch value.

StretchMin(value)

Value is a floating point number indicating the minimum stretch value.

Cropping

Cropping selects a specified region of the window. All arguments are short integers.

Crop (XOffset, YOffset, Width, Height)

XOffset	Integer	horizontal coordinate of left edge of the cropping rectangle
YOffset	Integer	vertical coordinate of top edge of the cropping rectangle
Width	Integer	width of the cropping rectangle
Height	Integer	height of the cropping rectangle

Save As

MaxIm DL allows images to be saved as many different file types. The default file type is the Flexible Image Transport System (FITS) used by astronomers. This is a 16 bit grayscale format for transport, analysis and archival storage of data endorsed by NASA. The batch program also allows images to be saved as the 8 bit uncompressed grayscale Tagged Image File Format (TIFF) and 85% compressed Joint Photographic Experts Group (JPEG) format. Below the arguments section are the specific values used for each format type.

SaveFile (FilePath, FileFormat, AutoStretch[, SizeFormat, CompressionType])

FilePath	String	specifies the name of the image file to be written
FileFormat	ImageFormatType	specifies the format of the file to be written
AutoStretch	Boolean	specifies whether to automatically stretch the image while the saving
SizeFormat	PixelDataFormatType	[optional] specifies how pixels are represented
CompressionType	Short	[optional] specifies whether the image should be compressed or not

Argument Values For Specific File Types			
Argument	FITS	TIFF	JPEG
File Format	mxFITS (3)	mxTIFF (5)	mxJPEG (6)
Autostretch	False	True	True
Size Format	mx16BitPF (1)	mx8BitPF (0)	mx8BitPF (0)
Compression Type	0	0	0

The result of this development produced a software application, which can process approximately one hundred images in about thirty seconds. This is much more efficient than manual processing which took approximately thirty seconds per images. Enhanced animations of the time sequence can now be generated quickly and effectively.

Chapter 3

Three-Dimensional Imaging Using Confocal and Two-Photon Microscopy

3.1 OpenDX

Once a large time-lapse dataset of 3-D images is acquired using any imaging system, the optimal visualization of the data may require considerable experimentation. Furthermore, if the dataset evolves as a series of time-lapse images, the best choice of viewing angle and magnification are dependent on motion of the sample. Available commercial software is expensive and invariably constrained to only narrow choices of application and data formats, based on previous customer needs. Modifications to this software are very expensive and slow to implement.

The Data Exploration software package developed commercially in the early 1990's by IBM, was a package of general visualization and analysis procedures to allow the scientific community to quickly and easily import, visualize and analyze datasets independent of file format. However, due to the many specific need of the research community, IBM transformed the package into OpenDX, an open source project in the late 1990's, allowing researchers to freely access and customize the software to meet their needs. A downloadable copy of the OpenDX source and various Linux binaries are available at opendx.org and compiled Microsoft and Macintosh binaries can be purchased from Visualization and Imaging Solutions, Inc. for a modest fee. The Microsoft version requires the Exceed X-Windows emulation and fortunately, Lakehead has several licences for this. The current version of OpenDX is 4.2, but IBM stopped providing manuals after version 3.1.4. However, IBM continues to support the project through four e-mail mailing-list forums. Currently, the Cornell Theory Center at Cornell University is one

of the most active OpenDX developers and also provides considerable online documentation and tutorial support for users[37,38,39].

The primary programming interface employed in OpenDX is the Visual Program Editor (VPE). Applications are created by dragging desired modules from a menu and dropping them onto a block diagram page. The modules are then connected by dragging a "wire" from an output terminal of the first module to an input terminal of the second as shown in Figure 3.1. This type of interface has many benefits including: a low learning curve, it allows the powerful functions of OpenDX to be utilized by a scientist without extensive experience in computer programming, and dramatically reduces development time[40].

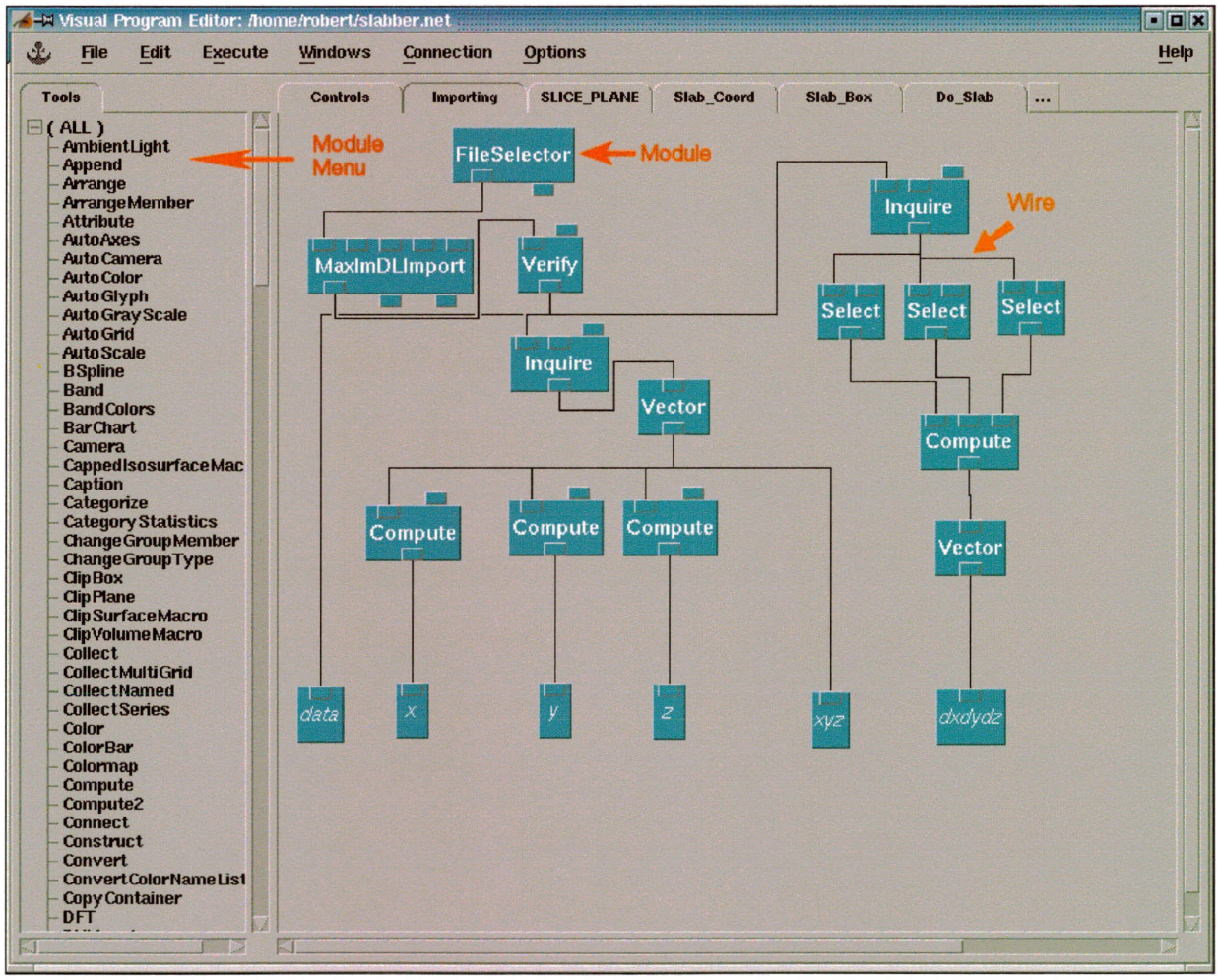


Figure 3.1: OpenDX's Visual Program Editor interface.

The primary disadvantage of the VPE is that the user is required to use predefined, generalized functions. If a user requires a highly customized application with specialized functions, the VPE alone is insufficient. To work around this, the creators of OpenDX have provided the source code to its functions and examples of creating customized modules. These modules are written in the standard C programming language and can be compiled and loaded into OpenDX at the launch of the programming environment. While module creation does require the user to be versed in the C programming language, a high level of customization cannot be otherwise achieved[41].

The necessity for custom module creation presents itself most significantly with OpenDX's data import module, which only recognizes a small set of data formats. While OpenDX does have a tool for handling general data in grid format, but a separate header file must be created for each data file, which can be time consuming. Also, the generation of this file becomes difficult if the data is irregularly spaced, or labels are present within the data. These issues arose in the Fluoview TIFF and FITS formats used in the data collection software during this work, creating a need for specific data import modules to be developed. The marriage of the VPE and custom module creation has made OpenDX and invaluable tool during the course of this work.

3.2 Confocal Imaging and Fluoview TIFF Import

During the course of this study, an Olympus BX51 based confocal scanning laser microscope became available at Lakehead University. The microscope is intended to detect fluorescence but couldn't detect an adequate autofluorescence signal from untagged cells. Thus, we were forced to use fluorescently tagged cells. The instrument produced very clean 3-D section images of the cells to thicknesses of about ½ micron. And, because the cells are approximately 10 microns thick, about twenty sections through the live cells would capture their complete Z-layer properties.

The acquisition software for the BX51 (Image Pro 4.5), stores image data in a variation of the Tagged Image File Format (TIFF), which it calls Fluoview TIFF. The TIFF files collected were about 45 megabytes per section set, so imaging every 5 minutes or so produced huge data sets in a short time. The runs were limited to less than three hours due to optical toxicity effects induced by the relatively high light intensity levels of the excitation laser. While Image Pro has many features and can produce high quality animations for stepping through Z-layers (see Figure 3.2, and video Chap_3_1.mov), its 3-D rendering capabilities were found to be wanting. Thus, this instrument provided the first opportunity to test the flexibility of OpenDX in working 3-D images.

Unfortunately, the Fluoview TIFF format is not natively compatible with OpenDX, hence an import module is required. The TIFF format stores information about its data in *image file directories* (IFD). These directories contain information such as pixel count, data type, image dimensions, and byte offsets of image data[42]. The Fluoview TIFF extends this format by storing multiple image planes in the same file with each plane having its own set of IFDs. It also stores dimensional information such as the number of counts points per dimension and the real space distance between points in that dimension. Therefore, the task for this module is to extract the appropriate image information from each plane's IFD and build a single 3-D array of data values. For the complete source code, refer to Appendix B-1.

The module has successfully allowed for the import of Fluoview TIFF data into OpenDX. Presented in Figure 3.3 is an image of a tagged breast cancer cells (cell line MCF-7) taken using the Olympus BX51 rotated about the Y-axis in 45° increments up to 225°. The image was scanned at a resolution of 0.23 microns in the XY-dimensions and 0.75 microns in the Z-dimension (refer to video Chap_3_2.mov for a full animation taken every 5°). As can be seen, OpenDX provides high quality 3-D rendering, with the images artificially coloured through the available12 bit intensity values.

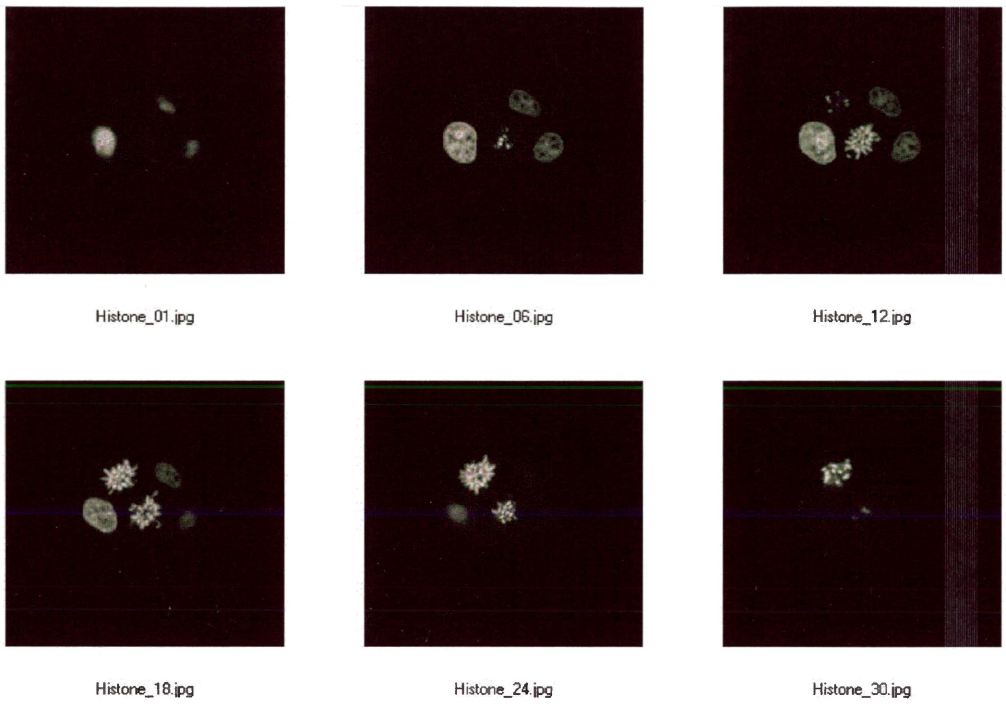


Figure 3.2: Confocal images of MCF-7 breast cancer cells tagged for Histone 1. The images are a sequence of z-planes through the cells.

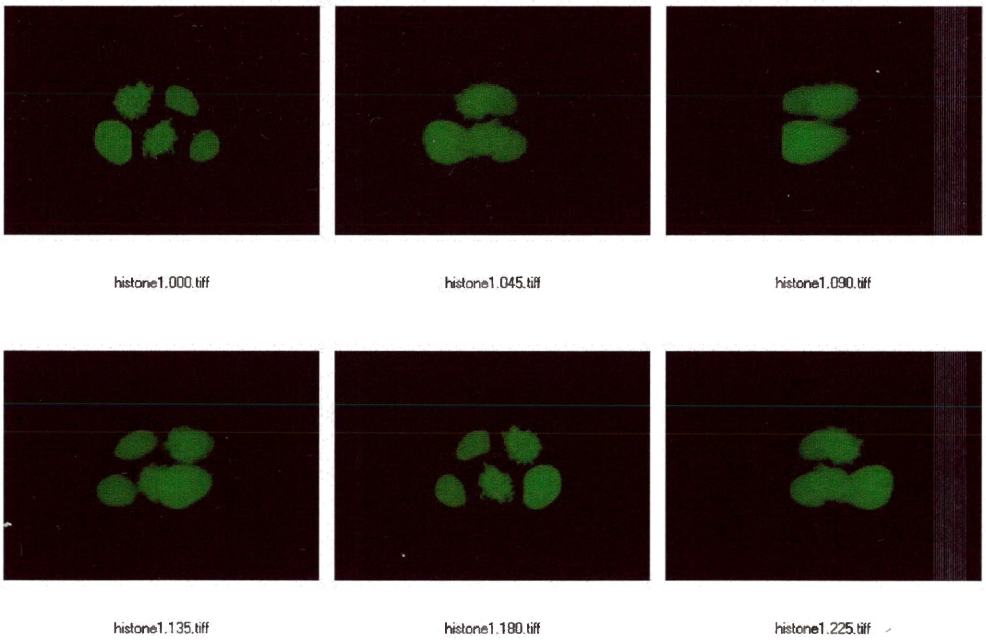


Figure 3.3: 3-D Confocal microscope image of a MCF-7 breast cancer cells rotated every 45° up to 225° using OpenDX.

3.3 Two-Photon Imaging and FITS Import

Currently the laser two-photon microscope has no eyepiece viewing capability. Thus, when imaging samples that are approximately 10 microns thick in the Z-dimension, it is difficult to locate and place the sample in the focal plane of the objective, particularly since only fluorescence derived signal can be recorded. To aid in locating samples in the correct Z-position, a thin film of fluorescent Rhodamine 6G about 3 microns thick, squeezed between a microscope slide and cover slip, is used as an alignment setup tool. Test images of analog driven mirror scans of the type shown in Figure 3.4 aid in locating the position for subsequent sample placement. The grain structure is due to the 90 Mpps arriving while the mirrors are scanning. The left image illustrates the pulsing effect and a scan rate that is too rapid, while the right image shows a nearly uniform illumination at more optimum scan settings.

Figure 3.5 shows two examples of fixed breast cancer (MCF-7) cells tagged with GFP taken with the two-photon microscope in 2-D imaging mode. Average input power was 3mW and a Zeiss 63x (0.9 NA) objective was used. Overall system magnification is approximately 800x. The resolution and image contrast are of very good quality.

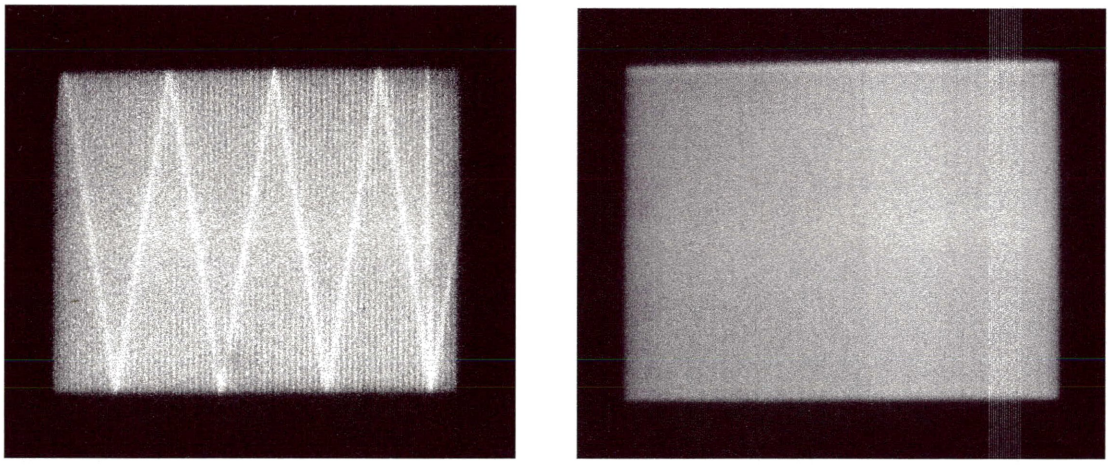


Figure 3.4: Images of two-photon raw scans of R6G between two cover-slips. Left image shows unfilled aliased scanning, while right image has better filling and overlap region is observable.

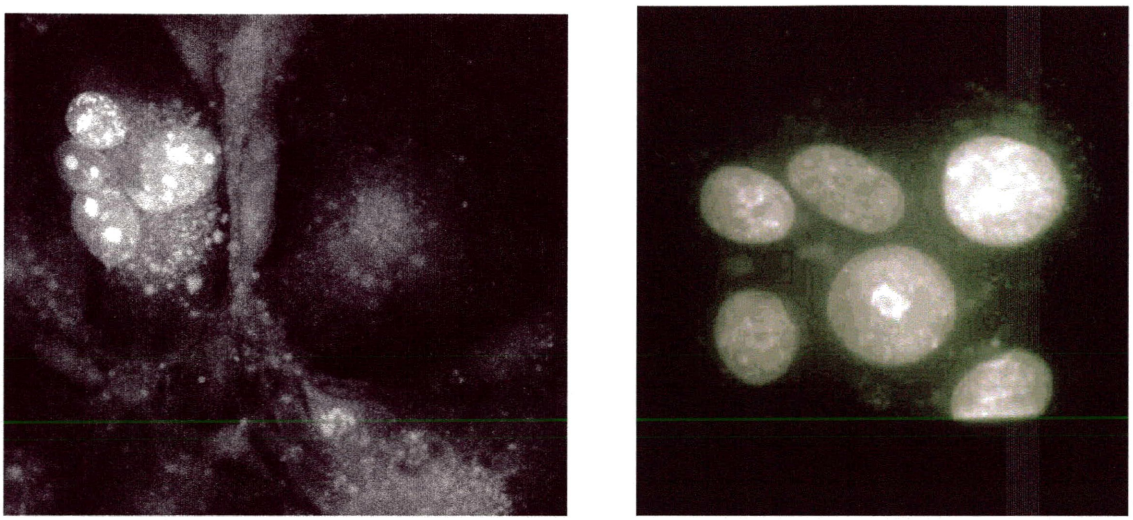


Figure 3.5: Two-photon images of fixed tagged MCF-7 cells.

In order to acquire, manipulate, and render 3-D stacks of image planes on the laser microscope, control software and OpenDX software first had to be developed. Additionally, considerable effort was being devoted to developing the live cell imaging capabilities already described. Once work resumed on this system, it was noticed that the laser microscope images were now significantly degraded. Near this time, Dr. Landry began culturing Hela cells which are large and flat. After some time it became clear that the silver layer on the mirror surfaces had become pitted, or worse had flaked off regions, which, in combination, resulting in much poorer beam quality. The manufacturer estimated a replacement time of 10 months for these mirrors, so they were used despite the problem. In an attempt to compensate, higher laser powers were used but this only led to more rapid cell photobleaching with little improvement in image quality. Nevertheless, 3-D images were taken for a series of Z-planes through Hela cells using the 63x objective and are shown in Figure 3.6 (see video Chap_3_3.mov). Due to the blurring, Z-planes were taken at a spacing of 1 micron apart.

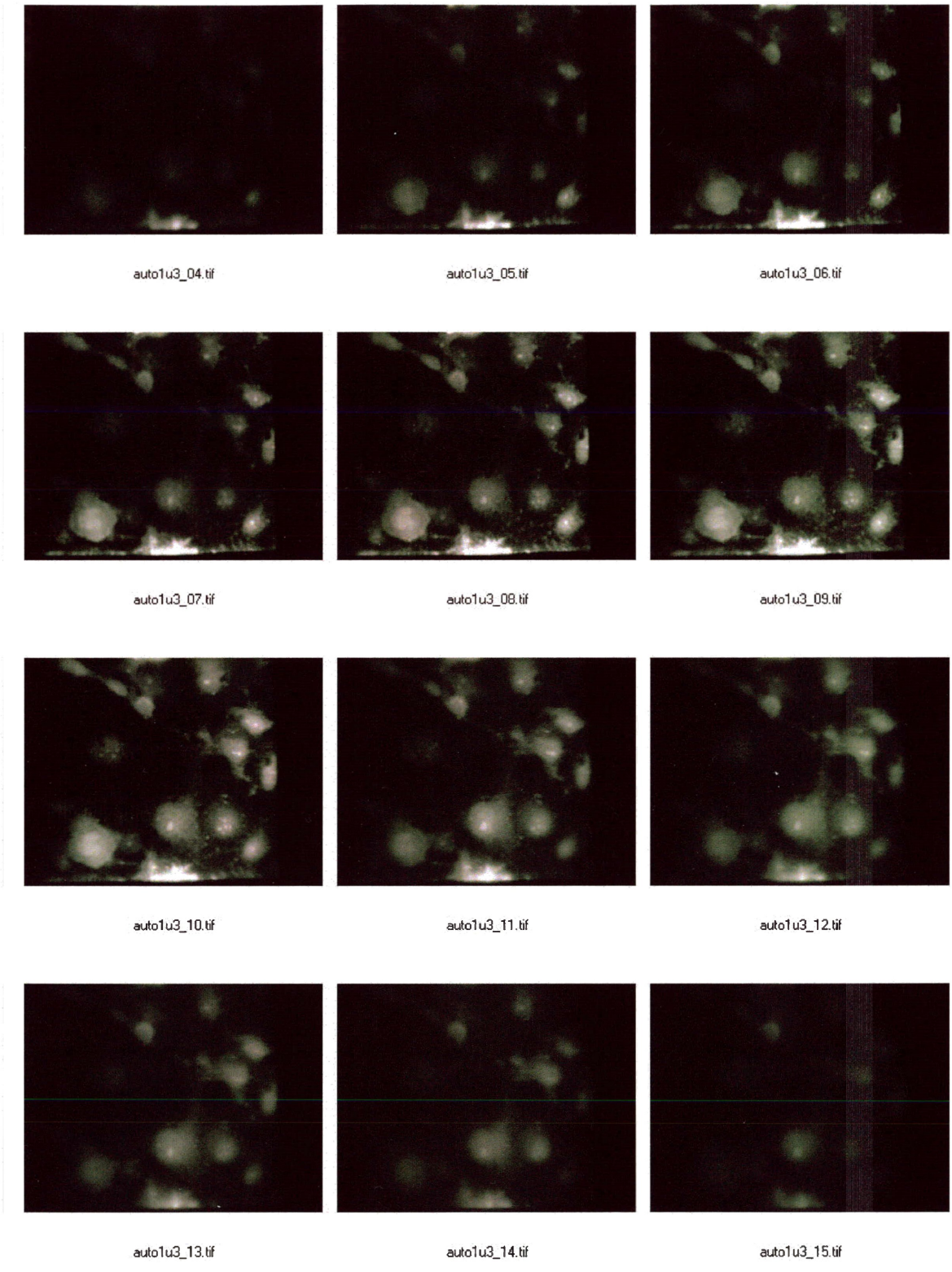


Figure 3.6: Two-Photon Z-sections through Hela cells.

As mentioned earlier, MaxIm DL uses the Flexible Image Transport System (FITS) file format. To be able to import the FITS stacks into OpenDX for rendering, an import module had to be developed. The FITS file structure contains a multi-lined header (one value per line), containing information such as image dimensions, colour depth, contrast settings, etc. Each header line consists of an 8-byte keyword identifier, and optional value and an optional comment. The keyword must be exactly 8-bytes (using ASCII spaces to fill unused characters) and only contain uppercase alphanumeric characters, the underscore, or the hyphen. A header line may contain no more than 80 characters[43]. While the FITS standard does allow multiple images to be stored in a single file, the MaxIm DL only stores one image per file, with the image data directly following the header. Thus, the role of the module for this format is to extract the header information and concatenate a series of images into a single 3-D array of data values. Also, since a variety of microscope objectives may be used with this system, the real space distance between data points must be passed as arguments to the module along with the number of images in the sequence. Refer to Appendix B-2 for the complete source code of this module.

Presented in Figure 3.7 is a series of 3-D rendered images of the Z-stack shown in Figure 3.6. For memory and viewing purposes the stack was cropped in the lower-mid section of the stack and rotated 180° about the Z-axis. The sequence depicts the stack rotated about the Y-axis from 0° – 225° in 45° increments (refer to video Chap_3_4.mov for a full animation taken every 5°). While these images are not yet on par with those of the confocal microscope, a clear variation along the Z-direction can be seen. Based on the earlier 2-D results, once the scan mirrors are replaced the 3-D imaging quality should compete with that of the confocal system.

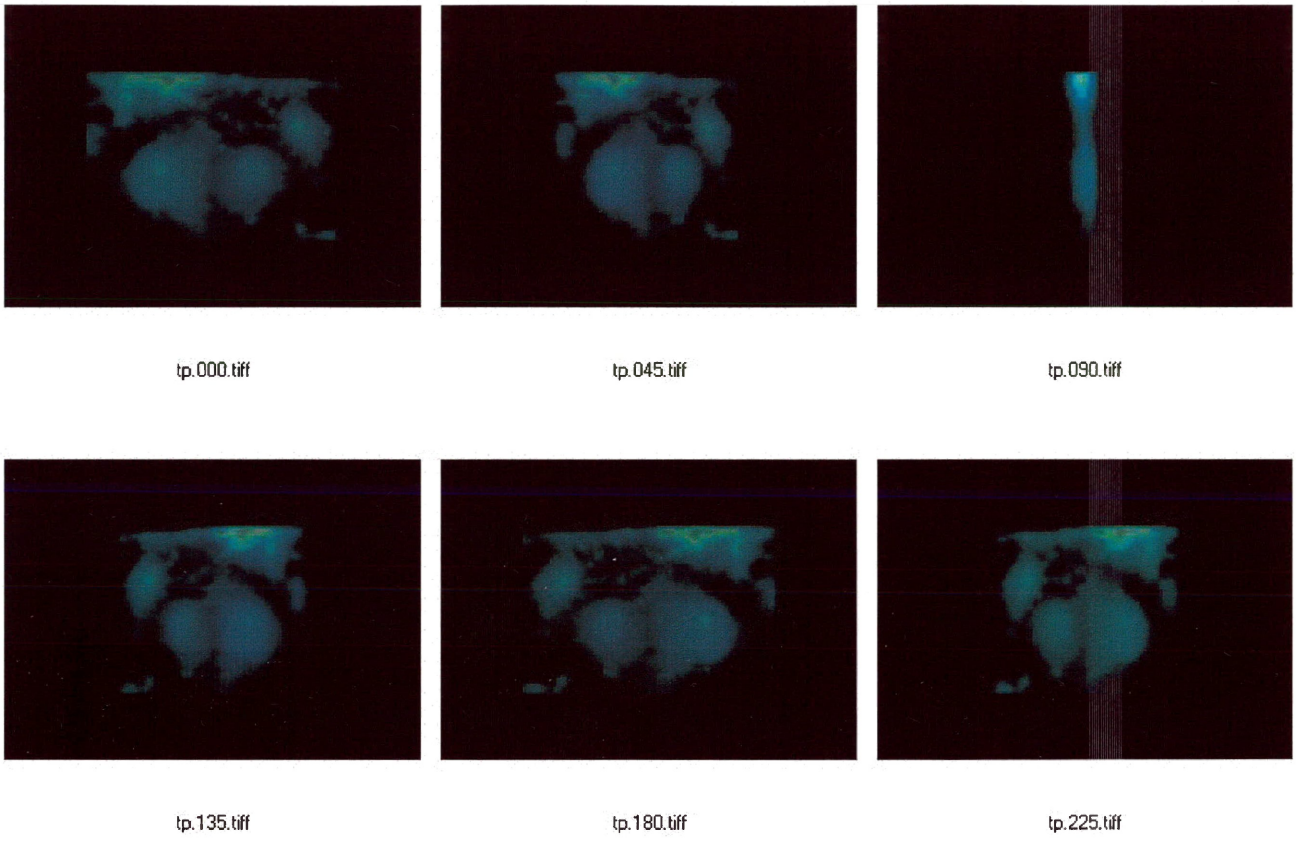


Figure 3.7: 3-D Two-Photon microscope image of a Hela cell stack rotated every 45° up to 225° using OpenDX.

3.4 Slabber Application

Even though the above modules allow for import of the data into OpenDX, the predefined mathematical and rendering functions are optimized for data files in the native OpenDX format. The imported images are typically large files, yet most of the interesting data is located within a small region of the image. This leads to valuable system memory wasted on regions with little scientific value and a significant reduction in processing speed.

To overcome these effects, the Slabber application (*slab* being OpenDX's module for selecting a subset of the data) was developed (see Figure 3.8). First the data is imported using either the Fluoview TIFF or FITS module. The user then defines a 3-D bounding box by adjusting the controls to set the upper-rear-left, and the lower-front-right coordinates of the box.

Finally the data within the bounding box is exported to a new data file in the native OpenDX format. With the data in this format, many different visualization applications can be rapidly investigated and can be transferred to other users without requiring the import module to be present on their systems. This is particularly beneficial to non-UNIX based users since the modules do not transfer well between operating systems.

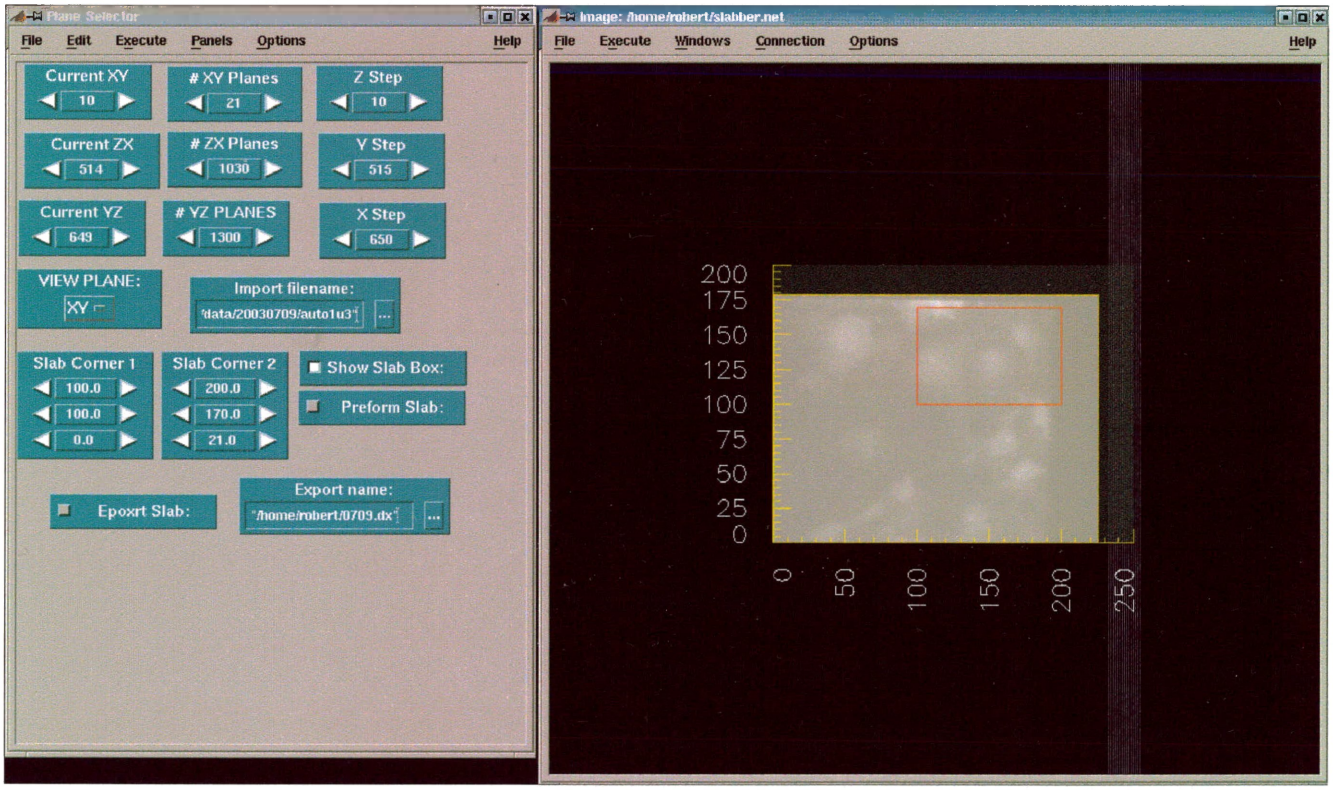


Figure 3.8: Screen capture of the Slabber application's user interface.

Summary

A femtosecond laser microscope system has been developed that can image in 2-D and 3-D, two-photon excited fluorescent samples that can absorb in the 360 - 440nm range. Details of the hardware assembly and controller software development have been given in detail. Based on images acquired to date, the system will compete favourably with commercially available laser microscope alternatives costing several times more.

A range of software products have been developed using C, Visual Basic, and OpenDX development tools for the acquisition, manipulation, and rendering of digital images. The software has been applied to images acquired using confocal, two-photon, fluorescent, phase-contrast, and oblique illumination microscopy techniques.

The results of Demos et al. which state that cancerous tissue appears brighter than normal tissue, when viewed as a 'reflected' image during illumination with visible and near IR light, were verified. Contrary to his supposition that the cause might be due to a difference in molecular content, we propose that the primary cause of a several fold intensity increase in cancerous regions appears to be due to differences in morphology. This conclusion is supported by the lack of significant spectroscopic differences in the two sample types, as well as by subsequent oblique imaging sessions of live cells. These sessions show that highly scattering macromolecules and organelles become confined to a smaller and nearly spherical volume during division; the intensity would be proportional to the density of the scattering sites which is clearly demonstrated in the time-lapse animations.

In the pursuit of improving the visualization of live cultured cells, both a variety of untagged and fluorescently tagged cell lines have been studied. A variety of incubators incorporating the best cell nutrient, thermal, and optical properties were designed and tested. A nearly forgotten technique called oblique illumination microscopy was revived, and found to be an excellent technique for imaging untagged cells. The method also exhibited the lowest level

of phototoxicity as determined by an apparent threefold increase in the duration of metabolic activity when compared to laser scanning and fluorescence imaging techniques. An additional strength of the technique is that it can be used sequentially or concurrently with fluorescence imaging. Based on the above observations, a collaborative study investigating the viability of live cells using various imaging methods was carried out in collaboration with Dr. Sylvie Landry and Dr. Peter McGhee of the NWORCC and was recently published in Optics Express[44].

References

[1] John G. Delly – "Photography Through the Microscope"; Eastman Kodak Company; 1980; Library of Congress Catalog Number 79-54858.

[2] F. Iborra, P. R. Cook and D. A. Jackson - "Applying microscopy to the analysis of nuclear structure and function"; Methods **29;** pp. 131-141; 2003.

[3] T. Haraguchi (preface to a series of reviews) - "Live cell imaging: Approaches for studying protein dynamics in living cells"; Cell Struct. Funct. **27**; pp. 333-334; 2002.

[4] Timothy R. Corle and Gordon S. Kino – "Confocal Scanning Optical Microscopy and Related Imaging Systems"; Academic Press; San Diego, CA; 1996.

[5] Ed. Alberto Diaspro – "Confocal and Two-Photon Microscopy: Foundations, Applications, and Advances"; Wiley-Liss, Inc; New York, NY; 2002.

[6] S.-W. Chu et al. - "In vivo developmental biology study using noninvasive multi-harmonic generation microscopy"; Opt. Express **11**; pp. 3093-3099; 2003.

[7] "Photodynamic Therapy for Cancer: Questions and Answers"; National Cancer Institute; Dec. 2004; Available Online at http://cis.nci.nih.gov/fact/7_7.htm

[8] M.Göppert-Mayer - "Über Elementarakte mit zwei Quanten-sprüngen"; Ann. Phys. 9; pp. 273-295; 1931.

[9] Chris Xu and Watt W. Webb - "Measurement of Two-Photon Excitation Cross Sections of Molecular Fluorophores with data from 690 to 1050 nm"; J. Opt. Soc. Am. B, **13**, No. 3; pp. 481-491; 1996.

[10] Alberto Diaspro and Mauro Robello - "Two-Photon Excitation of Fluorescence for Three-Dimensional Optical Imaging of Biological Structures"; J. Photochem. Photobiol B: Biol, **55**, Issue: 1; pp. 1-8; 2000.

[11] Coherent Laser Group - "Multiphoton Excitation Microscopy: MPE Tutorial; Coherent Inc.; Santa Clara", CA; 2000.

[12] Frank L. Pedrotti and Leno S. Pedrotti – "Introduction to Optics"; Prentice Hall; Upper Saddle River, NJ; pp. 137; 1993.

[13] Max Born and Emil Wolf – "Principles of Optics", 6th ed.; Pergamon Press; Oxford, NY; 1993.

[14] Victoria E. Centonze and John G. White - "Multiphoton Excitation Provides Optical Sections from Deeper within Scattering Specimens than Confocal Imaging"; Biophys. J., **75**; pp. 2015-2024; 1998.

[15] J. P. Zhou, G. Taft, C. P. Huang, M. M. Murnane, H. C. Kapteyn, and I. P. Christov - "Pulse Evolution in a broad bandwidth Ti:sapphire laser"; Optics Letters, vol. 19; pp. 1149-1152; 1994.

[16] I. P. Christov, H. C. Kapteyn, M. M. Murnane, C. P. Huang, and J. P. Zhou - "Space-time focusing of femtosecond pulses in Ti:sapphire," Optics Letters, vol. 20; pp. 309-311; 1995.

[17] I. P. Christov, V. Stoev, M. Murnane, and H. Kapteyn - "Sub-10fs operation of Kerr-lens modelocked lasers"; Optics Letters, vol. 21; pp. 1493; 1996.

[18] S. Backus, C. Durfee, M. M. Murnane, and H. C. Kapteyn - "High Power Ultrafast Lasers"; Review of Scientific Instruments, vol. 69; pp. 1207-1223; 1998.

[19] Many thanks are due to Mr. Rob Knutson for his assembly and wiring of the mirror controllers and slave amplifiers.

[20] Product Datasheet: PCI-6024E; National Instruments; Nov. 2004; Available Online at http://www.ni.com/pdf/products/us/4daqsc202-204_ETC_212-213.pdf

[21] "DAQ: NI-DAQ Function Reference Manual for PC Compatibles", Version 6.6; National Insturments; Austin, TX; Aug. 1999; Part. Number: 321645E-01.

[22] "T-Series Positioning Products User's Manual"; Zaber Technologies Inc; Nov. 2004; Available Online at http://www.zaber.com/t-series/docs/ZaberT-SeriesProductsUserManual.pdf

[23] "MaxIm DL User Manual: Scripting"; Diffraction Limited; Aug. 2004; Avalable Online at http://www.cyanogen.com/help/maximdl/MaxIm-DL.htm

[24] "Nikon USA: Eclipse E400"; Nikon USA; Aug. 2004; Available Online at http://www.nikonusa.com/template.php?cat=5&grp=25&productNr=E400

[25] "Nikon USA: Biological Microscopes E400/E600"; Nikon USA; Aug. 2004; Available Online at http://www.nikonusa.com/bluelink/brochures/E400brochure.pdf\

[26] "Fluorescence Filter Combinations: Blue Excitation: B-3A (Longpass Emission)"; Nikon MicroscopyU; Aug. 2004; Available Online at http://www.microscopyu.com/articles/fluorescence/filtercubes/blue/b3a/b3aindex.html

[27] "Nikon USA: CFI 60 Objectives for Nikon Eclipse Series Microscopes"; Nikon USA; Aug. 2004; Available Online at http://www.nikonusa.com/pdf/Optics_Bio_Chart.pdf

[28] "KX Series Overview"; Apogee Instruments, Inc; Aug. 2004; Available Online at http://www.ccd.com/KXSeries.PDF

[29] "The KX85"; Apogee Instruments, Inc; Aug. 2004; Available Online at http://www.apogee-ccd.com/KX85b.html

[30] S.G.Demos, M.Staggs, H.R. Radousky, R.Gandour-Edwards and R. deVere White - "Cancer detection using NIR elastic light scattering and tissue fluorescence Imaging"; CLEO; Baltimore, MD; May 2001.

[31] R.J.Girardin, W.J.Keeler, J.Th'ng, B.Arjune and P.McGhee – "NIR Auto-fluorescence Imaging Applications in Normal and Cancerous Cells and Tissues"; WesCan Medical Physics Conference; Thunder Bay, ON; Mar. 2002.

[32] Bruce Alberts et al. - "Molecular Biology of The Cell"; Garland Science; New York, NY; 2002.

[33] Ed. Steven R. Goodman – "Medical Cell Biology"; J.B. Lippincott Company; Philadelphia, PA; pp. 170-176; 1994.

[34] Harvey Lodish et al. – "Molecular Cell Biology", 3rd ed.; Scientific American Books; New York, NY; pp. 178-181; 1995.

[35] R. Girardin, W.J.Keeler, John Th'ng and P. McGhee – "NIR Auto-fluorescence Imaging"; Interactions: Canadian Organization of Medical Physics Newsletter; **48** (3); pp. Cover - 1; July 2002 (These results were presented after an invitation by the editor).

[36] "Illumination for Stereomicroscopy: Oblique Illumination"; Nikon MicroscopyU; Aug. 2004; Available Online at http://www.microscopyu.com/articles/stereomicroscopy/stereooblique.html

[37] "About OpenDX"; OpenDX.org; Nov. 2004; Available Online at http://www.opendx.org/about.html

[38] "Open Visualization Data Explorer: Detailed Description"; IBM Corporation; Nov. 2004; Available Online at http://www.research.ibm.com/dx/dxDetailedDescription.html

[39] David L. Thompson – "OpenDX: Paths to Visualization; Visualization and Imagery Solutions"; Missoula, MT; pp.18-21; 2001.

[40] "IBM Visualization Data Explorer: User's Reference", Version 3.1.4; IBM Corp.; Yorktown Heights, NY; 1997.

[41] "IBM Visualization Data Explorer: Programmer's Reference", Version 3.1.4; IBM Corp.; Yorktown Heights, NY; 1997.

[42] Aldus Developers Desk – "TIFF Revision 6.0: Final — June 3, 1992"; Aldus Corporation, Seattle, WA; June 1992.

[43] "Definition of the Flexible Image Transport System (FITS)"; NASA/Science Office of Standards and Technology; Greenbelt, MD; 1999.

[44] Sylvie Landry et al. - "Monitoring live cell viability: Comparative study of fluorescence, oblique incidence reflection and phase contrast microscopy imaging techniques"; Optics Express, Vol. 12, No. 23; pp. 5754-5759, 2004.

Appendix A:

Zaber T-LA28 Class Source Code

```
'*******************************************************************************
'*                                                                             *
'* ZaberTLA28.cls                                                              *
'*                                                                             *
'* PURPOSE                                                                     *
'*                                                                             *
'*    This class is to provide a set of functions and properties for           *
'*    the Zaber T-LA28 linear actuator.  These functions may be used           *
'*    independently of the user interface. One a new ZaberTLA28 object is      *
'*    declared, it will have access to the properties and functions below.     *
'*                                                                             *
'* HISTORY                                                                     *
'*                                                                             *
'*    Author:       Robert Girardin                                          *
'*    Institution:  Lakehead University                                      *
'*    Date:         April 2002 - August 2003                                 *
'*                                                                             *
'*******************************************************************************

Private Const COMPLEMENT = 4294967296#

Private Const MODULUS_MAX = 16777216#
Private Const MODULUS_MED = 65536#
Private Const MODULUS_MIN = 256#

Private Const UM_PER_STEP = 6.35                          'microns per step
Private Const USTEPS_PER_STEP = 64                    'number of microsteps in one step
Private Const UM_PER_USTEP = UM_PER_STEP / USTEPS_PER_STEP  'microns per microstep

Private Const MICRONS_MAX = 28000  'maximum extension in microns
Private Const MICRONS_MIN = 0      'minimum extension in microns

'enumerator corresponding to command codes provided in TLA28 specs
Public Enum ZABER_COMMAND

    zbrReset = 0
    zbrHome = 1
    zbrRenumberAll = 2
    zbrPositionTrack = 8            'reply-only
    zbrManualPosition = 10          'reply-only
    zbrPowerOutOfRange = 14         'reply-only
    zbrMoveAbsolute = 20
    zbrMoveRelative = 21
    zbrConstantSpeed = 22
    zbrStop = 23
    zbrSetMode = 40
    zbrSetMaxStepTime = 41
    zbrSetMinStepTime = 42
    zbrSetAccelRate = 43
    zbrSetMaxExtension = 44
    zbrSetPosition = 45
    zbrSetAlias = 48
    zbrReturnID = 50
    zbrReturnFirmware = 51
    zbrReturnPowerSupply = 52
    zbrReturnSetting = 53
    zbrReturnPosition = 60
    zbrPositionOutOfRange = 255     'reply-only

End Enum
```

```
Public Unit As Byte          'unit number of the actuator
Public Position As Double    'extension in microns of the actuator
Public Comm As Object        'comm object the actuator is connected to

Dim nullData(1 To 4) As Byte  'null data to pass to functions

'*************************************************************************
'*                                                                       *
'*  Function move() returns Integer                                      *
'*                                                                       *
'*  PURPOSE                                                              *
'*                                                                       *
'*      Used to change the extension length of the actuator.  Returns -1 if    *
'*      the desired position is out of the range of the actuator.  Returns 0    *
'*      if the new position is within range.                             *
'*                                                                       *
'*  ARGUMENTS                                                            *
'*                                                                       *
'*      New_Position     - new desired extension length of actuator in microns *
'*      Relative_Change  - boolean indicating if the change is to be made      *
'*                         relative to the current position              *
'*                                                                       *
'*  LOCAL VARIABLES                                                      *
'*                                                                       *
'*      data_bytes - array of bytes to store the position after it has been    *
'*                   converted to a sequence of bytes to be sent over the      *
'*                   Comm Port                                           *
'*                                                                       *
'*************************************************************************

Public Function move(ByVal New_Position As Double, Optional ByVal Relative_Change As
Boolean = False) As Integer

    '* declare local variables
    Dim data_bytes(1 To 4) As Byte


    'determine if position change is to be relative or absolute
    If Relative_Change = False Then

        'absolute change
        If New_Position > MICRONS_MAX Or New_Position < MICRONS_MIN Then

            move = -1  'if position is out of bounds return -1

        Else
            'if position is in bounds

            'convert position to data byte array
            Call positionToBytes(New_Position, data_bytes)

            'send data byte array to unit
            Call sendCommand(Unit, zbrMoveAbsolute, data_bytes)

            'return 0 for indicating postion is in bounds
            move = 0

        End If

    Else

        'relative change
        If (New_Position + Position) > MICRONS_MAX Or (New_Position + Position) <
MICRONS_MIN Then

            move = -1  'if position is out of bounds return -1

        Else

            Call positionToBytes(New_Position, data_bytes)        'convert position to
data byte array
```

```
            Call sendCommand(Unit, zbrMoveAbsolute, data_bytes) 'send data byte array
to unit
            move = 0                                                    'return 0 for
indicating position is in bounds

        End If
    End If

End Function 'end of function Move
```

```
'*************************************************************************************
'*                                                                                   *
'*  Function positionToBytes() returns Variant                                       *
'*                                                                                   *
'*  PURPOSE                                                                          *
'*                                                                                   *
'*      Converts the given position from a double precision number into a            *
'*      four byte array of microsteps (whole number) to be sent over the Comm        *
'*      Port. Due to rounding the uncertainty in the position is +/- 0.5             *
'*      microsteps or 0.05 microns. The function then returns this array.            *
'*                                                                                   *
'*  ARGUMENTS                                                                        *
'*                                                                                   *
'*      New_Position - the double precision number to be converted                   *
'*      data_array   - the array to store the microstep bytes in                     *
'*                                                                                   *
'*  LOCAL VARIABLES                                                                  *
'*                                                                                   *
'*      position_bytes - for byte array to store postion                             *
'*                                                                                   *
'*************************************************************************************

Public Sub positionToBytes(ByVal New_Position As Double, data_array() As Byte)

    'declare local variables
    Dim position_bytes(1 To 4) As Byte

    ' convert position into microsteps
    New_Position = Round(New_Position / UM_PER_USTEP, 0)

    'Complement the value if the position is negative
    If New_Position < 0 Then
        New_Position = COMPLEMENT + New_Position
    End If

   'convert the microsteps into a 4 byte array, most significant byte last
    position_bytes(4) = Int(New_Position / MODULUS_MAX)
    New_Position = New_Position - position_bytes(4) * MODULUS_MAX

    position_bytes(3) = Int(New_Position / MODULUS_MED)
    New_Position = New_Position - position_bytes(3) * MODULUS_MED

    position_bytes(2) = Int(New_Position / MODULUS_MIN)
    New_Position = New_Position - position_bytes(2) * MODULUS_MIN

    position_bytes(1) = New_Position

    For i = 1 To 4
        data_array(i) = position_bytes(i)
    Next i

End Sub      'end function positionToBytes
```

```
'*************************************************************************
'*                                                                       *
'*  Function bytesToPosition() returns Double                            *
'*                                                                       *
'*  PURPOSE                                                              *
'*                                                                       *
'*      Converts the data bytes recieved over the Comm port into a double *
'*      precision position value in microns. Due to rounding the uncertainty *
'*      in the position is +/- 0.5 microsteps or 0.05 microns.           *
'*                                                                       *
'*  ARGUMENTS                                                            *
'*                                                                       *
'*      position_bytes - four byte microstep array to be converted to microns *
'*                                                                       *
'*  LOCAL VARIABLES                                                      *
'*                                                                       *
'*      New_Position - the double precision number to store position     *
'*                                                                       *
'*************************************************************************

Public Function bytesToPosition(position_bytes() As Byte) As Double

    'declare local variables
    Dim New_Position As Double

    'convert four byte array into microsteps, the convert microsteps to microns
    New_Position = MODULUS_MAX * position_bytes(4) + MODULUS_MED * position_bytes(3) +
MODULUS_MIN * position_bytes(2) + position_bytes(1)
    New_Position = New_Position * UM_PER_USTEP

    bytesToPosition = New_Position

End Function    'end function bytesToPosition

'*************************************************************************
'*                                                                       *
'*  Subroutine sendCommand()                                             *
'*                                                                       *
'*  PURPOSE                                                              *
'*                                                                       *
'*      Sends a specifc command with data to a specified actuator via the *
'*      communications port.                                             *
'*                                                                       *
'*  ARGUMENTS                                                            *
'*                                                                       *
'*      act_unit - the actuator unit number the command is to be sent to *
'*      command  - the reference number or the zaber command to initiate *
'*      data     - four byte array containing the binary data for the command *
'*                                                                       *
'*************************************************************************

Private Sub sendCommand(act_unit As Byte, command As ZABER_COMMAND, data As Variant)

    'Send the eight byte data stream to the Comm port
    Comm.Output = Chr$(act_unit) + Chr$(command) + Chr$(data(1)) + Chr$(data(2)) +
Chr$(data(3)) + Chr$(data(4))

    'Delay execution for 10 milliseconds
    timeDelay (0.01)
End Sub     'end sub sendCommand
'*************************************************************************
'*                                                                       *
'*  Subroutine homeUnit()                                                *
'*                                                                       *
'*  PURPOSE                                                              *
'*                                                                       *
'*      Sends the "home" command to the actuator, which returns it to a  *
'*      position of zero microns / microsteps.                           *
'*                                                                       *
'*************************************************************************

Public Sub homeUnit()
```

```
    'Send the home command, no data is required so a null array is passed
    Call sendCommand(Unit, zbrHome, nullData)

End Sub     'end sub homeUnit

'*****************************************************************************
'*                                                                           *
'*  Subroutine renumberUnits()                                               *
'*                                                                           *
'*  PURPOSE                                                                  *
'*                                                                           *
'*      Sends the "renumber" command to all the actuators, which resets the  *
'*      unit numbers of each to their position in the chain from the         *
'*      communications port.                                                 *
'*                                                                           *
'*****************************************************************************

Public Sub renumberUnits()

    'Send the renumber command, no data is required so a null array is passed
    Call sendCommand(0, zbrRenumberAll, nullData)

    'Delay execution for 2.5 seconds
    timeDelay (2.5)

End Sub 'end sub renumberUnits


'*****************************************************************************
'*                                                                           *
'*  Subroutine returnPositionAll()                                           *
'*                                                                           *
'*  PURPOSE                                                                  *
'*                                                                           *
'*      Sends the "return position" command to all the actuators, which causes *
'*      them to return their current extensions (in microsteps) to the system. *
'*                                                                           *
'*****************************************************************************

Public Sub returnPositionAll()

    'Send the "return position" command, no data is required so a null
    'array is passed
    Call sendCommand(0, zbrManualPosition, nullData)

End Sub 'end sub returnPositionAll
```

```
'*******************************************************************************
'*                                                                             *
'*  Subroutine timeDelay()                                                     *
'*                                                                             *
'*  PURPOSE                                                                    *
'*                                                                             *
'*      Suspends execution of the program for the specified number of seconds  *
'*      allowing the firmware in the actuator to complete its operations.      *
'*                                                                             *
'*  ARGUMENTS                                                                  *
'*                                                                             *
'*      d - the delay (in seconds) for the program to suspend execution        *
'*                                                                             *
'*  LOCAL VARIABLES                                                            *
'*                                                                             *
'*      t - stores the system time at which the loop begins                    *
'*                                                                             *
'*******************************************************************************

Private Sub timeDelay(d)

    'Read the current time
    t = Timer

    'Loop until the difference between the current time and start time is
    'greater than the delay
    While (Timer - t) < d
    Wend

End Sub 'end sub timeDelay

'*******************************************************************************
'*                                                                             *
'*  Subroutine returnPosition()                                                *
'*                                                                             *
'*  PURPOSE                                                                    *
'*                                                                             *
'*      Sends the "return position" command to the actuator, calling the       *
'*      routine which causes it to return its current extensions               *
'*      (in microsteps) to the system.                                         *
'*                                                                             *
'*******************************************************************************

Public Sub returnPosition()

    'Send the "return position" command to the actuator numbered "Unit",
    'no data is required so a null array is passed
    Call sendCommand(Unit, zbrReturnPosition, nullData())

    'Delay execution for 0.3 seconds
    timeDelay (0.3)

End Sub 'end sub returnPosition
```

```
'*****************************************************************************
'*                                                                           *
'*  Subroutine resetUnit()                                                   *
'*                                                                           *
'*  PURPOSE                                                                  *
'*                                                                           *
'*      Sends the "reset" command to the actuator, resetting the current unit *
'*      to its startup condition as if it had been powered off. Both the     *
'*      position and number will be lost.                                    *
'*                                                                           *
'*****************************************************************************

Public Sub resetUnit()

    'Pass the reset command to the unit
    Call sendCommand(Unit, zbrReset, nullData())

End Sub 'end sub resetUnit

'*****************************************************************************
'*                                                                           *
'*  Subroutine returnSetting()                                               *
'*                                                                           *
'*  PURPOSE                                                                  *
'*                                                                           *
'*      Caues the actuator to return the value of a specific setting (i.e.   *
'*      position, speed, range, etc.).                                       *
'*                                                                           *
'*  ARGUMENTS                                                                *
'*                                                                           *
'*      setting_number - the setting which is to have its value returned     *
'*                                                                           *
'*  LOCAL VARIABLES                                                          *
'*                                                                           *
'*      setting_bytes - data array storing the setting value                 *
'*                                                                           *
'*****************************************************************************

Public Sub returnSetting(setting_number As ZABER_COMMAND)

    'declare local variables
    Dim setting_bytes(1 To 4)

    'create the command array
    setting_bytes(1) = setting_number
    setting_bytes(2) = 0
    setting_bytes(3) = 0
    setting_bytes(4) = 0

    'send the command
    Call sendCommand(Unit, zbrReturnSetting, setting_bytes)

End Sub 'end sub returnSetting
```

```
'*******************************************************************************
'*                                                                             *
'*  Subroutine setMode()                                                       *
'*                                                                             *
'*  PURPOSE                                                                    *
'*                                                                             *
'*      Used to activate or deactivate certain feature on the actuator.        *
'*                                                                             *
'*  ARGUMENTS                                                                  *
'*                                                                             *
'*      disable_auto_reply - disables all replies except for return commands   *
'*      anti_backlash - enables anti-backlash mode                             *
'*      anti_sticktion - enables anti-stiction mode                            *
'*      disable_pot - disables potentiometer disallowing manual adjustments    *
'*      enable_const_speed_pos_tracking - returns position perodically while   *
'*                                        unit is travelling at constant speed *
'*      disable_manual_pos_tracking - diables replies during manual moves      *
'*      enable_logical_channels_comm_mode - causes data to only be sent in     *
'*                                          byte 3-5, byte 6 is used as ID     *
'*                                                                             *
'*  LOCAL VARIABLES                                                            *
'*                                                                             *
'*      new_mode      - byte storing the mode options                          *
'*      setting_bytes - data array storing the setting value                   *
'*                                                                             *
'*******************************************************************************

Public Sub setMode(disable_auto_reply As Boolean, anti_backlash As Boolean,
anti_sticktion As Boolean, disable_pot As Boolean, enable_const_speed_pos_tracking As
Boolean, disable_manual_pos_tracking As Boolean, enable_logical_channels_comm_mode As
Boolean)

  'declare local variables
  Dim new_mode As Byte
  Dim setting_bytes(1 To 4)

  'use the option values to set the bits of the mode byte
  new_mode = -1 * (CInt(disable_auto_replay) + 2 * CInt(anti_backlash) + 4 *
CInt(anti_sticktion) + 8 * CInt(disabe_pot) + 16 *
CInt(enable_const_speed_pos_tracking) + 32 * CInt(disable_manual_pos_tracking) + 64 *
CInt(enable_logical_channels_comm_mode))

  'create the command array
  setting_bytes(1) = new_mode
  setting_bytes(2) = 0
  setting_bytes(3) = 0
  setting_bytes(4) = 0

  'send the command
  Call sendCommand(Unit, zbrSetMode, setting_bytes)

End Sub 'end sub setMode
```

Appendix B:

OpenDX Data Import Modules

Appendix B-1: Fluoview Tiff Import Source Code

```
/*****************************************************************************
*                                                                            *
* fluoviewImport.h                                                           *
*                                                                            *
* PURPOSE                                                                    *
*                                                                            *
*    A header file with methods for extracting data from images employing    *
*    the Olympus "Fluoview" variation of the Tagged Image File Format (tiff). *
*                                                                            *
* HISTORY                                                                    *
*                                                                            *
*    Author:       Robert Girardin                                           *
*    Institution:  Lakehead University                                       *
*    Date:         April 2002 - August 2003                                  *
*                                                                            *
*****************************************************************************/

/* include header files */

#include <stdio.h>
#include <stdlib.h>

/* define global constants */

#define MAXRANK 3 /* the maximum number of dimensions in the image */
#define X 0       /* integer representation of the x-dimension */
#define Y 1       /* integer representation of the y-dimension */
#define Z 2       /* integer representation of the z-dimension */
```

```
/*****************************************************************************
*                                                                            *
* int fluoviewGetCounts()                                                    *
*                                                                            *
* PURPOSE                                                                    *
*                                                                            *
*    To determine the the number of data points in each dimesion and their   *
*    respective real space deltas, and to set the origin of the dataset to   *
*    [0 0 0]                                                                 *
*                                                                            *
*****************************************************************************/

int fluoviewGetCounts(char *filename, int *counts, float *origins, float *deltas)
{

   /* define local variables */

   FILE *in_file;        /* pointer to the input file */
   char entry[101];      /* string to store bytes read from file */
   unsigned short tag;   /* two bytes of directory inicating type of data */
   unsigned long ifdOffset;  /* byte offest of the current IFD */
   unsigned short dir_count; /* number of directories in the current IFD */
   unsigned short i;         /* dimension counter */
   unsigned short j;         /* dimension counter */
   unsigned long Offset100Bytes = 0; /* byte offset of Fluoview information */



/* check to see if the input file can be opened, if not return 1 */

   if((in_file = fopen(filename, "r")) == NULL) return 1;

   /* read first for bytes and check to see if it corresponds to hex value
      2A4949, if it does, the file is a fluoview tiff, if not return 1     */

   fscanf(in_file, "%4c", entry);

   if(*((unsigned long *) entry) != 0x2A4949)
   {
      fclose(in_file);
      return 1;
   }

   /* read the next 4 bytes to determine the byte offset of the first
      image file directory and move to that location                    */

   fscanf(in_file, "%4c", entry);

   ifdOffset = *((unsigned long *) entry);
   fseek(in_file, ifdOffset, SEEK_SET);

   /* read to next 2 bytes to determine the number of tags directory
      entries                                                          */

   fscanf(in_file, "%2c", entry);

   dir_count = *((unsigned short *) entry);

   /* loop through the 12 byte directory entries until the entry
      corresponding to the offset of the 100 bytes of aquisition information
      is reached. It is reached when the first two bytes of the entry
      equals 34361 or hex value 8639                                    */

   for(i = 0; i < dir_count; i++)
   {
      fseek(in_file, ifdOffset + 12*i + 2, SEEK_SET);

      fscanf(in_file, "%12c", entry);

      tag = *((unsigned short *) entry);

      if(tag == 34361)
```

```
      {
         Offset100Bytes = *((unsigned long *) (entry + 8)) + 284;
      }
   }

   /* move to the offset of the first 100 bytes of information and loop
      through each dimension                                          */

   fseek(in_file, Offset100Bytes, SEEK_SET);
   for(i = 0; i <  MAXRANK; i++)
   {
       /* ignore the first 16 bytes */
       fscanf(in_file, "%16c", entry);


      /* read the next 4 bytes which indicate the number of data points
         in that dimension and record in the counts array              */

      fscanf(in_file, "%4c", entry);

      counts[i] = (int) *((unsigned long *) entry);

      /* ignore the next 8 bytes */
      fscanf(in_file, "%8c", entry);

      /* read the next 8 bytes and loop through each dimension */

      fscanf(in_file, "%8c", entry);

      for(j = 0; j < MAXRANK; j++)
      {
         /* set the origin of the current dimension to 0 */
         origins[i] = 0.0f;
         if(i == j)
         {
           /* set the spacing between points in the current dimension
              to the last value read from the file                    */

           deltas[i*MAXRANK + j] =(float) *((double *) entry);
         }
         else
         {
           deltas[i*MAXRANK + j] = 0.0f;
         }
      }


/* ignore the last 64 bytes of data */
      fscanf(in_file, "%64c", entry);
   }

   /* close the image file */
   fclose(in_file);

   /* return the success value of 0 to the calling function */
   return 0;
}
/* end fluoviewGetCounts */

/*****************************************************************************
*                                                                            *
* int fluoviewGetCounts()                                                    *
*                                                                            *
* PURPOSE                                                                    *
*                                                                            *
*    To read the data from the image planes into a single 3D data array.     *
*                                                                            *
*****************************************************************************/

int fluoviewGetData(char *filename, int *counts, unsigned short *data)
{
```

```
/* define local variables */

FILE *in_file;       /* pointer to the input file */
char entry[13];      /* string to store bytes read from file */

unsigned short tag;
unsigned short tag_type; /* two bytes of directory indicating type of data */
unsigned long tag_counts;/* number of counts in the current tag */
unsigned long tag_value; /* the data value of the current tag */

unsigned long strip_offset_count = 0;  /* the number of strip offsets */
unsigned long strip_offset_offset = 0; /* the byte offset the fist strip offset */
unsigned long *strip_offsets;          /* array of strip offsets */
unsigned long strip_offset_counter;    /* counter to loop through strip offsets */

unsigned long strip_byte_count = 0;    /* the number of bytes in the strip */
unsigned long strip_byte_offset = 0;   /* the byte offset of the strip's bytes */
unsigned long *strip_bytes;            /* array of strip bytes */
unsigned long strip_byte_counter;  /* counter to loop through strip bytes */

unsigned long ifdOffset;     /* byte offest of the current IFD */
unsigned short dir_count;    /* number of directories in the current IFD */
unsigned short cur_dir;      /* the current directory being read */
unsigned long i;   /* dimension counter */
unsigned long j;   /* dimension counter */
unsigned long k;   /* dimension counter */
unsigned long counter; /* integer counter */
unsigned long sum;     /* integer counter */

/* open the image file */
in_file = fopen(filename, "r");

/* read in the byte offset of the IFD */
fseek(in_file, 4, SEEK_SET);
fscanf(in_file, "%4c", entry);
ifdOffset = *((unsigned long *) entry);
/* loop through each plane */
for(k = 0; k < counts[Z]; k++)
{
   /* go to the IFD offset */
   fseek(in_file, ifdOffset, SEEK_SET);



   /* determine the number of directories */
   fscanf(in_file, "%2c", entry);
   dir_count = *((unsigned short *) entry);

   /* loop through each directory */
   for(cur_dir = 0; cur_dir < dir_count; cur_dir++)
   {
         /* extract the current tag and it's attributes */
         fscanf(in_file, "%12c", entry);
         tag        = *((unsigned short *) entry);
         tag_type   = *((unsigned short *) (entry + 2));
         tag_counts = *((unsigned long *) (entry + 4));
         tag_value  = *((unsigned long *) (entry + 8));

         /* tag of 273 indicates that this is strip
            offset data */
         if(tag == 273)
         {
            strip_offset_count  = tag_counts;
            strip_offset_offset = tag_value;
         }

         /* tag of 279 indicates that this is strip
            byte data */
         if(tag == 279)
         {
            strip_byte_count  = tag_counts;
```

```
            strip_byte_offset = tag_value;
         }
      }

      /* get the next IFD byte offset */
      fscanf(in_file, "%4c", entry);
      ifdOffset = *((unsigned long *) entry);

      /* allocate memory for the strip arrays */
      strip_bytes   = malloc(strip_byte_count * sizeof(unsigned long));
      strip_offsets = malloc(strip_offset_count * sizeof(unsigned long));

      /* go to the strip byte offset */
      fseek(in_file, strip_byte_offset, SEEK_SET);

      /* get the number of bytes in each strip */
      for(counter = 0; counter < strip_byte_count; counter++)
      {
         fscanf(in_file, "%4c", entry);
         strip_bytes[counter] = *((unsigned long *) entry);
      }

      /* go to the strip offset offset */
      fseek(in_file, strip_offset_offset, SEEK_SET);
      /* get the offsets of each strip */
      for(counter = 0; counter < strip_offset_count; counter++)
      {
         fscanf(in_file, "%4c", entry);
         strip_offsets[counter] = *((unsigned long *) entry);
      }
      /* reset the sum counter */
      sum = 0;

      /* loop through each strip offset */
      for(strip_offset_counter   =   0;   strip_offset_counter   <   strip_offset_count;
strip_offset_counter++)
      {
         /* go to the current strip offset */
         fseek(in_file, strip_offsets[strip_offset_counter], SEEK_SET);

         /* loop through each byte int he current strip */
         for(strip_byte_counter        =        0;        strip_byte_counter
strip_bytes[strip_offset_counter] / 2; strip_byte_counter++)
         {
            /* read in the data value */
            fscanf(in_file, "%2c", entry);

            /* determine the array coordinates the data will be entered into */
            i = sum % (unsigned long) counts[X];
            j = (unsigned long) counts[Y] - sum / (unsigned long) counts[X] - 1;

            /* store the data in the array */
            data[(unsigned  long)  counts[Z]*(unsigned  long)  counts[Y]*i  +  (unsigned
long) counts[Z]*j + k] = *((unsigned short *) entry);

            /* increment the sum counter */
            sum++;
         }
      }

      /* free the memory taken by the arrays */
      free(strip_bytes);
      free(strip_offsets);

   }

   /* close the image file */
   fclose(in_file);

   /* return a success value of 0 */
   return 0;
}
```

```
/* end fluoviewGetData
/* end fluoviewImport.h
```

```
/*****************************************************************************
*                                                                            *
* fluoviewImport.c                                                           *
*                                                                            *
* PURPOSE                                                                    *
*                                                                            *
*    An OpenDX module for importing data stored using                        *
*    the Olympus "Fluoview" variation of the Tagged Image File Format (tiff). *
*    Most of this is simply a modifation of the example provided with OpenDX. *
*                                                                            *
* HISTORY                                                                    *
*                                                                            *
*    Author:       Robert Girardin                                           *
*    Institution:  Lakehead University                                       *
*    Date:         April 2002 - August 2003                                  *
*                                                                            *
*****************************************************************************/

/* include header files */
#include <dx/dx.h>
#include <stdio.h>
#include "fluoviewImport.h"

/* standard OpenDX Module function definition */
Error m_fluoviewImport(Object *in, Object *out)
{
   Array a=NULL; /* an OpenDX array object */
   Field f=NULL; /* an OpenDX field object */

   int i;                  /* integer counter */
   char *filename;         /* name of the Fluoview file */
   unsigned short *data;   /* array to stor the image data in */
   int numelements;    /* the total number of elements in the data array */
   int counts[MAXRANK];/* array containing the number of data points in each dimension
*/
   float origins[MAXRANK*MAXRANK]; /* array defining the origin of the data */
   float deltas[MAXRANK]; /* array containg the spacial deltas between data points */

   /* make sure file name is provided */
   if(!in[0])
   {
      DXSetError(ERROR_BAD_PARAMETER, "filename is required");
      goto error;
   }
   /* make sure file name is a string */
   else if (!DXExtractString(in[0], &filename))
   {
      DXSetError(ERROR_BAD_PARAMETER, "filename must be a string");
      goto error;
   }

   /* retrieve the aquisition information from the file */
   if(fluoviewGetCounts(filename, counts, origins, deltas) != 0)
   {
      /* report error if file does not exist or is not a Fluoview Tiff */
      DXSetError(ERROR_BAD_PARAMETER, "file \"%s\" in not accessible, or is not a
fluoview multi-tiff", filename);
      goto error;
   }

/* calculate the number of elements in the data array */
   numelements = 1;
   for(i = 0; i < MAXRANK; i++)
   {
      numelements = numelements*counts[i];
   }

   /* OpenDX array creation method */
   a = DXNewArray(TYPE_USHORT, CATEGORY_REAL, 0);
   DXAddArrayData(a, 0, numelements, NULL);
   data=(unsigned short *) DXGetArrayData(a);
```

```
    /* fill the array with the data */
    fluoviewGetData(filename, counts, data);

    /* OpenDX example method for field creation */
    f = DXNewField();
    if(!f) goto error;

    DXSetStringAttribute((Object) a, "dep", "positions");
    DXSetComponentValue(f, "data", (Object) a);
    a = NULL;

    a = DXMakeGridConnectionsV(MAXRANK, counts);
    DXSetComponentValue(f, "connections", (Object) a);
    a = NULL;

    a = DXMakeGridPositionsV(MAXRANK, counts, origins, deltas);
    DXSetComponentValue(f, "positions", (Object) a);

    /* output the new OpenDX object */
    out[0] = (Object) f;

    /* return successfully to calling application */
    return OK;

/* in the case of an error, halt execution and return an error message
    to the calling application */

error:
return ERROR;

}
/* end fluoviewImport.c */
```

Appendix B-2: FITS Import Source Code

```
/*****************************************************************************
*                                                                            *
* MaxImDLImport.h                                                            *
*                                                                            *
* PURPOSE                                                                    *
*                                                                            *
*    A header file with methods for extracting data from image stacks        *
*    taken using MaxIM DL and stored in the fits format. All image planes    *
*    must be located in the same directory and comform to the numbering       *
*    system used by the Zaber controll and acquisiton software:              *
*                                                                            *
*    filebase_filenumber.fit                                                 *
*                                                                            *
*    for example                                                             *
*                                                                            *
*    /home/user/imagename_000.fit                                            *
*                                                                            *
*    filebase   = "/home/user/imagename                                      *
*    filenumber = "000"                                                      *
*                                                                            *
*    Note:  the filenumber string must begin at zero, increment by 1 and     *
*           contain the corrent number of leading zeros (i.e. for 1000 - 9999 *
*           images, the first image must have a file number of "0000").      *
*                                                                            *
* HISTORY                                                                    *
*                                                                            *
*    Author:       Robert Girardin                                           *
*    Institution:  Lakehead University                                       *
*    Date:         April 2002 - August 2003                                  *
*                                                                            *
*****************************************************************************/

/* include header files */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/* define global constants */

#define MAXRANK 3 /* the maximum number of dimensions in the image */
#define X 0       /* integer representation of the x-dimension */
#define Y 1       /* integer representation of the y-dimension */
#define Z 2       /* integer representation of the z-dimension */
```

```
/*****************************************************************************
*                                                                            *
* void getHeaderValue()                                                      *
*                                                                            *
* PURPOSE                                                                    *
*                                                                            *
*    To read the header data value from a specific line in the FITS header.  *
*                                                                            *
*****************************************************************************/

void getHeaderValue(FILE *in_file, int line, float *header_var)
{
   /* go to the specified header line and move to the data position */
   fseek(in_file, 80*line + 10, SEEK_SET);

   /* read in the header value */
   fscanf(in_file, "%f", header_var);
}
/* end getHeaderValue */

/*****************************************************************************
*                                                                            *
* int maximGetCounts()                                                       *
*                                                                            *
* PURPOSE                                                                    *
*                                                                            *
*    To determine the the number of data points in each dimesion and data    *
*    scaling and contrst factors.                                            *
*                                                                            *
*****************************************************************************/

int maximGetCounts(char *filebase, int slices, int *counts, float *bscale, float
*bzero, float *background, float *range)
{
   /* define local variables */

   FILE *in_file; /* pointer to the input file */
   char header_tag[11]; /* variable for storing the label part of the header */
   float rows;         /* counter store the number of data points in the y-dimension */
   float cols;         /* counter store the number of data points in the x-dimension */
   int cur_line;         /* counter store the number current header line being read */

   /* The infile_pattern variable stores the string format to be used with the sprintf
      command when generating filename strings */
   char infile_pattern[240];

   char infile_name[255];/* string to hold the complete path of the current z-plane */
   unsigned short leading_zeros; /* the number of leading zeros required in the file
path */
   int n; /* stores the return value of the first sprintf function call for
debugging*/


   /* determine the number of leading zeros in the file names */
   leading_zeros = (unsigned short) log10(slices) + 1;


   /* generate the file pattern */
   n = sprintf(infile_pattern, "%s_%%0%dd.fit", filebase, leading_zeros);
   /* generate the file name for the first z-plane and open the file */
   sprintf(infile_name, infile_pattern, 0);
   if((in_file = fopen(infile_name, "r")) == NULL) return 1;

   /* set the number of z-planes to 1 */
   cur_line = 1;

   /* scan in the header label */
   fscanf(in_file, "%s", header_tag);

   /* if the header label matches the desired label, read in the data value */
   while(strcmp(header_tag, "END") != 0)
   {
```

```
      fseek(in_file, 80*cur_line, SEEK_SET);
      fscanf(in_file, "%s", header_tag);

      /* number of points in the x-dimension */
      if(strcmp(header_tag, "NAXIS1") == 0) getHeaderValue(in_file, cur_line, &cols);

      /* number of points in the y-dimension */
      if(strcmp(header_tag, "NAXIS2") == 0) getHeaderValue(in_file, cur_line, &rows);

        /*  BZERO and BSCALE are scaling factors for the data point where
         REAL_VALUE = BZERO + BSCALE * STORED_VALUE
         */
      if(strcmp(header_tag, "BZERO") == 0)  getHeaderValue(in_file, cur_line, bzero);
      if(strcmp(header_tag, "BSCALE") == 0) getHeaderValue(in_file, cur_line, bscale);

      /* contrast variables */
      if(strcmp(header_tag,  "BACKGRND=")  ==  0)  getHeaderValue(in_file,  cur_line,
background);
      if(strcmp(header_tag, "RANGE") == 0) getHeaderValue(in_file, cur_line, range);

      /* go to next line */
      cur_line++;
   }

   /* store the number of points for each dimension */
   counts[X] = (int) cols;
   counts[Y] = (int) rows;
   counts[Z] = slices;

   /* close the input file */
   fclose(in_file);

   /* return successfully to calling application */
   return 0;
}
/* end maximGetCounts */
```

```
/*****************************************************************************
*                                                                            *
* int maximGetData  ()                                                       *
*                                                                            *
* PURPOSE                                                                    *
*                                                                            *
*    To read the data from the image files into a single 3D data array.      *
*                                                                            *
*****************************************************************************/

int maximGetData(char *filebase, int *counts, float bscale, float bzero, unsigned
short *data)
{
/* define local variables */

   unsigned long cur_file; /* the current image file being read */
   unsigned short leading_zeros;/* the number of leading zeros required in the file
path */
   unsigned long i; /* varibale for determining array element location */
   unsigned long j; /* varibale for determining array element location */

   /* The infile_pattern variable stores the string format to be used with the sprintf
      command when generating filename strings */
   char infile_pattern[240];
   char infile_name[255]; /* string to hold the complete path of the current z-plane
*/

   unsigned char pixel[3];   /* array to store pixel data */
   unsigned long num_pixels; /* the total number of pixels */
   unsigned long cur_pixel;  /* the current pixel being read */
   unsigned short pix_val;   /* the value of the current pixel being read */

   FILE *in_file; /* pointer to the input file */

   /* determine the number of leading zeros in the file names */
   leading_zeros = (unsigned short) log10(counts[Z]) + 1;

   /* loop through each z-plane */
   for(cur_file = 0; cur_file < counts[Z]; cur_file = cur_file++)
   {
      /* generate the input file name and open the file for reading */
      sprintf(infile_pattern, "%s_%%0%dd.fit", filebase, leading_zeros);
      sprintf(infile_name, infile_pattern, cur_file);
      in_file = fopen(infile_name, "r");

      /* seek byte 2880 in the file which is where the data begins */
      fseek(in_file, 2880, SEEK_SET);

      /* calculate the number of pixels in the image */
      num_pixels = (unsigned long) (counts[X]*counts[Y]);

      /* loop through each pixel */
      for(cur_pixel = 0; cur_pixel < num_pixels; cur_pixel++)
      {
         /* read the stored value in the file */
         fscanf(in_file, "%2c", pixel);
```

```
         /* calculate the real value */
         pix_val = (unsigned short) (bzero + bscale * (float) (256*pixel[0]
+pixel[1]));

         /* determine the data array element locations */
         i = cur_pixel % (unsigned long) counts[X];
         j = cur_pixel / (unsigned long) counts[X] - 1;

         /* store the pixel value in the data array */
         data[(unsigned long) counts[Z]*(unsigned long) counts[Y]*i + (unsigned long)
counts[Z]*j + cur_file] = pix_val;
      }

      /* close the input file */
      fclose(in_file);
   }

   /* return successfully to the calling application */
   return 0;
}

/* end maximGetData */

/* end MaxIMImport.h */
```

```
/*****************************************************************************
*                                                                            *
* MaxImDLImport.c                                                            *
*                                                                            *
* PURPOSE                                                                    *
*                                                                            *
*    An OpenDX module for importing data stored in FITS files generated by   *
*    MaxIM DL.                                                               *
*                                                                            *
*    All image planes must be located in the same directory and comform to the*
*    numbering system used by the Zaber controll and acquisiton software:    *
*                                                                            *
*    filebase_filenumber.fit                                                 *
*                                                                            *
*    for example                                                             *
*                                                                            *
*    /home/user/imagename_000.fit                                            *
*                                                                            *
*    filebase   = "/home/user/imagename                                      *
*    filenumber = "000"                                                      *
*                                                                            *
*    Note:  the filenumber string must begin at zero, increment by 1 and     *
*           contain the corrent number of leading zeros (i.e. for 1000 - 9999 *
*           images, the first image must have a file number of "0000").      *
*                                                                            *
*    Most of this is simply a modifation of the example provided with OpenDX. *
*                                                                            *
* HISTORY                                                                    *
*                                                                            *
*    Author:       Robert Girardin                                           *
*    Institution:  Lakehead University                                    *
*    Date:         April 2002 - August 2003                                  *
*                                                                            *
*****************************************************************************/

/* include header files */
#include <dx/dx.h>
#include <stdio.h>
#include "MaxImDLImport.h"

/* standard OpenDX Module function definition */
Error m_MaxImDLImport(Object *in, Object *out)
{
   Array a=NULL; /* an OpenDX array object */
   Field f=NULL; /* an OpenDX field object */

   int i, j;       /* array index counters */
   char *filebase; /* refer to file PURPOSE section */
   unsigned short *data; /* array to store the image data in */

   int numelements; /* the total number of elements in the data array */
   int  counts[MAXRANK];  /*  array  containing  the  number  of  data  points  in  each
dimension */
   float origins[MAXRANK*MAXRANK]; /* array defining the origin of the data */
   float deltas[MAXRANK}; /* array containg the spacial deltas between data points */

   float bzero;  /* stored data numerical offset value */
   float bscale; /* stored data scaling factor */
   float background; /* background level for contrast setting */
   float range;      /* difference between background an maximum data value */
   float stretch_max;/* calculated maximum for contrast setting */

   int num_z; /* number of z-planes */

   /* set the data origin and delta values to zero */
   for(i = 0; i < MAXRANK; i++)
   {
      origins[i] = 0.0f;
      for(j = 0; j < MAXRANK; j++)
      {
         deltas[i*MAXRANK + j] = 0.0f;
      }
```

```
    }

    /* make sure filebase is provided */
    if(!in[0])
    {
       DXSetError(ERROR_BAD_PARAMETER, "filebase is required");
       goto error;
    }

    /* make sure filebase is a string */
    else if (!DXExtractString(in[0], &filebase))
    {
       DXSetError(ERROR_BAD_PARAMETER, "filename must be a string");
       goto error;
    }

    /* make sure that the number of z-planes is an integer */
    else if (!DXExtractInteger(in[1], &num_z))
    {
       DXSetError(ERROR_BAD_PARAMETER, "slices must be an integer");
       goto error;
    }

    /* make sure the delta values are floating point numbers */
    else if (!DXExtractFloat(in[2], &deltas[0]))
    {
       DXSetError(ERROR_BAD_PARAMETER, "delta x must be a float");
       goto error;
    }
    else if (!DXExtractFloat(in[3], &deltas[4]))
    {
       DXSetError(ERROR_BAD_PARAMETER, "delta y must be a float");
       goto error;
    }
    else if (!DXExtractFloat(in[4], &deltas[8]))
    {
       DXSetError(ERROR_BAD_PARAMETER, "delta z must be a float");
       goto error;
    }

    /* retrieve the aquisition information from the first file */
    if(maximGetCounts(filebase, num_z, counts, &bscale, &bzero, &background, &range) !=
0)
    {
       /* report error if file does not exist or is not the proper format */
       DXSetError(ERROR_BAD_PARAMETER, "file \"%s\" in not accessible, or is not a
MaxImDL fit", filebase);
       goto error;

    }

    /* calculate the number of elements in the data array */
    numelements = 1;
    for(i = 0; i < MAXRANK; i++)
    {
       numelements = numelements*counts[i];
    }

    /* OpenDX array creation method */
    a = DXNewArray(TYPE_USHORT, CATEGORY_REAL, 0);
    DXAddArrayData(a, 0, numelements, NULL);
    data=(unsigned short *) DXGetArrayData(a);

    /* fill the array with the data */
    maximGetData(filebase, counts, bscale, bzero, data);

    /* OpenDX example method for field creation */
    f = DXNewField();
    if(!f) goto error;

    DXSetStringAttribute((Object) a, "dep", "positions");
    DXSetComponentValue(f, "data", (Object) a);
```

```
    a = NULL;

    a = DXMakeGridConnectionsV(MAXRANK, counts);
    DXSetComponentValue(f, "connections", (Object) a);
    a = NULL;

    a = DXMakeGridPositionsV(MAXRANK, counts, origins, deltas);
    DXSetComponentValue(f, "positions", (Object) a);

    /* determine the maximum contrast parameter */
    stretch_max = background + range;

    /* generate the three module outputs */
    out[0] = (Object) f;
    out[1] = (Object) DXNewArray(TYPE_FLOAT, CATEGORY_REAL, 0, 0);
    out[2] = (Object) DXNewArray(TYPE_FLOAT, CATEGORY_REAL, 0, 0);

    /* assign the background and maximum contrast values to outputs 1 and 2
       respectively */
    DXAddArrayData((Array) out[1], 0, 1, &background);
    DXAddArrayData((Array) out[2], 0, 1, &stretch_max);

    /* return successfully to calling application */
    return OK;

    /* in the case of an error, halt execution and return an error message
       to the calling application */
error:
    return ERROR;

}
/* end fluoviewImport.c */
```