Development of an Annotation Plug-in to assist Classifying Phenotypes in μCT Images of Mutant Mice

William Grimes

Department of Computer Science

University College London

Supervision: Prof A. Kitamoto and Dr K. Bryson

2013 September

Abstract

The National Institute of Informatics (NII) in collaboration with the National Institute of Genetics (NIG) in Japan are pioneering research interpreting the roles of genes on the development of the mammalian embryo. Using mice as the principle study model, genes are systematically removed or knocked-out and mice embryos developed to 14.5 days post-coitum, whereupon a μCT scan is performed, by looking for any abnormalities in the μCT scan the function of the knocked-out gene can be indicated. In order to automatically detect phenotypic abnormalities deformation fields are registered against a normal mean.

The absence of significant visual features can make it difficult to automatically recognise novel phenotypes and requires human experts to analyse images from μCT scans. The pipeline for this analysis involves multiple two-dimensional images of detected areas being viewed and analysed by biology experts in the field of mice embryology, areas of significance are then commented upon. Analysis by experts of large image sets from μCT slices is not an ideal methodology to obtain feedback about candidate novel phenotypes, and has the potential for misinterpretation.

The development of a software solution would improve analysis of areas detected by the defective area algorithm in images of mutant mice embryos. An annotation system that would easily allow Regions Of Interest (ROIs) identified by the defective area algorithm and deformation field morphometry to be classified as suspicious or not would improve the efficiency of this feedback process. Comments added would be semantically related with specific ROIs and this annotation data along with quantitative data maybe output in a useful format to make extracting meaning from the data easier.

Acknowledgements

I would like to acknowledge the support of my supervisors Prof A. Kitamoto and Dr K. Bryson. Their guidance and useful critiques of this research work has been invaluable to successfully completing the project. In particular I would like to thank Prof A. Kitamoto for his guidance in our biweekly meetings and Dr K. Bryson for his help formulating this report.

I would also like to thank Prof. Boudier at the Universite Pierre et Marie Curie, for his programming advice and assistance. My grateful thanks are also extended to my colleagues Samuel Kerjose and Sharmili Roy Sethy.

I would also like to extend my thanks to the National Institute of Informatics for providing the laboratory and for their help in offering me the resources to conduct this research.

Contents

List of Figures

List of Tables

1

Introduction

This chapter gives an introduction to the project, setting the context of the research, giving an outline of the objectives of the project, its goals and the approach taken to the software engineering process. In addition the scope of this report will be discussed.

1.1 Project Outline

DNA sequencing techniques have allowed the complete genome to be collected for hundreds of species, including the human species. The Human Genome Project (HGP) has successfully determined the sequence of chemical base pairs which make up DNA, and has identified all of the genes in the human genome from a physical standpoint. The current scientific challenge is to to ascribe function to the thousands of discovered genes. This requires a description of functions of a gene in normal physiology and development, as well as the contribution of mutant alleles to inherited diseases (1, 2).

A powerful approach to help decipher the function of genes is to manipulate genes by performing gene knockout; a genetic technique in which one or more of an organism's genes are made inoperative. This has been particularly successful using embryonic stem cells in mice via gene targeting[1] or gene trapping[2] techniques.

[1]Gene targeting is a directed approach that uses homologous recombination to mutate an endogenous gene. Specifically, this method can be used to delete a gene, remove exons, and introduce point mutations either permanently or in a conditional fashion.

[2]Gene trapping is a high-throughput approach that is used to introduce insertional mutations across the mammalian genome.

The principle study model for the human genome is the mouse owing to the ability to knockout genes and the 99% genetic homology between mouse and human (3). In addition the mouse is suitable because it has an excellent track record as a model for human diseases and traits, it has a highly accurately recorded mouse genome sequence, and there are sophisticated genetic tools and resources available for the mouse.

To ascribe function and interpret genetic information gene targeting technology is used on the approximately 23,000 mouse genes (4). High-throughput phenotypic assessment systems systematically knockout genes and analyse and interpret the genetic information generated. Multiple systems have been proposed within the scientific community to analyse the genetic information generated and identify phenotypes.

In developing mice embryos a generalised defect detection framework that automatically computes candidate phenotypic areas has shown promising results (1). The sensitivity and specificity of the detection algorithm with respect to two established genetic defects ventricular septal defect (VSD) and polydactyly allows for evaluation in terms of the precision of the detection algorithm. Complete phenomic analysis of a mouse strain is however a very tedious and slow process.

In order to evaluate all of the detected areas, manual annotation by experts is required. In the case of research conducted at National Institute of Informatics into phenotype detection in morphological mutant mice using deformation features (1), feedback is provided by experts at the National Institute of Genetics (NIG) in Japan indicating the biological significance of detected areas. The format in which this feedback was provided was sufficient for the small scale evaluation database, but it is important for a larger data set that the feedback mechanism be improved and the data outputted in a useful form to make analysis easier.

This paper describes the software development process to improve the efficiency of the feedback system that assesses whether candidate phenotypes can be classified as novel or as known. It is an extension to the work conducted at the National Institute of Informatics in Tokyo, Japan on phenotype detection in morphological mutant mice using deformation features(1).

The human genome project has already yielded many biological and computational challenges, the development of software and extensions to aid in analysis and interpretation of image data continues to be essential to the success of this global scientific undertaking.

1.2 Project Goals

This project aims to deliver the following overarching goals. In addition to these generic goals a full list of specific requirements can be found in section 3.3 of this report.

- Viewing of 3D stacks and navigation through stack. It is of principle importance for accurate analysis that the 3D μCT scan is compatible with the software package being developed. The user should also be able to interact with the scan by navigating through slices of the stack.
- Overlay Regions Of Interest (ROIs) onto 3D stacks, allowing suspicious areas that the defective area algorithm detects to be highlighted on the 3D μCT scan.
- Semantically relate ROIs with unique IDs. For each ROI a unique ID should be assigned to improve analysis of large data sets and for referencing regions.
- Provide a Graphical User Interface (GUI) for the addition of comments to IDs. The principle purpose of the software system being developed is to allow scientists to add comments relating to a detected volume, so a user friendly GUI should be available to add comments.
- Provide a quantitative and general feedback mechanism for each ROI perhaps indicating if an area is significant or not. The exact nature of feedback is ascribed by NII.

1.3 Project Approach

An iterative and incremental approach was taken towards the project. In the initial planning stage a project schedule and preliminary goals were established, following this in the planning stage small test projects were run and the development environment established, requirements were collected from interviews with researchers in the area leading to detailed analysis and design of the project. The project was implemented, tested and evaluated, further refinements were made before being available to use for research purposes, see figure 1.1.

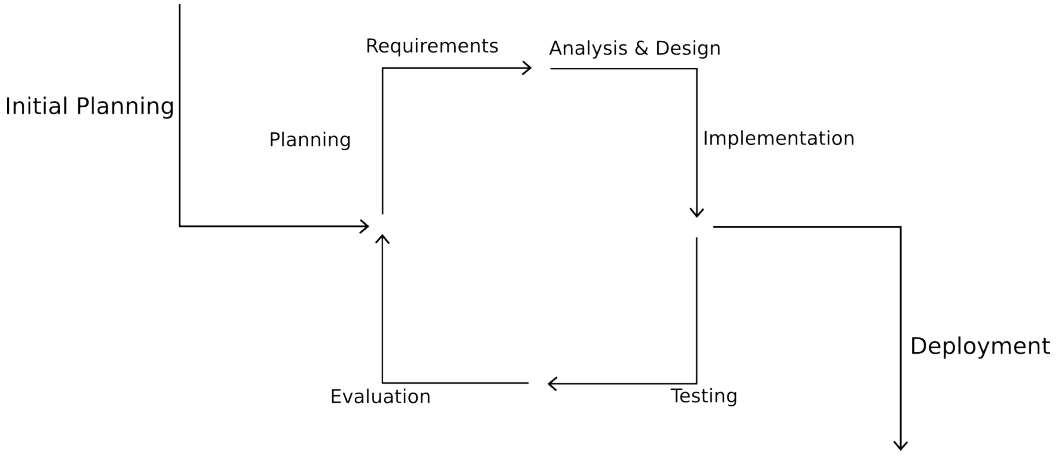

Figure 1.1: Iterative development model - the phases used in the software engineering process for this project.

1.4 Report Scope

The scope of this report following from the introduction will include a chapter describing the research and background information relating to this project along with related work in chapter 2. Chapter 3 describes the requirements of the project and analysis. The design and implementation of the project is discussed in chapter 4, followed by project testing in chapter 5. Finally, chapter 6 will evaluate the effectiveness of the solution provided and conclude with potential further development.

2

Research and Related Work

In order to comprehend the nature of the problem that this software seeks to solve background to the biological and computational aspects of the project is included in this report. This chapter will outline related work, in addition, previous research and existing software will be critiqued against the goals of this project. Alternative imaging software and plug-ins will be discussed as well as useful software, tools, library code and frameworks.

2.1 Biological

The International Mouse Phenotyping Consortium (IMPC) has the goal of phenotyping targeted knockout mouse strains throughout the whole mouse genome by 2021 (4). Subsequently, there is a large body of published and ongoing work relating to mouse embryonic phenotyping which provides a useful basis to understanding how this project and software system fits into a larger global context of matching genes with their function.

This software engineering project focuses heavily on creating an annotation system to assist in phenotype detection in morphological mutant mice using deformation features (1). A generic deformation based defect detection framework for 3D μCT images of mutant mice has shown promising results and greatly improves throughput of phenotypic detection. The system also highlights candidate novel defects with an absence of visual features which can then be analysed by experts. The annotation system will be of use providing feedback that can be easily interpreted to give useful results. It

is proposed that the software solution should be generic so can be applied in multiple contexts within a customisable framework.

There are various other approaches being applied to the challenge of phenotyping that will briefly be discussed. The method of characterising morphological phenotypes differs in these various research methodologies. Many studies rely on the idea of an atlas, which is achieved by registering multiple images into a mean image and segmenting the image based on anatomical structures. Variation in these anatomical structures based on volume for mutant mice strains can be automatically detected (4). Other studies rely on morphometric analysis via MRI images (5, 6, 7, 8, 9). Staining of tissues followed by X-ray computed tomography is an alternative approach to phenotyping that has been successful (10, 11). A variety of 3D imaging modalities are available for assessment of genetically engineered mice embryos: magnetic resonance microscopy (MRM), X-ray microcomputed tomography (μCT), optical projection tomography (OPT), episcopic and cryoimaging, and ultrasound biomicroscopy (UBM). MRM and micro-CT are particularly well suited for evaluating structural information at the organ level, whereas episcopic and OPT imaging provide structural and functional information from molecular fluorescence imaging at the cellular level (12).

2.2 Computational

Digital images are two-dimensional grids of pixel intensities with the width and height being defined by the number of pixels in the x and y directions (13). A stack is a three-dimensional visualisation comprised of multiple two-dimensional images called slices. In two-dimensions pixels are the smallest single components of an image holding numeric values or pixel intensities, a three-dimensional volumetric pixel element is known as a voxel. The bit-depth for pixels and voxels is defined as the number of unique intensity values that can exist in the image, the higher the bit-depth the better the representation of an image, there is however a trade off since higher bit-depth images contain more data so are larger. Images and stacks may be of different file formats, which are standardised means of organising and storing digital images. An image file format may store data in uncompressed, compressed, or vector formats. Once rasterised, an image becomes a grid of pixels, each of which has a number of bits to designate its colour equal to the colour depth of the device displaying it.

There are a cornucopia of software tools available for visualisation and image analysis, many of which are available for clinical evaluation of medical images. This project focuses on open source scientific software tools for scientific visualisation and image analysis rather than clinical evaluation. Two processes important in image analysis for scientific visualisation are segmentation and registration. Segmentation is the process of identifying and classifying data found in a digitally sampled representation. Typically the sampled representation is an image acquired from such medical instrumentation as CT or MRI scanners. Registration is the task of aligning or developing correspondences between data. For example, in the medical environment, a CT scan may be aligned with an MRI scan in order to combine the information contained in both.

Within the pipeline of this project image data for phenotype detection in morphological mutant mice is initially provided by the National Institute of Genetics from 3D μCT scans using Scanxmate-E090S (Comscantecno) scanners in Digital Imaging and Communications in Medicine (DICOM) format. DICOM is a standard for handling, storing, printing, and transmitting information in medical imaging, it also includes a file format definition and a network communications protocol.

From the DICOM files deformation features are extracted and analysed to compute phenotypic and candidate phenotypic areas using non-linear registration of mutant embryo to a normal consensus average image. This data is output in MetaImage (mhd) format which is a header file's suffix. MetaImage format is supported by the software libraries ITK[1] and VTK[2] which are widely used in scientific image processing.

[1]Insight Segmentation and Registration Toolkit (ITK) is a cross-platform, open-source application development framework widely used for the development of image segmentation and image registration programs. The toolkit provides segmentation and registration algorithms in two, three, and more dimensions. ITK is implemented in C++ and it is wrapped for Python and Java. This enables developers to create software using a variety of programming languages.

[2]The Visualization Toolkit (VTK) is an open-source, cross-platform, freely available software system for 3D computer graphics, image processing and visualization. VTK consists of a C++ class library and several interpreted interface layers including Tcl/Tk, Java, and Python. VTK supports a wide variety of visualisation algorithms including: scalar, vector, tensor, texture, and volumetric methods. At its core VTK is implemented as a C++ toolkit, requiring users to build applications by combining various objects into an application. The system also supports automated wrapping of the C++ core into Python, Java and Tcl, so that VTK applications may also be written using these interpreted programming languages.

2.3 Survey of Existing Software

An investigation into the existing software and available tools was essential to understand to what extent the existing software can provide the desired features. This section gives a detailed investigation into four existing packages VV, the 4D Slicer, ITK-SNAP, 3D Slicer and ImageJ to ascertain to what extent they meet the goals of the project and how they maybe extended to satisfy all the requirements.

2.3.1 VV, the 4D Slicer

The software previously used at the National Institute of Informatics for the purposes of visualising the 3D μCT images was VV, the 4D slicer (14). VV is an open-source and cross platform image viewer built on ITK and VTK, espressly designed for fast and simple visualisation of spatio-temporal images and supporting 3D imaging through time. VV is highly suited for the purposes of qualitative evaluation of image registration and deformation field visualization. VV provides support for MetaImage file formats natively, however does not provide many tools for quantitative feedback. It is commonly used in the field of radiation therapy to help researchers and clinicians evaluate deformation in 4D CT images of the thorax and is implemented in C++ (15).

VV has features for sophisticated synchronised navigation between several images, orientations or time sequences as shown in figure 2.1. Pixel values of each synchronised images at current cursor position are available. Another advantage of the software is the ease with which image stacks can be overlayed, a factor of great importance to this project. There is further functionality within VV that make it good for this task such as the cross-hairs that display the slice number and coordinates. Further to this the transparency of the overlay may also be changed or the overlay easily hidden and shown. Many features however relate to the temporal aspect of imaging, which is not relevant for this project.

2.3.2 ITK-SNAP

ITK-SNAP is an interactive software application that allows users to navigate three-dimensional medical images, manually delineate anatomical regions of interest, and perform automatic image segmentation (figure 2.2). The software was designed for

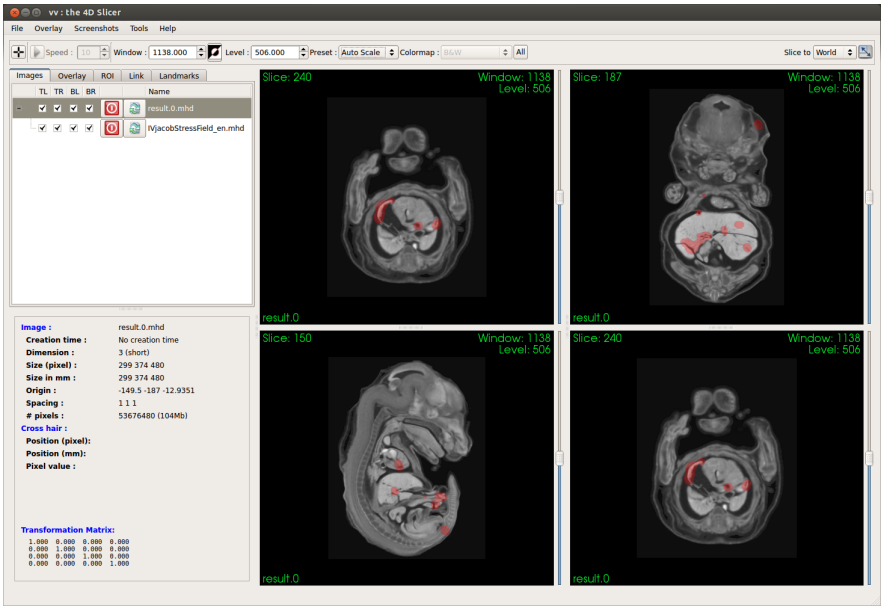

Figure 2.1: VV, the 4D Slicer - screen capture showing how VV maybe used to view 3D μCT scans of mouse embryos. The red areas are the overlayed deformation field highlighting regions of interest to be analysed, this overlay can be removed as desired.

clinical and scientific research with an emphasis on a user-friendly interface and maintaining a limited feature set to prevent feature creep. ITK-SNAP is most frequently used to work with magnetic resonance imaging (MRI) and computed tomography (CT) data sets (16).

ITK-SNAP is a tool similar to VV but has features for level-set segmentation, this is not particularly beneficial to the objectives of this project. The greatest drawback is not being able to overlay regions of interest onto image stacks which makes ITK-SNAP an unlikely candidate software for development.

2.3.3 3D Slicer

3D Slicer aims to provide a software platform for research in both basic biomedical and clinically applied settings, including modules on image guided surgery robotics, brain mapping, and virtual colonoscopy (17). 3D Slicer is a free open source software that is a flexible, modular platform for image analysis and visualisation. It can be easily extended to enable development of both interactive and batch processing tools for a variety of applications.

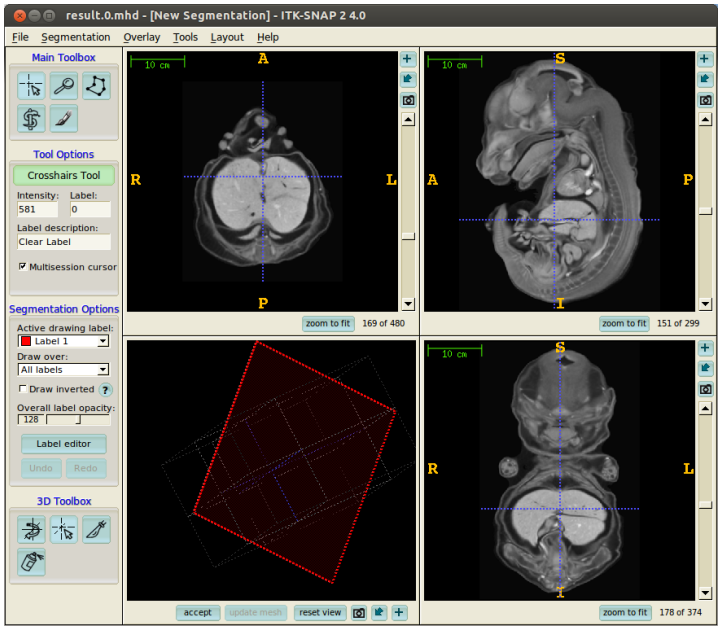

Figure 2.2: ITK-SNAP - screen capture showing how ITK-SNAP maybe used to view 3D μCT scans of mouse embryos. It was not possible to overlay regions of interest onto image stacks on two-dimensional images.

3D Slicer provides image registration, an interface to external devices for image guidance support, and Graphical Processing Unit (GPU) enabled volume rendering, among other capabilities. 3D Slicer has a modular organisation that allows the easy addition of new functionality and provides a number of generic features not available in competing tools. The interactive visualisation capabilities of 3D Slicer include the ability to display arbitrarily oriented image slices, build surface models from image labels, and high performance volume rendering, some of which can be seen in figure 2.3.

3D slicer can handle DICOM images, display interactive visualisation of volumetric voxel images, polygonal meshes, and volume renderings, automatic image segmentation, analysis and visualisation of diffusion tensor imaging data and tracking of devices for image-guided procedures.

2.3.4 ImageJ

ImageJ is a public domain open-source Java based image processing program inspired by NIH Image software for the Macintosh (13). It runs, both as an online applet or as

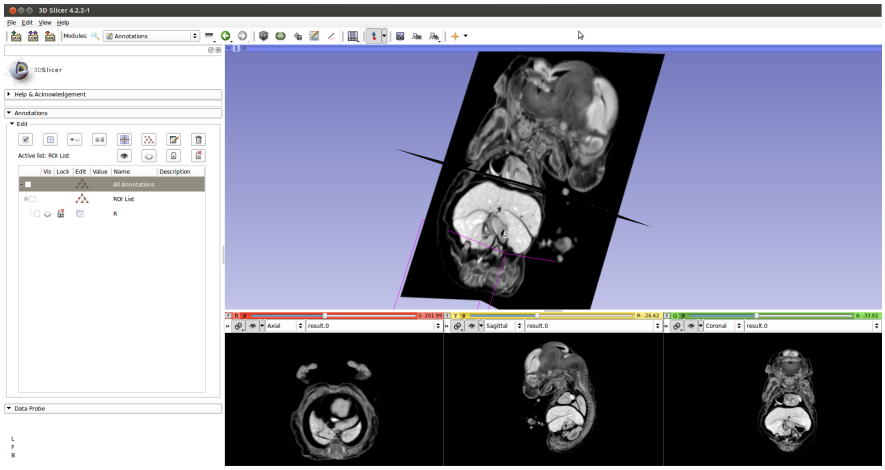

Figure 2.3: 3D Slicer - screen capture showing how 3D Slicer maybe used to view 3D μCT scans of mouse embryos and the 3D visualisation features. It was not possible to overlay regions of interest in MetaImage format onto image stacks or two-dimensional images.

a downloadable application, on any computer with a Java 1.1 or later virtual machine. Figure 2.4 ImageJ was developed primarily for image processing in research rather than for use in a clinical setting. It can display, edit, analyse, process, save and print 8-bit, 16-bit and 32-bit images in formats including TIFF, GIF, JPEG, BMP, DICOM, FITS and raw, with a suitable plug-in it also supports MetaImage formats. It supports stacks and hyper-stacks and is multi-threaded, so time-consuming operations such as image file reading can be performed in parallel with other operations.

ImageJ has the following useful features for processing and analysis built in:

- it can calculate area and pixel value statistics of user-defined selections,
- it can measure distances and angles,
- it can create density histograms and line profile plots,
- it supports standard image processing functions such as contrast manipulation, sharpening, smoothing, edge detection and median filtering,
- it does geometric transformations such as scaling, rotation and flips,
- spatial calibration is available to provide real world dimensional measurements in units such as millimetres,

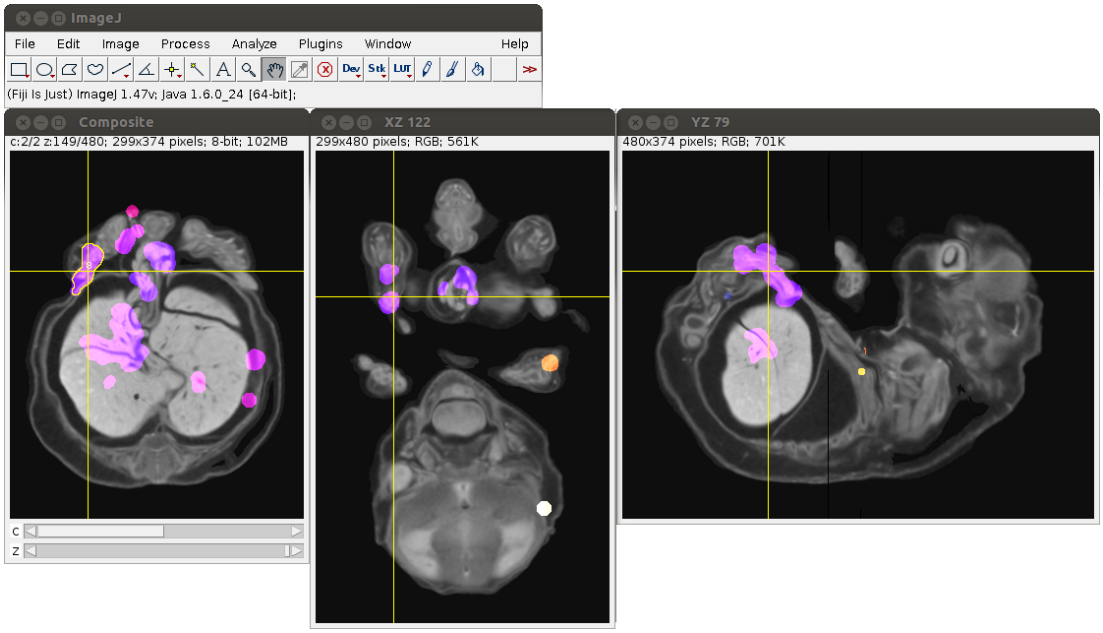

Figure 2.4: ImageJ - screen capture showing how ImageJ maybe used to view 3D μCT scans of mouse embryos with orthogonal views and overlayed regions of interest. The ImageJ 3D viewer can also be used to visualise a stack in 3D.

- it can zoom images up to 32:1 and down to 1:32,
- the program supports any number of images simultaneously, limited only by available memory.

ImageJ was designed with an open architecture that provides extensibility via Java plug-ins. Custom acquisition, analysis and processing plug-ins can be developed using ImageJs built in editor and Java compiler. There are already many plug-ins available and user-written plug-ins make it possible to solve almost any image processing or analysis problem. The ImageJ source code is freely available in the public domain.

ImageJ plug-ins include the 3D object counter (18) which is used to locate 3D objects within a stack and multiple plug-ins with 3D functionality (19). There is also helpful documentation and support for developing ImageJ plug-ins (20).

There are many different flavours of ImageJ available for download, the three principle flavours are ImageJ, ImageJ2 and FIJI. The current, stable version is known simply as ImageJ or ImageJ1. This includes a basic feature set for analysis and processing described but with no extra plug-ins. The ImageJDev project is developing version 2.0

of ImageJ, referred to as ImageJ2. It is a rewrite of ImageJ, but includes ImageJ1 with a compatibility layer, so that old-style plug-ins and macros can run. This project is still in the development phase but will eventually replace ImageJ. FIJI (FIJI Is Just ImageJ), is an ImageJ distribution with many plug-ins useful for image analysis in the life sciences, an automatic updater, and improved scripting capabilities. It hugely improves the capabilities for 3D imaging.

2.3.4.1 ImageJ Plug-ins

The 3D Object Counter (18) plug-in for ImageJ counts the number of 3D objects in a stack. This is useful to identify the regions of interest from the stack image of highlighted areas in MetaImage format. For each object located in 3D the following information is saved: integrity density, mean of the gray values, standard deviation of the gray values, minimum gray value, maximum gray value, median of the gray values, mean distance from the geometrical centre of the object to surface, standard deviation of the distance to surface, median distance to surface, centroid, centre of mass and the bounding box. The plug-in then generates an objects map.

The 3D ImageJ Suite (mcib) provides a host of plug-ins to enhance 3D capabilities of ImageJ (19). The suite includes filters, segmentation, mathematical morphology tools, analysis, mereo topology and a 3D region of interest manager. The 3D Roi Manager is of particular relevance as it provides a tool for locating ROIs and editing them, features of this suite can be incorporated by using the mcib_ core library.

2.4 Survey Findings

On balance VV slicer was deemed unsuitable as a platform for development owing largely to the limited support and existing plug-ins available although it did natively support MetaImage formats. The software was also found to be sometimes unstable and contained bugs. There is no functionality to view scans in three-dimensions.

ITK-SNAP also has limited support for development and does not support 3D imaging so would not be suitable as a development platform. The purpose of ITK-SNAP is for visualisation and is not designed to be an extensible piece of software.

3D Slicer has very good 3D visualisation features and is a good base for development however it was found not to support MetaImage files and could not handle overlays so was ruled out as a development platform.

Given these factors and the available plug-ins for ImageJ it was deemed a suitable platform for development. It can support MetaImage format with suitable plug-in and can visualise in 3D and is highly extensible. The greatest challenge lies in overlaying image stacks, but by merging channels of different colours this can be overcome.

3

Requirements and Analysis

This chapter will give an explanation of how the project fits into the larger work-flow of the analysis of μCT images for mouse embryos at the National Institute of Informatics (NII) and National Institute of Genetics (NIG). As well as a detailed problem statement and a list of requirements for the software to meet.

3.1 Scientific Pipeline

The pipeline for phenotyping of μCT images involves multiple stages of which annotation is one. The National Institute of Informatics has the following pipeline for analysis:

1. Data ingestion: NIG scientists upload raw μCT scan image data onto a server.
2. Data preperation: NII prepare data for processing, including copying the data to a specified place, and converting the data to a required format.
3. Registration: grouped data is registered, and the result is output as a mean image or similar.
4. Identification: suspicious areas are generated as a separate stack to be added as an overlay.
5. Feedback: all data (images and overlay) are sent to NIG for annotation.
6. Annotation: NIG scientists use the ImageJ-based software to annotate the data.
7. Evaluation: annotated data is sent back to NII for evaluation.

Within this pipeline the proposed software system is intended to improve the final stages of this workflow. Principally the annotation of data by NIG scientists but also the interpretation of this annotation at NII. Figure 3.1 shows this workflow schematically and that the seven step pipeline can be conceptually simplified into three essential stages, firstly data is acquired from NIG, second the data is processed by NII and finally the results of this processing is sent back to NIG to be checked.

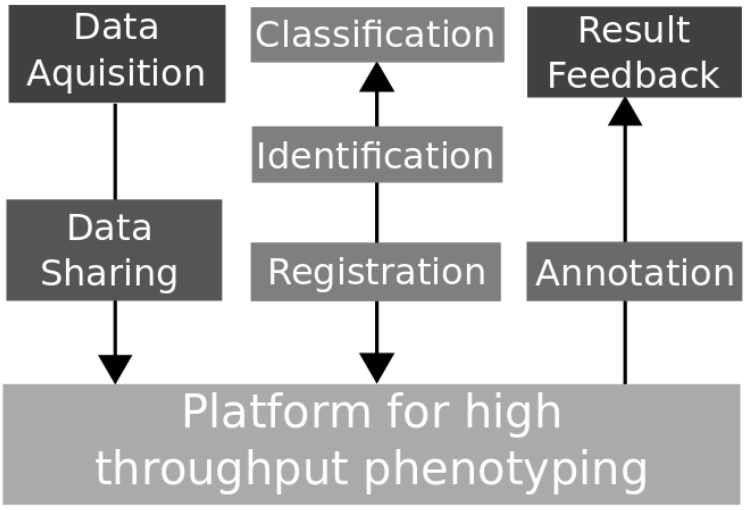

Figure 3.1: Phenotyping Pipeline - schematic showing the stages in phenotyping from data aquisition to result feedback. The first and third column are stages performed by NIG and the middle column shows stages performed at NII, registration, identification and classification maybe executed multiple times within the pipeline.

3.2 Problem Statement

The proposed software system seeks to improve the efficiency and ease with which scientists at NIG can provide feedback to NII on the phenotypic areas highlighted by the defective area algorithm. The process of identifying whether a candidate phenotype is indeed a novel phenotype is labour intensive, requiring an expert in the field of mouse embryological development to analyse many images. This annotation process is a bottleneck since it is the only part of the pipeline that cannot be fully automated and requires human input, this drastically slows the process of obtaining useful results from the scientific methodology.

The proposed system seeks to make it as easy as possible for scientists at NIG to provide feedback to NII and will thereby improve the whole work-flow. In addition the software will make it easier to analyse areas highlighted by the defective area algorithm. As there are many scientific groups working in The International Mouse Phenotyping Consortium it is imperative that findings from one research group are made available as soon as possible.

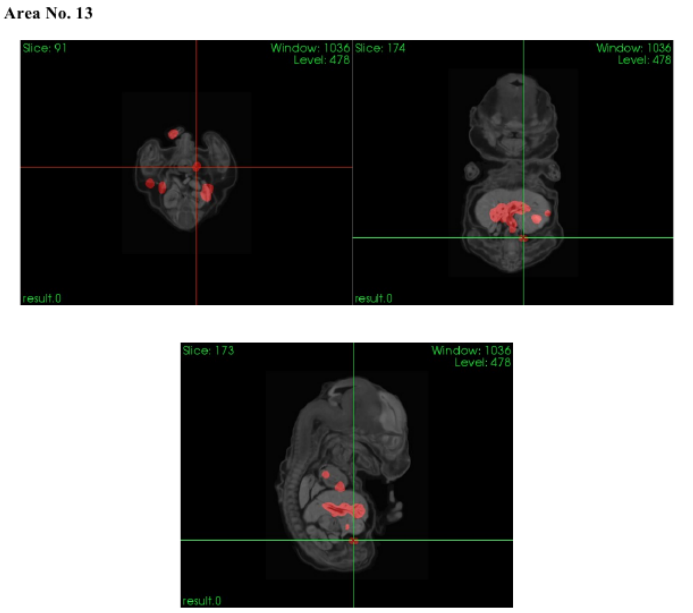

Figure 3.2: National Institute of Genetics Example Feedback - a demonstration of how feedback has previously been provided by NIG, showing an annotated text in red, this is one page within a large document.

The pipeline outlined in figure 3.1 has been implemented with a small data set and shown promising results (1). It is hoped by optimising the annotation and feedback within the pipeline the research can be scaled up for larger data sets that will be

produced at NIG.

The method previously employed to provide feedback was to send a set of images within a document and for scientists at NIG to annotate these images, this is demonstrated in figure 3.2. This has several key drawbacks, firstly it is time consuming to create such a document compiling many images of different slices and ROIs with sagittal sections, coronal (YZ projection image) and transverse (XZ projection image). Analysis of single images at NIG is not optimal and it would be beneficial to view other slices within a 3D stack rather than making an evaluation based on one slice. Furthermore, annotations on these images are not semantically related to a region of interest so must be reinterpreted at NII, this process of reinterpreting data is again lengthy and the data is not stored in a useful format. The proposed system should therefore have comments semantically associated to ROIs and uniquely identified to improve data analysis. It should be possible to navigate through slices of a ROI to make evaluation more accurate. Finally since the software will be used by biologists rather than computer scientists it should be intuitive, well documented and have a user friendly interface.

3.3 MoSCoW Requirements

The project requirements are divided into a prioritised list based on the MoSCoW method, which is an abbreviation for must, should, could and would haves. Must haves describes a requirement that must be satisfied in the final solution for the solution to be considered a success. Should have represents a high-priority item that should be included in the solution if it is possible. This is often a critical requirement but one which can be satisfied in other ways if strictly necessary. Could have describes a requirement which is considered desirable but not necessary. This will be included if time and resources permit. Would have represents a requirement that would ideally be implemented but is most likely beyond the scope of the project, it is often useful to include would haves to indicate how the software maybe developed in future.

3.3.1 Must Haves

- The ability to open 3D stacks and navigate through slices using appropriate file formats.

- The functionality to overlay Regions Of Interest (ROIs) onto each slice in a 3D stack.
- Ability to assign each volumetric ROI with a unique ID.
- The user must be able to label ROI by the ID as of significance or not.
- Data regarding ROIs must be saved and exported in a useful form.
- Compatibility for multiple operating systems.

3.3.2 Should Haves

- The user should be able to add comments which are semantically associated with corresponding ROIs.
- An annotation and classification GUI to easily input and edit comments and classify regions.
- Orthogonal viewing of slices displaying selected ROI.
- A feature to show and hide overlayed ROIs.

3.3.3 Could Haves

- A 3D viewer for 3D stack allowing transformations such as rotation and zoom.
- Ability to display ROI on 3D stack and use annotation and classification GUI on 3D viewer.
- Ability to update currently viewed ROI in orthogonal views and 3D view.

3.3.4 Would Haves

- An algorithm to suggest ROIs where the same comment maybe applicable (e.g. if both heart ventricles are separate ROIs).
- Prioritisation of ROIs that require feedback based on previous assignments.
- Guided navigation between ROIs based on their prioritisation to improve efficiency.

3.4 Use Case Summary

A use case is a list of steps which define interactions between a UML actor and a system, to achieve a goal. The actor may be a human or an external system. In the software system being developed there are two actors, an NIG scientist (NIGS) acting as a client and an NII researcher (NIIR) which interact with the system. Use cases are useful to understand all of the possible interactions that can occur with the system and what role each actor plays.

Table 3.1: Use Case Sumamry - list of use cases and short description. The actors NIGS and NIIR are acronyms for National Institute of Genetics Scientist and National Institute of Informatics Researcher respectively.

Use Case ID	Use Case Description	Primary Actor
UC1	Open plug-in - launch plug-in from ImageJ	NIGS/NIIR
UC2	Open mhd file - reads and opens MetaImage file	NIGS/NIIR
UC3	Merge channels - merge images by colour channels if images are already opened	NIGS/NIIR
UC4	View channels - hide or display colour channels in an image	NIGS/NIIR
UC5	Orthogonal view - display stack in orthogonal views	NIGS/NIIR
UC6	3D view - project stack in three dimensions	NIGS/NIIR
UC7	Identify ROIs - finds 3D ROIs in selected image	NIGS/NIIR
UC8	Select ROI - select an ROI from the list	NIGS/NIIR
UC9	Checkbox - select or deselect checkbox about selected ROI	NIGS
UC10	Add comments - add comments about selected ROI	NIGS
UC11	Import XML - import XML set of ROIs	NIGS/NIIR
UC12	Export XML - export set of ROIs to XML	NIGS

Table 3.1 gives a summary of how actors can interact with the system, this is represented schematically in figure 3.3 and a full list of use cases and descriptions can be found in appendix A. Since the project has an iterative approach the use cases are liable to change during design and implementation phases.

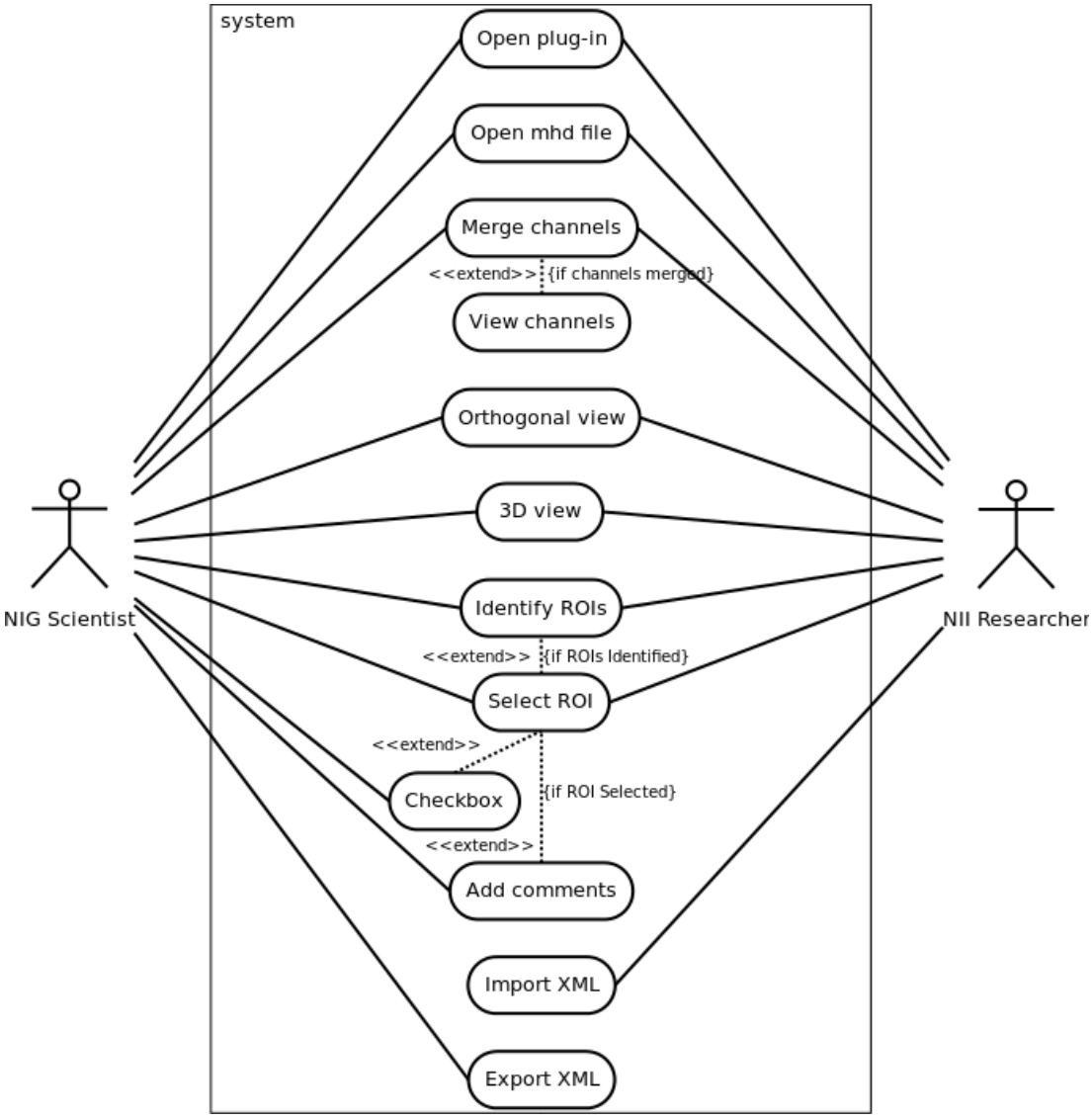

Figure 3.3: Use Case Diagram - a UML use case diagram representing the interaction of the NIG Scientist (client) and NII Researcher with the software system.

4

Design and Implementation

This chapter will describe the design of the software system and give an overview of the code implementation. The object-oriented design extends the work from the analysis and requirement gathering stages, and aims to translate the requirements into a design that can be implemented and tested. In particular the application logic and architecture will be discussed including the overall structure and features. The data storage method for the project will be addressed and implementation details given.

4.1 Project Name

Having conducted a survey of similar software in section 2.3 and decided to develop an ImageJ based plug-in a suitable name was conceived that aptly describes the plug-in. Considering the requirements in section 3.3 and the project goals in section 1.2 it was decided to call the plug-in 'Annotation ROI 3D'. This suitably captures the goals of the project since it is primarily focused on annotation in three dimensions. Although the project aims to improve a very specific scientific process the software system and therefore name, is designed to be generic and applicable to other projects with a long term goal of being available and as part of the FIJI package.

4.2 ImageJ Development

ImageJ plug-ins are powerful at extending the features of ImageJ and most of ImageJs built-in menu commands are implemented as plug-ins. ImageJ is built on Java, therefore plug-ins are implemented as Java classes. This gives ImageJ plug-ins use of all features

of the Java language, access to the full ImageJ API and use of all standard and third-party Java APIs in a plug-in. This is a significant advantage to ImageJ development and opens a wide range of possibilities of what can be achieved with an ImageJ plug-in.

The most common uses of plug-ins are filters performing some analysis or processing on an image or image stack and input output plug-ins for reading or writing to formats not natively supported by ImageJ. There are also many other things that ImageJ plug-ins can be used for, such as rendering graphics or creating extensions of the ImageJ graphical user interface (20).

ImageJ user plugins are located in a folder called plug-ins, which is a sub-folder of the ImageJ folder. Only class files in the plug-ins folder with at least one underscore in their name appear automatically in the Plugins menu. So 'Annotation ROI 3D' becomes 'Annotation_ ROI_ 3D' After version 1.20 of ImageJ it is also possible to create subfolders of the plug-ins folder and place plug-in files there. The sub-folders are displayed as sub-menus of ImageJs Plugins menu. To install a plug-in copy the .class file into the plugins folder or one of its sub-folders. The plug-in will appear in the plug-in menu, when ImageJ is next started the plug-in will be visible. It is also possible to add it to a menu and assign a shortcut to the plug-in using the Plugins/ Shortcut/ Install plugin... menu. In this case, the plug-in will appear in the menu without restarting ImageJ. Alternatively, with the source code of a plug-in, the plug-in may be compiled and run from within ImageJ. The plug-ins directory is specified by using the plugins.dir property. This can be done by adding an argument like -Dplugins.dir=c:\plugindirectory to the commandline calling ImageJ.

4.3 Application Logic and Architecture

The 'Annotation ROI 3D' plug-in uses a model-view-controller (MVC) software architecture where possible, this architecture separates the representation of information from the user's interaction with it. The model consists of application data, logic and functions. The view is the representation of that data such as in a list or table and the controller provides input, converting it to commands for the model or view. The idea of using this architecture is to make the code reusable and separate concerns.

Within the 'Annotation_ ROI_ 3D' project the code is split into three packages gui, io and methods. The gui package contains classes relevant to the graphical user

interface of the plug-in, this includes the Java swing components. This package is effectively the view since it controls what the user will see. The io package controls the input and output to the plug-in, including import and export to XML and opening MetaImage files. The functionality of the controller package is most closely met by the event handler class within the gui package. This is similar to the controller in the MVC architecture. Finally the methods file contains all of the core methods such as segmentation and updating ROIs this is where the application logic is so can be considered the model package.

4.4 Mock-ups

A mock-up, is a scale model of a design or device, used for teaching, demonstration, design evaluation, promotion, and other purposes. The most common use of mock-ups in software development is to create user interfaces that show the end user what the software will look like without having to build the software or the underlying functionality. At the beginning of the design phase mock-ups were created and feedback based upon the mock-up was acquired from the users. After feedback from National Institute of Informatics researchers and scientists at the National Institute of Genetics, the mock-ups were revised and a suitable user interface decided upon. Figures 4.1, 4.2 and 4.3 are a set of revised mock-ups to show how the plug-in will be accessed from ImageJ, the user interface for the plug-in and how the whole system may look.

The plug-in will be accessible via the Plug-ins menu in ImageJ as is shown in figure 4.1, and it will launch alongside the main ImageJ window. If ImageJ is closed the plug-in will also close since the plug-in will require some features of ImageJ and cannot be operated as a standalone module. After launching the plug-in a new window will appear entitled 'Annotation ROI 3D', as is shown in figure 4.2. The 'Annotation ROI 3D' window will contain all of the features of the plug-in to allow annotation of ROIs. The window is comprised of a list of identified ROIs on the left hand side (this will be empty upon loading), a set of buttons to operate certain features and an editable text box will store the annotation relating to each ROI. In addition a checkbox is including pertaining certain information about the selected region of interest. The exact nature of the feedback from the checkbox depends on the scientific survey being conducted,

but for the purposes of the National Institute of Genetics feedback it will most likely indicate whether a phenotype can be considered as novel.

The plug-in relies on ImageJ and images being open, the whole system may therefore look similar to that shown in figure 4.3. This shows how a selected ROI will be highlighted with a yellow border open in orthogonal views. A yellow border is suitable since it will not inhibit the ability to analyse the region.

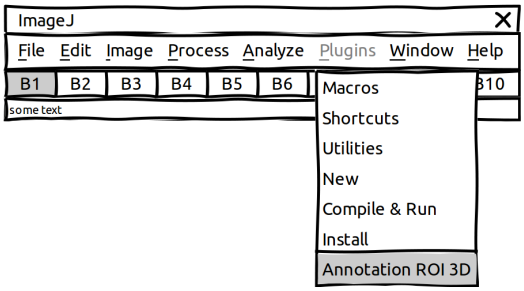

Figure 4.1: ImageJ Mock-up - a mock-up showing the user interface of ImageJ and how the Annotation ROI 3D plug-in will be accessed from within ImageJ.

4.5 Design & Features

This section will explain the design choices and features of the 'Annotation ROI 3D' plug-in. An explanation of each feature in the mock-up figure 4.2 and a brief description of how it can be implemented will be given. The 'Annotation ROI 3D' graphical user interface is designed with features in an order in which they would most probably be used from left to right and top to bottom, so the button in top left is to open files and bottom right is to export an XML document. Code examples in the following subsections are from EventAction.java file, which is listed in Appendix C.

4.5.1 Open

The open button is provided to open image stacks, it was decided to include this feature because ImageJ does not natively include support for MetaImage files. An external plug-in the MetaImage Reader/Writer can be installed to load and save images in MetaImage (.mhd/.mha) text-based tagged format.

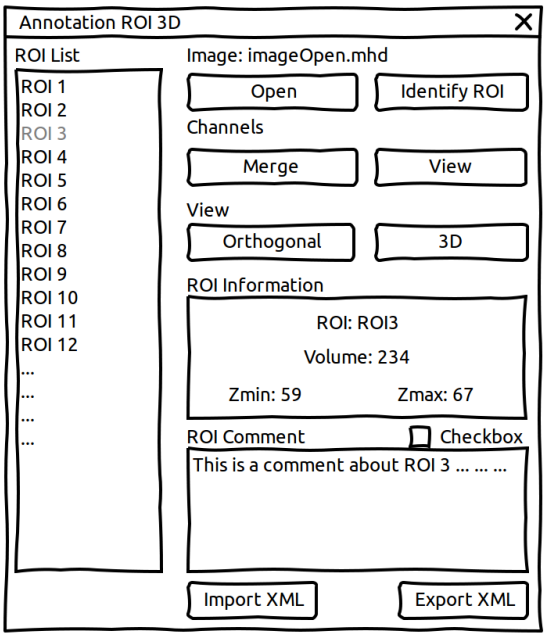

Figure 4.2: Annotation ROI 3D Mock-up - a mock-up to showing the user interface of Annotation ROI 3D, with a list of ROIs in the left hand column and buttons to perform certain functions.

The MetaImage Reader/Writer is included in 'Annotation ROI 3D' to give functionality to open MetaImage files. It was decided that it would be better to include the MetaImage Reader/Writer with the 'Annotation ROI 3D' plug-in rather than run both concurrently. This simplifies installation since only one plug-in will have to be installed if it is bundled, more importantly opening MetaImage files is a very common operation and it is easier to have a button for this. If the MetaImage Reader/Writer plug-in is installed separately then the user must navigate to File/Import/MetaImage to open an image stack. The button saves time especially opening multiple MetaImage files.

When the open button is clicked a dialog window will appear showing a navigable directory so the user can locate the MetaImage file to open. Upon selecting a file and clicking open the file will open as a seperate window. MetaImage support is implemented with the java files MetaImage_Reader.java and MetaImage_Writer.java, which are called on a button click. The open button opens the files and also converts them to 16-bit using an ImageJ macro.

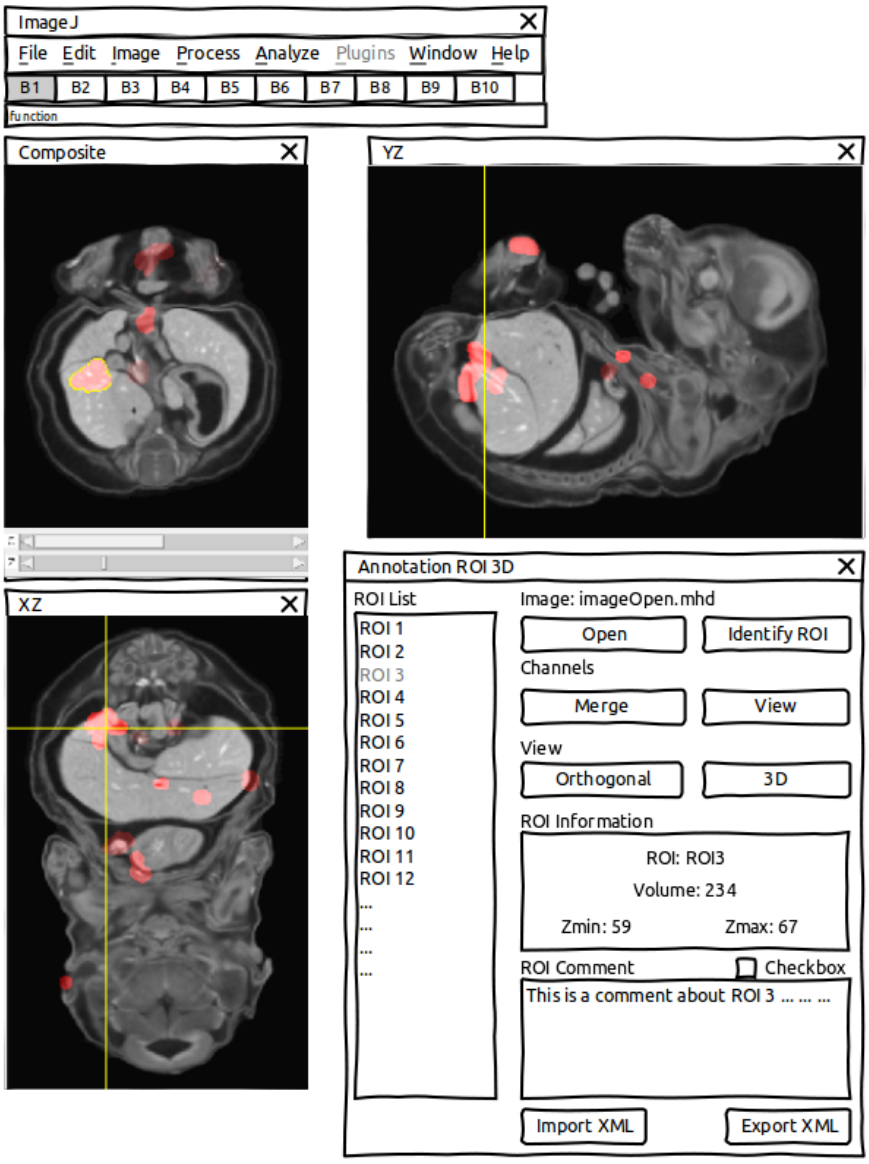

Figure 4.3: Annotation ROI 3D System Mock-up - a mock-up to show how Annotation ROI 3D plug-in maybe used with ImageJ to analyse a set of Images, in this mock-up the images are displayed in orthogonal views.

```
protected static void buttonOpenActionPerformed(java.awt.event.ActionEvent evt) {
   IJ.runPlugIn("org.annotationRoi3D.io.MetaImage_Reader", "");
   IJ.run("16-bit");
}
```

It is useful to have all opened image stacks at the same bit depth so they can be merged together, later in the workflow.

4.5.2 Identify ROI

The Identify ROI button is an essential part of the annotation system. For a selected overlay image stack it identifies all of the regions of interest in 3D and updates the list in the 'Annotation ROI 3D' GUI showing all of the found ROIs. This feature works with 16-bit grey-scale images and runs a segementation() method that analyses the stack to identify voxels between two threshold values. An array list of objects is created of type Object3D and an addImage() method adds the array list of Object3D to the 'Annotation ROI 3D' GUI.

If no images are open, segmentation is not possible so a pop-up dialog box explains this, and in the case that ROIs have already been identified for that image to stop the list being duplicated a dialog explains that as well.

```
protected static void buttonIdentifyROIActionPerformed(java.awt.event.ActionEvent evt) {
   if (WindowManager.getCurrentWindow() == null){
      IJ.showMessage("No Image", "No images are open");
   }
   else{
      if (org.annotationRoi3D.gui.InterfaceGUI.identified == true){
         IJ.showMessage("ROIs Identied", "ROIs already identified");
      }
      else{
         lblImage.setText("Image: " + WindowManager.getCurrentWindow().toString());
         ImagePlus beforeSegmentation = WindowManager.getCurrentImage();
         org.annotationRoi3D.methods.Segmentation.segmentation3D();
         beforeSegmentation.hide();
         Image.addImage();
         identified = true;
      }
   }
}
```

4.5.3 Merge

The merge feature allows for the ROI layer to be overlayed on a base layer by colour channels, by default the base layer will be grayscale and the ROI layer red. A composite image will then be formed, this can be implemented by using the ImageJ merge channels macro. The current window is then selected and to update the ROI when the list value is changed the showRoi() method is used.

```
protected static void buttonMergeChannelsActionPerformed(java.awt.event.ActionEvent evt) {
   IJ.run("Merge Channels...");
   imp = Image.getImage();
   imp = WindowManager.getCurrentImage();
   RoiDisplay.showRoi(true, false);
}
```

4.5.4 View

The view feature is useful for analysis of images because it enables overlayed layers to be displayed or hidden, the idea of this is to aid the analysis by removing the overlay. This is activated on a button click of view and is implemented similarly to the merge feature.

```
protected static void buttonViewChannelsActionPerformed(java.awt.event.ActionEvent evt) {
   if (WindowManager.getCurrentWindow() == null){
      IJ.showMessage("No Image", "No images are open");
   }
   else{
      IJ.run("Channels Tool...", "");
   }
}
```

4.5.5 Orthogonal

Provides an orthogonal view display of the current stack, if a stack displays sagittal sections, coronal (YZ projection image) and transverse (XZ projection image) will be displayed through the data-set. The extra generated views will be displayed in panels next to the original image. It is very useful for correct analysis of an ROI to see different views of the same region.

The intersection point of the three views follows the location of the mouse click and can be controlled by clicking and dragging in either the XY, XZ or YZ view. XY and

XZ coordinates are displayed in the title of the projection panels. The mouse wheel changes the screen plane in all three views. When a region of interest is selected from the list it will be updated in the main view and then must be navigated to with the cursor in the other views.

```
protected static void buttonOrthogonalViewsActionPerformed(java.awt.event.ActionEvent evt) {
   IJ.run("Orthogonal Views", "");
}
```

4.5.6 3D

The 3D button projects the stack in three dimensions as a rotating volume onto a plane using brightest-point projection. Brightest-point projection examines points along the rays, projecting the brightest point encountered along each ray. Brightest-point projection is especially useful in CT scans because it produces a semi-transparent effect, that highlights areas such as bone and other dense areas. The 3D projection is about the Y-axis and can be rotated along this axis, it is possible to change this axis in the code. There is an interval of 1 pixel between slices that make up the volume, this can also be adjusted by adjusting the slice. ImageJ projects the volume onto the viewing plane at each rotation angle increment, beginning with the volume rotated by initial angle and ending once the volume has been rotated by total rotation. This is set to rotate on a range from 0° to 360°. The 'Select None' command ensures that no ROI is selected so that the whole stack is projected rather than the Region currently being analysed.

```
protected static void buttonView3DActionPerformed(java.awt.event.ActionEvent evt) {
   IJ.run("Select None", "");
   IJ.run(WindowManager.getCurrentImage(), "3D Project...", "projection=[Brightest Point] axis=Y-
        ↪ Axis slice=1 initial=0 total=360 rotation=10 lower=1 upper=255 opacity=0 surface=100
        ↪ interior=50");
}
```

4.5.7 ROI Information

The ROI information area is implemented as an uneditable JTextPane, the pane displays information about the currently selected ROI. When the list value is changed the

Object3D value in the array list is found and get methods are used to find information about the object which are outputted as a HTML document.

```
RoiInformation.setText(
  "<html><table align=\"center\" style = \"font-family: Dialog; font-size: 12; color: #333333\">"
      ↪  +
  "<tr>"
  + "<td><b>ROI: </b></td><td>"+roiLabel+"</td>"
  + "</tr>"+
  "</table>"
  + "<table align=\"center\" style = \"font-family: Dialog; font-size: 12; color: #333333\">" +
  "<tr>"
  + "<td><b>Volume: </b></td><td>"+volume+"</td>"
  + "</tr>"
  +"</table>"
  + "<table align=\"center\" style = \"font-family: Dialog; font-size: 12; color: #333333\">" +
  "<tr>"
  + "<td><b>Zmin: </b></td><td>"+Integer.toString(currentZmin)+"</td><td><b>Zmax: </b></td><td>"+
      ↪ Integer.toString(currentZmax)+"</td>"
  + "</tr>"+
  "</table></html>"
  );
```

The 'roiLabel' variable gives the name of ROI that is selected along with its centroid in x, y and z coordinates and the ROI volume is also given in voxels. Most importantly for analysis, $Zmin$ and $Zmax$ are given that is the first slice in which the ROI appears and the last slice in which it can be seen.

4.5.8 ROI Comment & Checkbox

The ROI comment JEditorPane is an editable region that allows comments for the selected ROI to be given, and the checkbox allow for quantitative feedback for each ROI. Before any ROIs have been identified the comment box and checkbox is not editable. When ROIs are identified and the selected list value changes the comment and checkbox value is displayed.

Upon a list value change the code check if an object is selected by checking the variable *obj* is not null. If *obj* is indeed not null then its comment is set to the information in the *RoiComment* JEditorPane. Similarly the value of the checkbox is set in the *obj* depending on whether it is selected or not. The display is then updated using $RoiDisplay.showRoi(true, false)$ this displays the current slice for the selected ROI in the list, the ROI information in the uneditable JTextPane is updated on a list value

change. The checkbox and comment variables are loaded from *obj* and displayed in the GUI.

```
protected static void listValueChanged(javax.swing.event.ListSelectionEvent evt) {
   if (obj != null){
      obj.setComment(RoiComment.getText());
      if (chckbxCheckBox.isSelected()==true){
         obj.setName("true");
      }
      else{
         obj.setName("false");
      }
   }
   RoiDisplay.showRoi(true, false);

   //ROI information updated

   RoiComment.setEnabled(true);
   chckbxCheckBox.setEnabled(true);
   RoiComment.setText(comment);
   if (obj.getName().equals("true")){
      chckbxCheckBox.setSelected(true);
   }
   else{
      chckbxCheckBox.setSelected(false);
   }
}
```

The checkbox is currently an ambiguous component that has been included to give a quantitative feedback about a selected ROI, the exact nature of the feedback depends on what is required by the scientists at NII. It maybe used to indicate that a selected ROI is a novel phenotype. The checkbox is implemented in a similar way to the ROI comment, when a list value is changed the current checkbox value is loaded and set in the object, then a new value is selected and a new checkbox value loaded.

4.5.9 Import XML

Import XML is a button that will import the relevant checkbox values and comments for a set of ROIs. It is important to note that this feature only works after a set of ROIs have been identified. If the names and coordinates of the identified ROIs matches those in the imported XML then the comments and checkbox for the identified ROIs will be loaded. This is implemented by checking that two arrays are equal, if they are equal then the data is imported.

```
String[] array0 = new String[model.getSize()];
for (int i = 0; i < model.getSize(); i++){
   array0[i] = model.get(i).toString(); // creates array for identified ROIs
}

String[] array1 = new String[nList.getLength()];
for (int i = 0; i < nList.getLength(); i++){
   Node nNode = nList.item(i);

   if (nNode.getNodeType() == Node.ELEMENT_NODE) {
      Element eElement = (Element) nNode;
      array1[i] = eElement.getElementsByTagName("name").item(0).getTextContent().substring(11);
      //creates array for ROIs in XML file
   }
}

if(Arrays.equals(array0, array1) == false){
   IJ.showMessage("Import XML", "Data sets to not match");
}
if (Arrays.equals(array0, array1) == true){
   for (int temp = 0; temp < nList.getLength(); temp++) {
      Node nNode = nList.item(temp);
      if (nNode.getNodeType() == Node.ELEMENT_NODE) {
         Element eElement = (Element) nNode;
         objects3D.getObject(temp).setComment(eElement.getElementsByTagName("comment").item(0).
             ↪ getTextContent());
         objects3D.getObject(temp).setName(eElement.getElementsByTagName("checkbox").item(0).
             ↪ getTextContent());
         //sets the comment value and checkbox for identified ROIs
      }
   }
}
```

4.5.10 Export XML

Export XML is a button that exports the identified ROI set with comments and checkbox values and generates an XML file. The JFileChooser dialog is used to find an appropriate directory to save the file in. This feature uses the Java XML parser and Document Factory.

Exportation is implemented by looping through each object in the Objects3D population for each Object3D certain variables are found and converted to a string the result is streamed as a file chosen by the JFileChooser.

```
DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();
DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
```

```
// root elements
Document doc = docBuilder.newDocument();
Element rootElement = doc.createElement("ROIs");
doc.appendChild(rootElement);

for (int i = 0; i < objects3D.getNbObjects(); i++){
   //System.out.println(objects3D.getObject(i).toString());

   Element ROI = doc.createElement("ROI");
   rootElement.appendChild(ROI);

   //X-coordinate elements
   Element name = doc.createElement("name");
   name.appendChild(doc.createTextNode(objects3D.getObject(i).toString()));
   ROI.appendChild(name);

   Element x = doc.createElement("x");
   x.appendChild(doc.createTextNode(String.valueOf(objects3D.getObject(i).getCenterX())));
   ROI.appendChild(x);

   // Y-coordinate elements
   Element y = doc.createElement("y");
   y.appendChild(doc.createTextNode(String.valueOf(objects3D.getObject(i).getCenterY())));
   ROI.appendChild(y);

   // Z-coordinate elements
   Element lastname = doc.createElement("z");
   lastname.appendChild(doc.createTextNode(String.valueOf(objects3D.getObject(i).getCenterZ())));
   ROI.appendChild(lastname);

   // checkbox elements
   Element checkbox = doc.createElement("checkbox");
   checkbox.appendChild(doc.createTextNode(String.valueOf(objects3D.getObject(i).getName().
        ↪ toString())));
   ROI.appendChild(checkbox);

   // comment elements
   Element comment = doc.createElement("comment");
   comment.appendChild(doc.createTextNode(String.valueOf(objects3D.getObject(i).getComment().
        ↪ toString())));
   ROI.appendChild(comment);
}

// write the content into xml file
TransformerFactory transformerFactory = TransformerFactory.newInstance();
Transformer transformer = transformerFactory.newTransformer();
DOMSource source = new DOMSource(doc);

StreamResult result = new StreamResult(new File(ExportFileChooser.ExportFile.getAbsolutePath()));
```

4.6 Storage Representation

The plug-in saves each ROI as an Object3D whilst 'Annotation ROI 3D' is open. For analysis comments and checkbox values can be exported and imported in eXtensible Markup Language (XML) format. It was decided that a human and machine readable format that could be parsed would be a useful way to format the data. In addition XML is simple and can be edited with most text editors, so comments maybe amended in the file.

The root element in the XML document is ROIs with child elements of ROI. Each ROI has in turn its own child elements, name, x, y, z, checkbox and comment. In this way all of the essential information about an ROI is stored. This system is advantageous because it is easy to add elements and to modify the XML to include more data fields if necessary in further development. The following is an example showing two ROI regions in XML format.

```
<ROIs>
 <ROI>
  <name>Object3D : 1 (135,247,176)</name>
  <x>134.76534337211956</x>
  <y>247.23568332192562</y>
  <z>176.3750855578371</z>
  <checkbox>false</checkbox>
  <comment></comment>
 </ROI>
 <ROI>
  <name>Object3D : 2 (74,164,98)</name>
  <x>74.31250205680061</x>
  <y>164.27376838779742</y>
  <z>97.94112613946754</z>
  <checkbox>false</checkbox>
  <comment></comment>
 </ROI>
</ROIs>
```

5

Testing

This chapter will discuss the software testing strategy as well as found bugs that were not able to be fixed within the time constraints of the project. The testing strategy applied was functional and a list of bugs that were identified after testing has been included.

5.1 Functional Testing

Functional testing verifies that each function of the software application operates in conformance with the software requirements. In functional testing the source code implementation is not a priority rather the high-level outcomes, this is known as black box testing. The functionality of the system is tested by providing an appropriate input, verifying the output and comparing the actual results with the expected results. The main functions, usability, accessibility and error conditions were all tested. Functional testing was first applied by simulating the workflow of an NIG scientist to check that each feature worked correctly. A typical workflow would be to open two MetaImage files, identify regions of interest of the overlay file, merge channels and select ROIs to make comments. The testing resulted in the discovery of a bug that after merging and creating a composite image, when the list value changes the view in the composite was not updated. This reason for this is that when creating a composite image the stack becomes a hyper-stack so a different command is needed to set the slice.

In addition to testing a typical workflow, each feature was tested in turn with a variety of inputs to check for any unexpected results. Testing was conducted with Im-

ageJ on a 32-bit version of Windows and 64-bit Linux operating system with consistent results between the two. A list of bugs found by this testing methodology can be seen in section 5.2.

5.1.1 Open

To test the open function, firstly an expected input was provided as a MetaImage file, as expected the file was opened. Opening of multiple files consecutively was also not a problem. When trying to open an image file of a different file format the function also opened the file for JPEG, TIFF, PNG and the other common image file types. When trying to open a file that is not an image an error occurs that there is no such file or directory as directory/name.mhd, this behaviour is expected for the open feature and a suitable error is displayed. The testing did yield an undesirable outcome, if the open button is clicked but the file chooser dialog is cancelled, a second window appear saying 'There are no images open'.

5.1.2 Identify ROI

With an expected input the Identify ROI achieved acceptable functionality and was able to find all ROIs and show a list, that could be navigated. Several other file formats were tested to see the outcome of using the identify ROIs feature, when using of a 5D stack the image became black for all slices, and one ROI was found. If identifying ROIs for a image that is not a stack the feature will not work and no ROIs will be identified, a useful error message should be provided in such a case.

5.1.3 Merge & View

The merge function worked well for expected inputs after functional testing it emerged that the feature would work even if only one image file was open. This is not desired behaviour and the merge feature should only be accessible with multiple images open. The same problem was applicable for view but since they are both ImageJ functions there use is not analysed in detail.

5.1.4 Orthogonal & 3D

The orthogonal and 3D views worked as expected, when trying with a 3D stack both image views provided the expected output. When trying to open a two-dimensional image in orthogonal views an error is generated, when the 3D view button is selected a three dimensional projection is generated. This is not the desired outcome of this function. With an ROI selected the 3D view would also only project the ROI, this was overcome by running an ImageJ macro to not select regions.

5.1.5 Comment & Checkbox

The comment and checkbox functions worked well. An error was observed when writing to XML for comments and checkbox values if the list value had not been changed before exporting to XML. This error occurred because of the implementation of the comment and checkbox function, values would only be written to the Objects3D array on a list value change. To overcome this error the comment and checkbox functions were set in the Ojbects3D array before they are exported as an XML file.

5.1.6 ImportXML & Export XML

Import XML and Export XML showed good results when using the expected data set. When trying to import an XML data set to a set of ROIs that do not match the data set an error occurs, this is the correct functionality, when trying to open a file that is not in XML format the ROI list is not generated. It would be useful to provide an error message for such instances.

5.2 Identified Bugs

The following is a list of bugs or areas where the software does not meet ideal functionality that could be improved in the future.

- Close open dialog of MetaImage and another dialog appears ‘There are no images open’, this should not appear.
- Identify ROI should only work for stacks of correct format, otherwise give an error.

- Merge dialog opens with only one image file open.
- ROI yellow border is not displayed in orthogonal views.
- When identifying ROIs the generated image has different intensity of ROI because of the bit conversion method.

6

Evaluation & Conclusions

This section will evaluate the successes of the project and address each of the goals outlined in section 1.2 and requirements described in section 3.3. A critical evaluation of the results of the project and areas for development and future work are considered.

6.1 Evaluation

6.1.1 Review of Project Goals

- Viewing of 3D stacks and navigation through stack - a fundamental goal of this project was to provide a platform to view 3D stacks and navigate through those stacks in MetaImage file format, for accurate analysis of μCT scans. This primary functionality was met by using features of the existing software ImageJ. To open the required MetaImage format, a MetaImage_ Reader plug-in was used. This was found to be a good solution for viewing of 3D stacks.

- Overlay ROIs onto 3D stacks - the ability to overlay multiple stacks onto each other was a goal of this project. The ImageJ API provided functionality for this by merging channels.

- Semantically relate ROIs with unique IDs - the identify ROIs function automatically identifies ROIs and generates a list. When an ROI is selected from the viewer automatically navigates to that ROI and draws a yellow border, each ROI is given a unique ID, based on its number within the image and its co-ordinates in x, y and z axis.

- Provide a GUI for adding comments to IDs - the 'Annotation ROI 3D' has a user friendly GUI that can be used to easily add comments for the ROI selected from the list.
- Provide a quantitative and general feedback mechanism for each ROI - the 'Annotation ROI 3D' has a checkbox function to provide feedback about the significance of a region, this is easy to use and provides useful quantitative feedback.

6.1.2 Future Work

The project achieved the core functionality given in the MoSCoW requirements, all of the must haves and should haves were implemented. It was not possible within the time constraints of this project to implement the would haves and two of the could have requirements. Therefore these areas could be developed upon in future work:

- An algorithm to suggest ROIs where the same comment maybe applicable (e.g. if both heart ventricles are separate ROIs).
- Prioritisation of ROIs that require feedback based on previous assignments.
- Guided navigation between ROIs based on their prioritisation to improve efficiency.
- Display ROI on 3D stack and use annotation and classification GUI on 3D viewer.
- Update currently viewed ROI in orthogonal views and 3D view.

In addition with feedback from the National Institute of Genetics several areas for development were suggested. The National Institute of Genetics is moving towards a more collaborative annotation process whereby experts comment on their area of expertise, for example a specialist would comment on the heart and another about the liver. The results of the feedback would be shared on a server. A pipeline would automatically segment and identify ROIs in the images and then display the results via a web interface, a number of experts could then annotate and assign priority levels to ROIs, so that you could display only ROIs at a certain priority level or better. The software as it stands works as a client-side self contained package, however in future all the data maybe on a server and the outputted XML files also on the server. For

this the plug-in would have to be modified and a user management and history feature implemented.

It has also been suggested that it would be useful to draw ROIs on the software and highlight regions, although at this stage it is not clear whether this would be desirable given the limitations of selecting regions in 3D.

6.2 Conclusions

This software meets the key functionality that it set out to solve and will greatly improve a niche scientific process that cannot be solved by any other software currently available. Within the pipeline of research being conducted at the National Institute of Informatics and National Institute of Genetics it should greatly improve the efficiency with which results can be obtained. In such a rapidly advancing scientific field it is of principle importance that results can be published quickly. The use of the software has been tested within the research pipeline and has been trialled at the National Institute of Genetics showing good results. With some modification to make it more generic the software can also be made available as an ImageJ plug-in.

Given more time there are certain features that should be taken forward to enhance the analysis and to remove bugs that have been identified. It would be especially useful to work on the 3D capabilities of the plug-in and server-side implementation. The software meets all of the goals intended and is implemented with documentation in such a way that biologists and non-specialist users alike will be able to have access to and use the software.

References

[1] Sharmili Roy, Xi Liang, Asanobu Kitamoto, Masaru Tamura, Toshihiko Shiroishi, and Michael S. Brown. **Phenotype Detection in Morphological Mutant Mice using Deformation Features**. *Medical Image Computing and Computer Assisted Intervention (MICCAI'13)*, Sep 2013. 1, 2, 5, 17

[2] International Mouse Knockout Consortium, Francis S. Collins, Janet Rossant, and Wolfgang Wurst. **A mouse for all reasons.** *Cell*, **128**[1]:9–13, jan 2007. 1

[3] R. H. Waterston, K. Lindblad-Toh, E. Birney, J. Rogers, J. F. Abril, P. Agarwal, R. Agarwala, R. Ainscough, and Et Al. **Initial sequencing and comparative analysis of the mouse genome**. *Nature*, **420**:520–562, Dec 2002. 2

[4] M.D. Wong, A.E. Dorr, J.R. Walls, J.P. Lerch, and R.M. Henkelman. **A novel 3D mouse embryo atlas based on micro-CT.** *Development*, **139**[17]:3248–56, 2012. 2, 5, 6

[5] Lerch JP Bhattacharya S Schneider JE Henkelman RM Sled JG. Zamyadi M, Baghdadi L. **Mouse embryonic phenotyping by morphometric analysis of MR images**. *Physiol Genomics*, pages 89–95, 2010. 6

[6] Jon O. Cleary, Marc Modat, Francesca C. Norris, Anthony Price, Sujatha A. Jayakody, Juan Pedro Martinez-Barbera, Nicholas D. E. Greene, David J. Hawkes, Roger J. Ordidge, Peter J. Scambler, Sbastien Ourselin, and Mark F. Lythgoe. **Magnetic resonance virtual histology for embryos: 3D atlases for automated high-throughput phenotyping.** *NeuroImage*, **54**[2]:769–778, 2011. 6

[7] Brian J. Nieman, Ann M. Flenniken, S. Lee Adamson, R. Mark Henkelman, and John G. Sled. **Anatomical phenotyping in the brain and skull of a mutant mouse by magnetic resonance imaging and computed tomography.** *Physiological genomics*, **24**[2]:154–162, jan 2006. 6

[8] Belma Dogdas, David Stout, Arion F Chatziioannou, and Richard M Leahy. **Digimouse: a 3D whole body mouse atlas from CT and cryosection data**. *Physics in Medicine and Biology*, **52**[3]:577, 2007. 6

[9] X. Josette Chen, Natasa Kovacevic, Nancy J. Lobaugh, John G. Sled, R. Mark Henkelman, and Jeffrey T. Henderson. **Neuroanatomical differences between mouse strains as shown by high-resolution 3D MRI**. *NeuroImage*, **29**[1]:99 – 105, 2006. 6

[10] John T. Johnson, Mark S. Hansen, Isabel Wu, Lindsey J. Healy, Christopher R. Johnson, Greg M. Jones, Mario R. Capecchi, and Charles Keller. **Virtual Histology of Transgenic Mouse Embryos for High-Throughput Phenotyping**. *PLoS Genetics*, **2**[4]:e61+, apr 2006. 6

[11] Zheng Xia, Xishi Huang, Xiaobo Zhou, Youxian Sun, V. Ntziachristos, and S. Wong. **Registration of 3-D CT and 2-D Flat Images of Mouse via Affine Transformation**. *Trans. Info. Tech. Biomed.*, **12**[5]:569–578, sep 2008. 6

[12] K. A. Powell and D. Wilson. **3-Dimensional Imaging Modalities for Phenotyping Genetically Engineered Mice**. *Veterinary Pathology Online*, **49**[1]:106–115, 2012. 6

[13] M. D. Abramoff, P. J. Magelhaes, and S. J. Ram. **Image processing with ImageJ**. *Biophotonics Int*, **11**[7]:36–42, 2004. 6, 10

[14] P. Seroul and D. Sarrut. **VV: a viewer for the evaluation of 4D image registration**. Jul 2008. 8

[15] S. Rit, R. Pinho, V. Delmon, M. Pech, G Bouilhol, J. Schaerer, B. Navalpakkam, J. Vandemeulebroucke, P. Seroul, and D. Sarrut. **VV, a 4D slicer**. In *Proceedings of the Fourth International Workshop on Pulmonary Image Analysis*, pages 171 – 175, Toronto, Canada, 09/2011 2011. 8

[16] Paul A. Yushkevich, Joseph Piven, Heather Cody Hazlett, Rachel Gimpel Smith, Sean Ho, James C. Gee, and Guido Gerig. **User-Guided 3D Active Contour Segmentation of Anatomical Structures: Significantly Improved Efficiency and Reliability**. *Neuroimage*, **31**[3]:1116–1128, 2006. 9

[17] Steve Pieper, Michael Halle, and Ron Kikinis. **3D SLICER**. pages 632–635, 04 2004. 9

[18] S. Bolte and F. P. Cordelières. **A guided tour into subcellular colocalization analysis in light microscopy.** *Journal of microscopy*, **224**[Pt 3]:213–232, dec 2006. 12, 13

[19] Jean Ollion, Julien Cochennec, Franccedilois Loll, Christophe Escudeacute;, and Thomas Boudier. **TANGO: A Generic Tool for High-throughput 3D Image Analysis for Studying Nuclear Organization.** *Bioinformatics*, 2013. 12, 13

[20] Werner Bailer. **Writing ImageJ PlugIns A Tutorial**, July 2006. 12, 23

Appendix A

Use Case Listing

Use Case ID	UC1
Use Case Name	Open plug-in
Description	launch plug-in from ImageJ
Primary Actors	NIGS/NIIR
Secondary Actors	
Preconditions	ImageJ must be running
Main flow	1. Select plugins on ImageJ file menu 2. Select 'Annotation ROI 3D'
Post conditions	Plug-in is opened and GUI window appears, if ImageJ window is closed the plug-in will also close

Table A.1: Use Case 1: Open Plug-in

Use Case ID	UC2
Use Case Name	Open mhd file
Description	reads and opens MetaImage file
Primary Actors	NIGS/NIIR
Secondary Actors	
Preconditions	ImageJ must be running and Annotation ROI 3D plug-in
Main flow	1. Select Open from Annotation ROI 3D window 2. Navigate to the directory containing MetaImage file. 3. Select MetaImage file and click open
Post conditions	MetaImage file Annotation ROI 3D Window and ImageJ are open

Table A.2: Use Case 2: Open mhd file

Use Case ID	UC3
Use Case Name	Merge channels
Description	merge images by colour channels creates a composite image of multiple channels
Primary Actors	NIGS/NIIR
Secondary Actors	
Preconditions	ImageJ must be running and Annotation ROI 3D plug-in with more than one image opened
Main flow	1. Select Merge from Annotation ROI 3D window 2. Select a colour and from a drop down list of open images select an image. 3. Select a second colour and from a drop down list of open images select a second image. 4. Select create a composite checkbox and click ok.
Post conditions	Multiple MetaImage files are merged into a single image file called composite.

Table A.3: Use Case 3: Merge channels

Use Case ID	UC4
Use Case Name	View channels
Description	hide or display colour channels in an image
Primary Actors	NIGS
Secondary Actors	NIIR
Preconditions	ImageJ must be running and Annotation ROI 3D plug-in with composite image opened.
Main flow	1. Select View from Annotation ROI 3D window 2. Select the checkbox next to each channel to hide or show the channel 3. Select ok
Post conditions	Colour channel is shown or hidden from composite image.

Table A.4: Use Case 4: View channels

Use Case ID	UC5
Use Case Name	Orthogonal view
Description	display stack in orthogonal views
Primary Actors	NIGS
Secondary Actors	NIIR
Preconditions	ImageJ must be running and Annotation ROI 3D plug-in with an open stack.
Main flow	1. Select image to show in orthogonal views 2. Select Orthogonal from Annotation ROI 3D window
Post conditions	Stack displays sagittal sections, coronal (YZ projection image) and transverse (XZ projection image) will be displayed through the data-set.

Table A.5: Use Case 5: Orthogonal View

Use Case ID	UC6
Use Case Name	3D view
Description	project stack in three dimensions
Primary Actors	NIGS/NIIR
Secondary Actors	
Preconditions	ImageJ must be running and Annotation ROI 3D plug-in with an open stack.
Main flow	1. Select image to show in 3D view 2. Select 3D from Annotation ROI 3D window
Post conditions	A seperate 3D image will be displayed with a navigation bar of the stack

Table A.6: Use Case 6: 3D View

Use Case ID	UC7
Use Case Name	Identify ROIs
Description	finds 3D ROIs in selected image
Primary Actors	NIGS/NIIR
Secondary Actors	
Preconditions	ImageJ must be running and Annotation ROI 3D plug-in with an open stack.
Main flow	1. Select image to identify ROIs 2. Select Identify ROIs
Post conditions	a list of found ROIs will appear in the scroll pane on left-hand side of Annotation ROI 3D plug-in

Table A.7: Use Case 7: Identify ROIs

Use Case ID	UC8
Use Case Name	Select ROI
Description	finds 3D ROIs in selected image
Primary Actors	NIGS/NIIR
Secondary Actors	
Preconditions	ImageJ must be running and Annotation ROI 3D plug-in with an open stack with found ROIs.
Main flow	1. Select an ROI from the list
Post conditions	ROI will be highlighted and information displayed along with ROI.

Table A.8: Use Case 8: Select ROI

Use Case ID	UC9
Use Case Name	Checkbox
Description	select or deselect checkbox about ROI
Primary Actors	NIGS
Secondary Actors	
Preconditions	ImageJ must be running and Annotation ROI 3D plug-in with an open stack with found ROIs and an ROI selected.
Main flow	1. Select an ROI from the list 2. Select checkbox to tick or untick
Post conditions	Checkbox will remain ticked or unticked

Table A.9: Use Case 9: Checkbox

Use Case ID	UC10
Use Case Name	Add comments
Description	add comments about selected ROI
Primary Actors	NIGS
Secondary Actors	
Preconditions	ImageJ must be running and Annotation ROI 3D plug-in with an open stack with found ROIs and an ROI selected.
Main flow	1. Select an ROI from the list 2. In comments box type comments
Post conditions	Comments will appear in Annotation ROI 3D window and will be saved

Table A.10: Use Case 10: Add comments

Use Case ID	UC11
Use Case Name	Import XML
Description	import XML set of ROIs
Primary Actors	NIIR
Secondary Actors	NIGS
Preconditions	ImageJ must be running and Annotation ROI 3D plug-in with an open stack with a stack opened and ROIs identified
Main flow	1. Select Import XML 2. Navigate to directory containing XML file 3. Select XML file and click open
Alternate flow	1. Select Import XML 2. Navigate to directory containing XML file 3. Select XML file and click open 4. Data set does not match identified ROIs
Post conditions	Comments and checkbox status will be loaded for ROI set

Table A.11: Use Case 11: Import XML

Use Case ID	UC12
Use Case Name	Export XML
Description	export set of ROIs to XML
Primary Actors	NIGS
Secondary Actors	NIIR
Preconditions	ImageJ must be running and Annotation ROI 3D plug-in with an open stack with a stack opened and ROIs identified
Main flow	1. Select Export XML 2. Navigate to directory to save XML file 3. Input name for XML file and click save
Post conditions	An XML file will be generated.

Table A.12: Use Case 12: Export XML

Appendix B

User Manual

This is a written guide with screen captures and associated images to demonstrate how to set-up and use the 'Annotation ROI 3D' plug-in in ImageJ. This user manual demonstrates the software being used in a Microsoft Windows operating system but the information contained is equally applicable to alternative operating systems.

B.1 Getting Started

To get started using 'Annotation ROI 3D' the following packages are required.

- ImageJ - software
- Annotation_ROI_3D.jar - the plug-in
- imagescience.jar - software library
- mcib3d-core.jar - software library

B.1.1 Annotation ROI 3D Installation

Self contained versions of ImageJ with the 'Annotation ROI 3D' plug-in bundled will be distributed along with this report and made available at the National Institute of Informatics and National Institute of Genetics. These versions will be ready out of the box and can be launched from the appropriate directory. Follow these instructions to easily install ImageJ with 'Annotation ROI 3D' on your system.

1. Visit this webpage and open the link of the ImageJ repository appropriate for your operating system:`https://github.com/wgrimes`.
2. Select the 'Download ZIP' option within the webpage.
3. Wait for the download to complete and extract the ZIP file.
4. Launch ImageJ by clicking the ImageJ executable within the directory.

B.1.2 Manual Installation and Configuration

Information about how to manually install and configure the plug-in is included here for completeness.

B.1.2.1 ImageJ

Follow these instructions to install ImageJ on your system.

1. Visit this webpage, `http://rsbweb.nih.gov/ij/download.html`
2. Select the correct version of ImageJ for the operating system, for example Windows 64 bit and download the appropriate file.
3. When the download is complete extract the contents of the file, if using Windows run the setup.exe file and choose an appropriate location to install to, see figure B.1. If installing on linux the filed downloaded will be a zip file that should be extracted when downloaded.

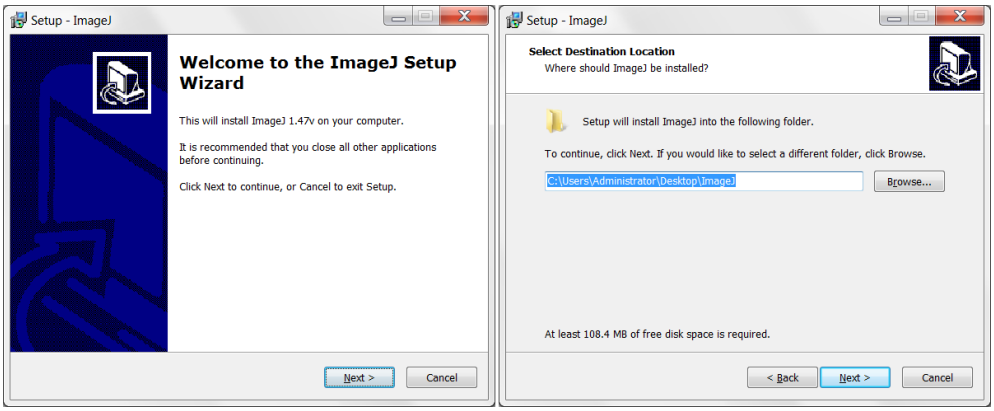

Figure B.1: ImageJ Windows Install - installation wizard for installing ImageJ in Windows, launched from downloaded setup.exe file.

B.1.2.2 Annotation ROI 3D

Follow these instructions to download Annotation ROI 3D, please note it will not work without imagescience.jar and mcib3d-core.jar files.

1. Visit this webpage, `https://github.com/wgrimes/fiji/blob/master/plugins/Annotation_ROI_3D-1.0.0.jar`
2. Select Raw and download the Annotation_ROI_3D-1.0.0.jar file.
3. When the download is complete navigate to the ImageJ directory and select the plug-ins folder, place the Annotation_ROI_3D-1.0.0.jar in the plug-ins folder, see figure B.2.

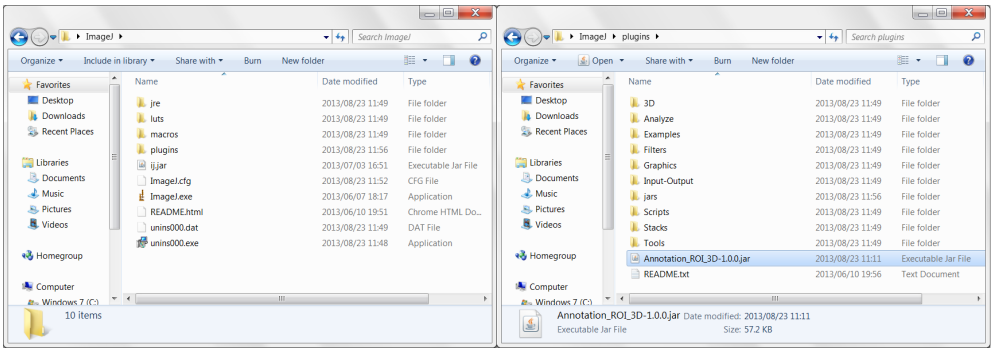

Figure B.2: ImageJ Main Directory and Plug-ins Folder - the main ImageJ directory and the plugins folder with the 'Annotation_ROI_3D-1.0.0.jar' file.

B.1.2.3 imagescience.jar & mcib3d-core.jar

Follow these instructions to download imagescience.jar and mcib3d-core.jar libraries.

1. Visit this webpage, `https://github.com/wgrimes/fiji/blob/master/jars/mcib3d-core2.6.jar` and `https://github.com/wgrimes/fiji/blob/master/jars/imagescience.jar`
2. Select Raw and download each of the jar files.
3. When the downloads are complete navigate to the ImageJ directory and select the jars folder within the plug-ins folder, place the imagescience.jar and mcib3d-core.jar files in the jars folder, see figure B.3.

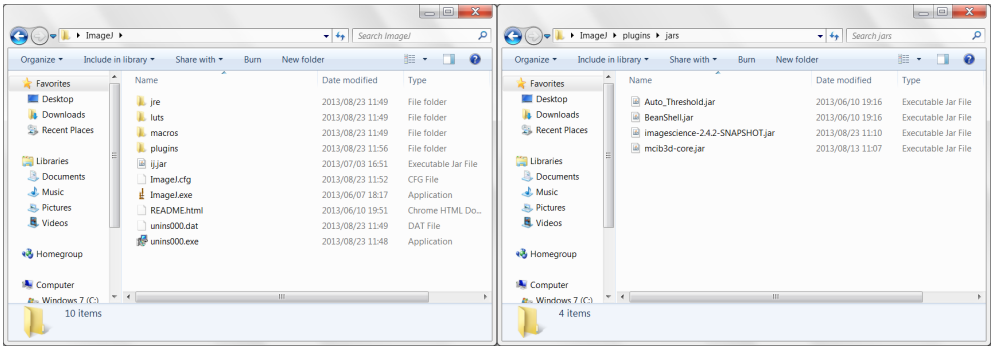

Figure B.3: ImageJ Main Directory and Jars Folder - the main ImageJ directory and the jars folder with the imagescience.jar and mcib3d-core.jar files.

B.2 Annotation ROI 3D

B.2.1 Launch the 'Annotation ROI 3D' Plug-in

1. From the main ImageJ directory launch the ImageJ application executable, ImageJ.exe in Windows. This will open an ImageJ window.
2. In the ImageJ file menu navigate to Plugins/Annotation ROI 3D, this will launch the 'Annotation ROI 3D' window, see figure B.4.

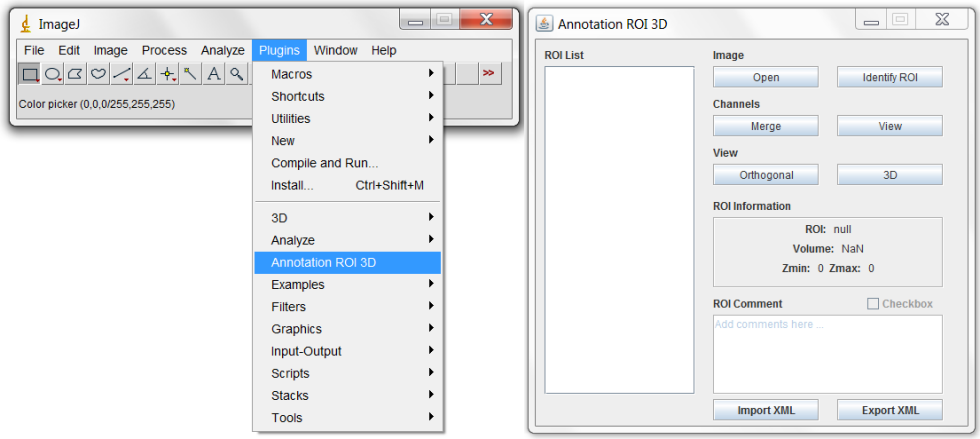

Figure B.4: Launching 'Annotation ROI 3D' From ImageJ - how to open the plug-in from the ImageJ window.

B.2.2 Open MetaImage Files

This features allows images and stacks with '.mhd' extension to be opened with the plug-in.

1. Select the Open button from the 'Annotation ROI 3D' window, this will launch a new window.
2. Select a file with file extension '.mhd' and click open. This will open the image stack, to open images of different file types, select File/Open from the ImageJ menu.

B.2.3 Identify ROIs

This function identifies all ROIs within an image based on the voxel values and creates a list in the GUI.

1. Select the window containing ROIs you wish to identify.
2. Select Identify ROI button from the 'Annotation ROI 3D' window. The window will relaunch and a list of identified ROIs will be displayed, see figure B.5.

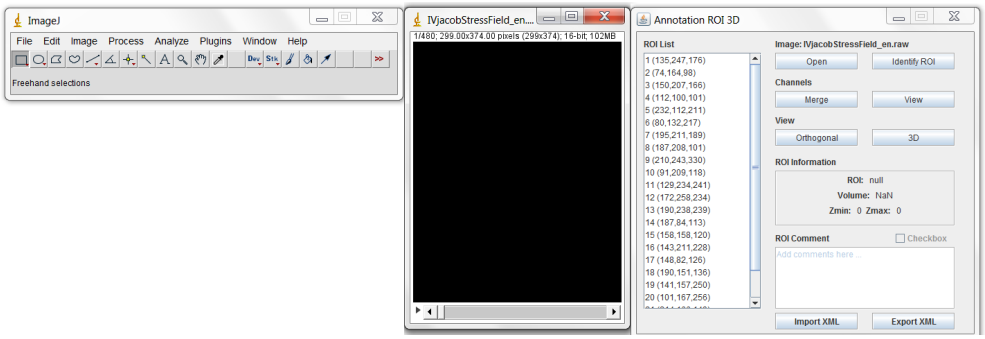

Figure B.5: Identify ROIs'Annotation ROI 3D' - how to identify ROIs within a selected image and the list output in the GUI with name of ROI and coordinates.

B.2.4 Merge

This is used to overlay images and create a composite based on different colour channels.

1. With two image stacks opened select merge. A dialog box will appear with various options.

2. For the colour channels desired from C1 to C7 select from a drop down list of open images, the channel you would like that image, see figure B.6.
3. The create composite checkbox creates a new image (figure B.7)merging the colour channels, the keep source images checkbox as suggested keeps the original images open. If checked, LUTs of source images are ignored. In this case, merged channels will adopt the lookup table mentioned besides the channel choice, this option is assumed when merging into RGB.

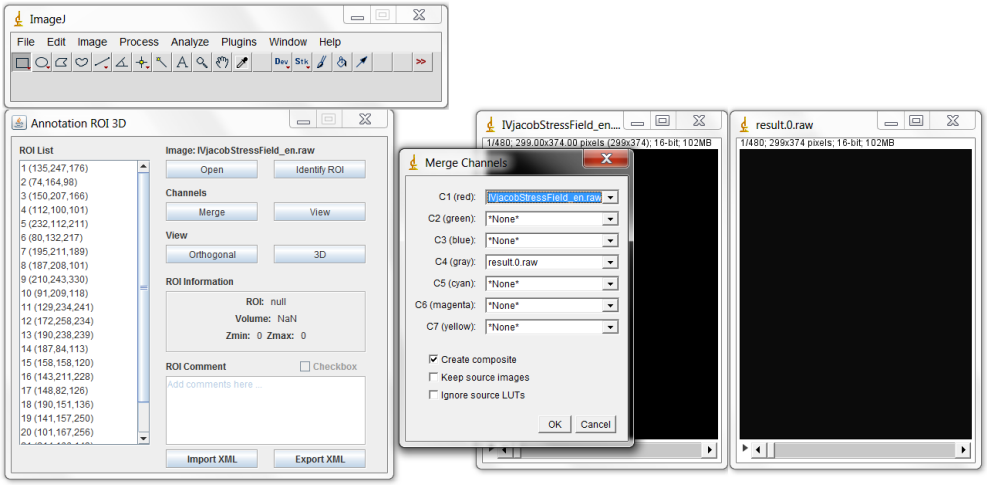

Figure B.6: Merge Channels 'Annotation ROI 3D' - merges image channels to create a combined composite image

B.2.5 View

This is used to view or hide merged channels within a composite image.

1. With an image already merged select the View button.
2. A dialog box will appear with the channels, select or deselect the checkbox associated with each channel you wish to view or hide as is shown in B.8.

B.2.6 Orthogonal & 3D Views

This is used to display the selected stack in Orthogonal view or 3D view.

1. With an image stack open select the Orthogonal button or 3D button.

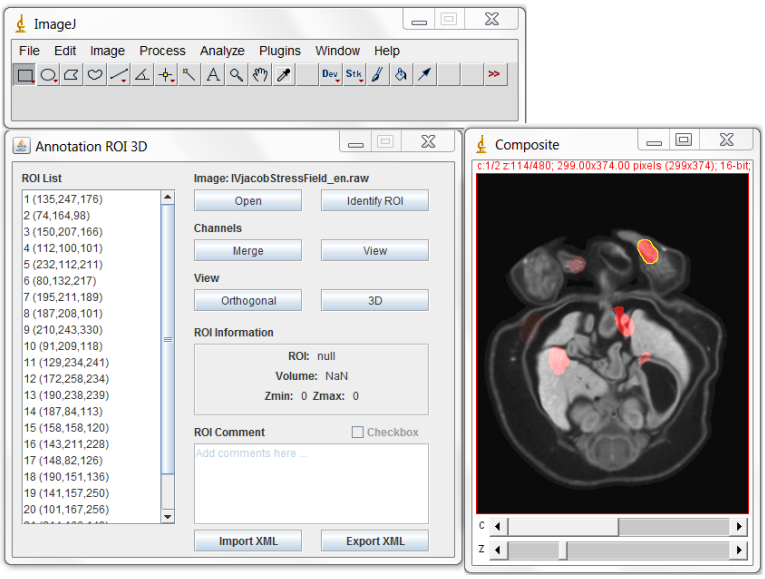

Figure B.7: Merge Channels - example image with merged channels as defined in previous screen capture.

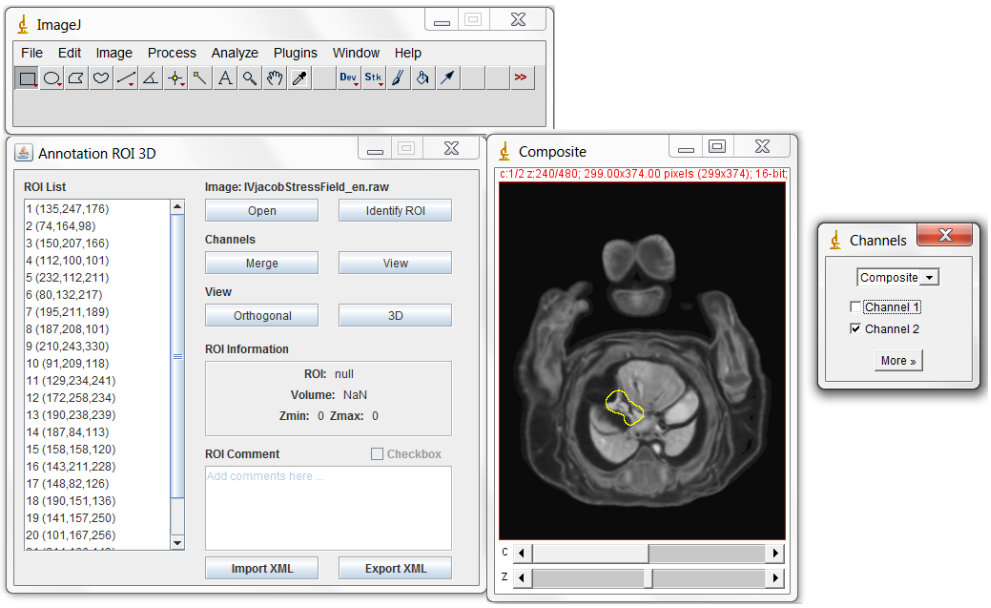

Figure B.8: View Channels Options 'Annotation ROI 3D' - the image is of the stack with the red overlay in channel 1 hidden.

2. If Orthogonal two new windows will open showing the image in orthogonal views (figure B.9), if 3D is selected the image will be projected in three dimensions as is shown in figure B.10.

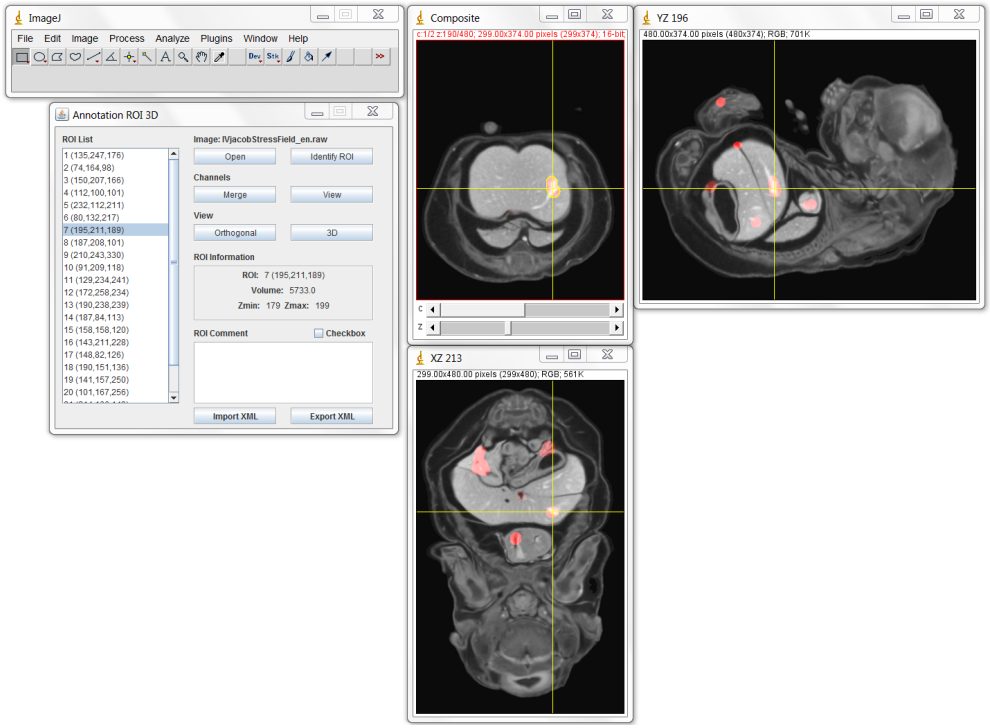

Figure B.9: Orthogonal 'Annotation ROI 3D' - displays the stack in orthogonal views.

B.2.7 ROI Information

The ROI information panel displays information about the currently selected stack. The fields are the name of the ROI and coordinates, the volume of the ROI and the initial slice $Zmin$ and final slice $Zmax$.

B.2.8 ROI Comment and Checkbox

ROI comment and checkbox provide a utility for feedback about the selected ROI, for each ROI the user can add feedback about that region. The use of the checkbox will be defined by the scientific survey being conducted but provides a quantitative feedback via a true false question. For each selected ROI the user can write a comment by

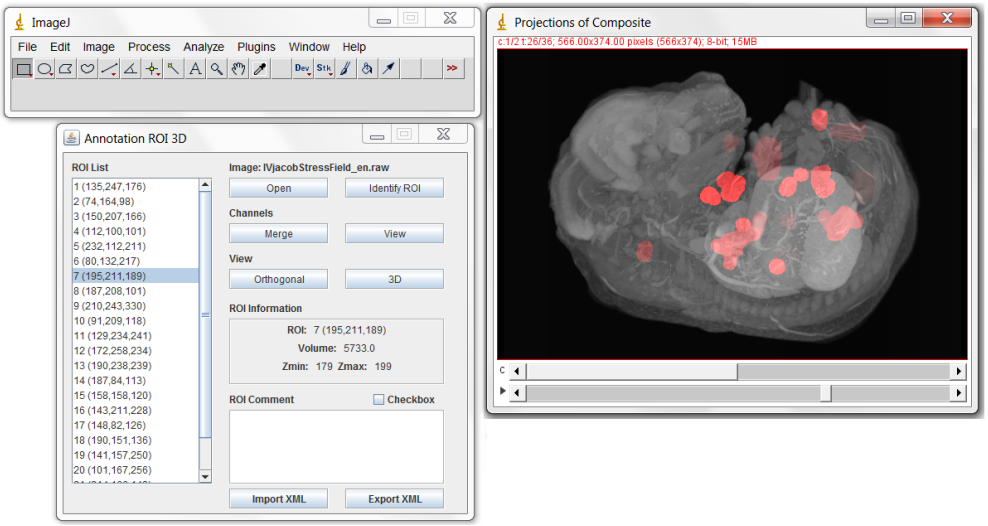

Figure B.10: 3D 'Annotation ROI 3D' - projects the stack in three dimensions.

navigating to the 'ROI Comment' box and inputting text and the checkbox can be selected or deselected by clicking on it.

B.2.9 XML Import & Export

XML import and export allows comments and checkbox values for a set of ROIs to be saved via XML export and loaded via XML import. For a set of annotated ROIs if 'Export XML' is selected a file chooser dialog will appear where the user can select the directory and name to create an XML file. This function can only work if a set of ROIs have already been identified.

The XML import feature imports comments and checkbox values for a set of identified ROIs. When selected XML import opens a dialog and allows the user to navigate to a file to be imported. If the data sets match then the data will be imported and displayed in the GUI.

Appendix C

GitHub & Code listing

C.1 Github

The web-based hosting service GitHub and Git revision control system was used throughout this project to host the source code for this open-source project. Four GitHub repositories that are publicly available host the code for this project:

- `https://github.com/wgrimes/fiji`
- `https://github.com/wgrimes/ImageJ_linux64`
- `https://github.com/wgrimes/ImageJ_win32`
- `https://github.com/wgrimes/MouseProject_TestData`

For development the 'fiji' git repository can be cloned to a local repository. The 'fiji' repository contains the source code used for development for the 'Annotation ROI 3D' plug-in along with all of the other plug-ins for the FIJI version of ImageJ. When changes are made to the code run the '/Build.sh' to build FIJI with all of the plug-ins located in the src-plugins folder.

The 'ImageJ_ linux64' git repository contains a version of ImageJ with 'Annotation ROI 3D' included and when cloned or downloaded as a zip ImageJ can be launched on a Linux 64-bit operating system. Similarly the 'ImageJ_ win32' git repository contains a version of ImageJ with 'Annotation ROI 3D' for Windows 32-bit operating systems.

Finally for testing purposes a 'MouseProject_ TestData' git repository is included, this contains four folders each containing a different sample μCT scan and overlay

MetaImage file. This is used for the purposes of testing the software functionality and to showcase how the plug-in will be used at the National Institute of Genetics.

C.2 Code Listing

This appendix also contains a listing of the 'Annotation ROI 3D' code, it is not a complete listing but includes the code that is most important to the function of the plug-in. The code was written in three packages org.annotationRoi3D.gui, org.annotationRoi3D.io and org.annotationRoi3D.methods. The Annotation_ ROI_ 3D.java file is the main file that launches the code, hence why it contains underscores.

C.2.1 Annotation_ ROI_ 3D.java

```
import org.annotationRoi3D.gui.InterfaceGUI;

import ij.IJ;
import ij.ImageJ;
import ij.plugin.PlugIn;

/**
 * Annotation ROI 3D
 *
 * The entry point of 3D annotation plugin, a plugin to annotate
 * 3D ROIs in ImageJ. The project is undertaken at the National
 * Institute of Informatics in Tokyo to assist the National
 * Institute of Genetics in providing feedback about ROIs relating
 * to mouse phenotypes. For each ROI comments can be added and a
 * classification.
 *
 * @author William Grimes
 * @version 1.0.0
 */
public class Annotation_ROI_3D implements PlugIn{

   /**
    * Method for running plugin from fiji build
    *
    * To compile fiji navigate to the fiji directory and run the
    * build script (./Build.sh) in linux. All plugins in
    * src-plugins folder will be included in the build.
    *
    * @param args unused
    */
   public void run(String arg) {
      new InterfaceGUI().setVisible(true);
   }

   /**
    * This method shows a message in a dialog box
    * titled message, about the program.
    */
   public void showAbout() {
      IJ.showMessage("Annotation_ROI_3D", "Annotation ROI 3D is a plugin to add comments
          ↪ and analysis to 3D regions of interest");
   }

   /**
    * Main method for debugging.
    *
    * For debugging, it is convenient to have a method that starts ImageJ, loads an
    * image and calls the plugin, e.g. after setting breakpoints.
    *
    * @param args unused
    */
   public static void main(String[] args) {
      new ImageJ(); // launches ImageJ
      new InterfaceGUI().setVisible(true);
   }
}
```

C.2.2 InterfaceGUI.java

```
package org.annotationRoi3D.gui;

import ij.*;
import ij.gui.*;
import ij.measure.ResultsTable;
import ij.plugin.PlugIn;

import java.awt.*;
import java.awt.event.*;
import java.util.HashMap;

import javax.swing.DefaultListModel;
import javax.swing.JFrame;
import javax.swing.GroupLayout.Alignment;
import javax.swing.GroupLayout;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JButton;
import javax.swing.JTextPane;
import javax.swing.ListSelectionModel;
import javax.swing.JLabel;
import javax.swing.JEditorPane;
import javax.swing.border.Border;
import javax.swing.border.EtchedBorder;

import mcib3d.geom.*;
```

```
import mcib3d.image3d.ImageHandler;

import org.annotationRoi3D.gui.EventAction;
import org.annotationRoi3D.methods.RoiDisplay;

import javax.swing.JCheckBox;

/**
 * This class initialises the GUI interface for the Annotation ROI 3D plug-in,
 * the layout is defined here. Java swing elements can be manipulated using
 * a window builder.
 *
 * @author William Grimes
 * @version 1.0.0
 */
public class InterfaceGUI extends JFrame implements PlugIn {

   private static final long serialVersionUID = -7508668331230924418L;
   public static ImageHandler ima;
   public static ImagePlus imp;
   public static ImagePlus currentImage;
   public static ImagePlus plus;
   public static ImagePlus activeImage;
   public static int middle;
   protected static Roi[] arrayRois;
   public static Object3D obj;
   protected static Objects3DPopulation objects3D;
   protected static DefaultListModel<Object> model = new DefaultListModel<Object>();
   protected static HashMap<Object, Object> hash;
   static boolean live = true;
   public static int[] indexes;
   public static int count;
   protected ResultsTable rtVoxels;
   public static String roiLabel = "null";
   public static double volume = Double.NaN;
   protected static int currentZmin = 0;
   protected static int currentZmax = 0;
   protected static int roiCmX = 0;
   protected static int roiCmY = 0;
   protected static int roiCmZ = 0;
   protected static String comment = "Add comments here ...";
   public JPanel jPanel = new JPanel();
   public static boolean identified = false;
   public Border border = new EtchedBorder(EtchedBorder.LOWERED);
   protected static JTextPane RoiInformation;
   protected static JEditorPane RoiComment;
   protected static JCheckBox chckbxCheckBox;
   protected static JLabel lblImage;
   protected static AdjustmentListener al;
   protected static MouseWheelListener ml;
   public static JList<Object> list = new JList<Object>();

   /**
    * Creates new Annotation_ROI_3D interface
    */
   public InterfaceGUI() {
      initComponents();
      this.setTitle("Annotation ROI 3D");
      model.clear();
      list.setModel(model);

      list.updateUI();

      setVisible(true);
      objects3D = new Objects3DPopulation();
      hash = new HashMap<Object, Object>();
      this.setDefaultCloseOperation(EXIT_ON_CLOSE);
   }

   /**
    * Initialises components of GUI and defines layout.
    */
   @SuppressWarnings("serial")
   private void initComponents() {
      jPanel.setLayout(null);
      setResizable(false);

      JLabel lblRoiList = new JLabel("ROI List");
      lblRoiList.setBounds(10, 10, 60, 15);
      jPanel.add(lblRoiList);

      list.setFont(new Font("Dialog", Font.PLAIN, 12));
      list.setModel(new javax.swing.AbstractListModel<Object>() {
         String[] strings = { "Item 1", "Item 2", "Item 3", "Item 4", "Item 5" };
         public int getSize() { return strings.length; }
         public Object getElementAt(int i) { return strings[i]; }
      });

      list.addListSelectionListener(new javax.swing.event.ListSelectionListener() {
         public void valueChanged(javax.swing.event.ListSelectionEvent evt) {
            EventAction.listValueChanged(evt);
         }
      });
      list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

      JScrollPane jScrollPane = new JScrollPane();
```

```
jScrollPane.setBounds(10, 30, 170, 370);
jPanel.add(jScrollPane);
jScrollPane.setViewportView(list);

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
layout.setHorizontalGroup(
   layout.createParallelGroup(Alignment.LEADING)
      .addGroup(layout.createSequentialGroup()
         .addComponent(jPanel, GroupLayout.PREFERRED_SIZE, 475, GroupLayout.
               ↪ PREFERRED_SIZE)
         .addContainerGap(GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
);
layout.setVerticalGroup(
   layout.createParallelGroup(Alignment.LEADING)
      .addComponent(jPanel, GroupLayout.DEFAULT_SIZE, 435, Short.MAX_VALUE)
);

lblImage = new JLabel("Image");
lblImage.setBounds(200, 10, 260, 15);
jPanel.add(lblImage);

JButton btnOpen = new JButton("Open");
btnOpen.setToolTipText("Open metaimage (.mhd) files");
btnOpen.setFont(new Font("Dialog", Font.PLAIN, 12));
btnOpen.setBounds(200, 30, 120, 25);
btnOpen.setBorder(border);
jPanel.add(btnOpen);
btnOpen.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      EventAction.buttonOpenActionPerformed(evt);
   }
});

JButton btnIdentifyROI = new JButton("Identify ROI");
btnIdentifyROI.setToolTipText("Identify ROIs within image and add to list");
btnIdentifyROI.setFont(new Font("Dialog", Font.PLAIN, 12));
btnIdentifyROI.setBounds(340, 30, 120, 25);
btnIdentifyROI.setBorder(border);
jPanel.add(btnIdentifyROI);
btnIdentifyROI.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      EventAction.buttonIdentifyROIActionPerformed(evt);
   }
});

JLabel lblChannels = new JLabel("Channels");
lblChannels.setBounds(200, 65, 70, 15);
jPanel.add(lblChannels);

JButton btnMergeChannels = new JButton("Merge");
btnMergeChannels.setToolTipText("<html>Merge colour channels and create a composite;
      ↪  <br> set C4 as scan image and C1 as overlay</html>");
btnMergeChannels.setFont(new Font("Dialog", Font.PLAIN, 12));
btnMergeChannels.setBounds(200, 85, 120, 25);
btnMergeChannels.setBorder(border);
jPanel.add(btnMergeChannels);
btnMergeChannels.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      EventAction.buttonMergeChannelsActionPerformed(evt);
   }
});

JLabel lblViews = new JLabel("View");
lblViews.setBounds(200, 120, 70, 15);
jPanel.add(lblViews);

JButton btnViewChannels = new JButton("View");
btnViewChannels.setToolTipText("View and hide channels");
btnViewChannels.setBounds(340, 85, 120, 25);
btnViewChannels.setFont(new Font("Dialog", Font.PLAIN, 12));
btnViewChannels.setBorder(border);
jPanel.add(btnViewChannels);
btnViewChannels.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      EventAction.buttonViewChannelsActionPerformed(evt);
   }
});

JButton btnOrthogonalViews = new JButton("Orthogonal");
btnOrthogonalViews.setToolTipText("Orthogonal view display of selected stack");
btnOrthogonalViews.setBounds(200, 140, 120, 25);
btnOrthogonalViews.setFont(new Font("Dialog", Font.PLAIN, 12));
btnOrthogonalViews.setBorder(border);
jPanel.add(btnOrthogonalViews);
btnOrthogonalViews.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      EventAction.buttonOrthogonalViewsActionPerformed(evt);
   }
});

JButton btnView3D = new JButton("3D");
btnView3D.setToolTipText("Project in 3D as rotating volume");
btnView3D.setBounds(340, 140, 120, 25);
btnView3D.setFont(new Font("Dialog", Font.PLAIN, 12));
btnView3D.setBorder(border);
jPanel.add(btnView3D);
```

```
btnView3D.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      EventAction.buttonView3DActionPerformed(evt);
   }
});

JLabel lblInformation = new JLabel("ROI Information");
lblInformation.setBounds(200, 180, 120, 15);
jPanel.add(lblInformation);

RoiInformation = new JTextPane();
RoiInformation.setToolTipText("<html>Displays information about selected ROI:<br>"
      + "ROI: name and coordinate positions (x, y and z)<br>"
      + "Volume: the ROI volume in voxels<br>"
      + "ZMin & ZMax: the minimum and maximum slice of ROI</html>");
RoiInformation.setContentType("text/html");
RoiInformation.setBorder(border);
RoiInformation.setBounds(200, 200, 260, 80);
RoiInformation.setEditable(false);
RoiInformation.setCursor(null);
RoiInformation.setOpaque(false);
RoiInformation.setFocusable(false);
RoiInformation.setText(
      "<html><table align=\"center\" style = \"font-family: Dialog; font-size: 12;
            ↪ color: #333333\">" +
           "<tr>"
           + "<td><b>ROI: </b></td><td>"+roiLabel+"</td>"
           + "</tr>"+
           "</table>"
           + "<table align=\"center\" style = \"font-family: Dialog; font-size: 12;
                 ↪ color: #333333\">" +
           "<tr>"
           + "<td><b>Volume: </b></td><td>"+volume+"</td>"
           + "</tr>"
           +"</table>"
           + "<table align=\"center\" style = \"font-family: Dialog; font-size: 12;
                 ↪ color: #333333\">" +
           "<tr>"
           + "<td><b>Zmin: </b></td><td>"+Integer.toString(currentZmin)+"</td><td><b
                 ↪ >Zmax: </b></td><td>"+Integer.toString(currentZmax)+"</td>"
           + "</tr>"+
           "</table></html>"
      );
jPanel.add(RoiInformation);

JLabel lblComments = new JLabel("ROI Comment");
lblComments.setBounds(200, 290, 125, 15);
jPanel.add(lblComments);

chckbxCheckBox = new JCheckBox("Checkbox");
chckbxCheckBox.setBounds(370, 290, 125, 15);
chckbxCheckBox.setEnabled(false);
jPanel.add(chckbxCheckBox);

RoiComment = new JEditorPane();
RoiComment.setEnabled(false);
RoiComment.setBounds(200, 310, 260, 90);
RoiComment.setBorder(border);
RoiComment.setText(comment);
jPanel.add(RoiComment);

JButton btnXmlImport = new JButton("Import XML");
btnXmlImport.setToolTipText("Import data from XML file");
btnXmlImport.setBounds(200, 405, 120, 25);
btnXmlImport.setBorder(border);
jPanel.add(btnXmlImport);
btnXmlImport.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      EventAction.buttonXmlImportActionPerformed(evt);
   }
});

JButton btnXmlExport = new JButton("Export XML");
btnXmlExport.setToolTipText("Export data to XML file");
btnXmlExport.setBounds(340, 405, 120, 25);
btnXmlExport.setBorder(border);
jPanel.add(btnXmlExport);
btnXmlExport.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      EventAction.buttonXmlExportActionPerformed(evt);
   }
});

getContentPane().setLayout(layout);

pack();
setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
}

/**
 * This method listens for adjustments in the JList
 * and implements updateRoi method.
 */
public void adjustmentValueChanged(AdjustmentEvent ae) {
   RoiDisplay.updateRois();
}
```

```
   @Override
   public void run(String arg) {
      new InterfaceGUI().setVisible(true);
   }
}
```

C.2.3 EventAction.java

```
package org.annotationRoi3D.gui;

import ij.IJ;
import ij.ImagePlus;
import ij.WindowManager;

import java.awt.Component;
import java.awt.Container;
import java.awt.Scrollbar;
import java.awt.event.AdjustmentListener;
import java.awt.event.MouseWheelListener;

import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;

import org.annotationRoi3D.io.ExportFileChooser;
import org.annotationRoi3D.io.ImportFileChooser;
import org.annotationRoi3D.methods.Image;
import org.annotationRoi3D.methods.RoiDisplay;

/**
 * This class provides a semantic output when a component defined action occurs,
 * this high-level event is generated by a component such as a Button when the
 * action occurs such as being clicked.
 *
 * @author William Grimes
 * @version 1.0.0
 */
public class EventAction extends InterfaceGUI{

   /**
    * This method runs the plug-in MetaImage_Reader to open .mhd files, All images are
        ↪ opened
    * in 16-bit since some features require 16-bit images or the same bit depth.
    */
   protected static void buttonOpenActionPerformed(java.awt.event.ActionEvent evt) {
      IJ.runPlugIn("org.annotationRoi3D.io.MetaImage_Reader", "");
      IJ.run("16-bit");
   }

   /**
    * This method performs a 3D segmentation of the current image and uses the addImage()
    * method to display ROI in a JList.
    */
   protected static void buttonIdentifyROIActionPerformed(java.awt.event.ActionEvent evt)
        ↪ {
      if (WindowManager.getCurrentWindow() == null){
         IJ.showMessage("No Image", "No images are open");
      }
      else{
         if (org.annotationRoi3D.gui.InterfaceGUI.identified == true){
            IJ.showMessage("ROIs Identied", "ROIs already identified");
         }
         else{
            lblImage.setText("Image: " + WindowManager.getCurrentWindow().toString());
            ImagePlus beforeSegmentation = WindowManager.getCurrentImage();
            org.annotationRoi3D.methods.Segmentation.segmentation3D();
            beforeSegmentation.hide();
            Image.addImage();
            identified = true;
         }
      }
   }

   /**
    * Upon the event of a list value change the current Roi is shown in the image, in
        ↪ addition
    * the values displayed in the GUI are updated to the relevant GUI.
    */
   protected static void listValueChanged(javax.swing.event.ListSelectionEvent evt) {
      if (obj != null){
         obj.setComment(RoiComment.getText());
         if (chckbxCheckBox.isSelected()==true){
            obj.setName("true");
         }
         else{
            obj.setName("false");
         }
      }
      RoiDisplay.showRoi(true, false);
      RoiInformation.setText(
```

```
        "<html><table align=\"center\" style = \"font-family: Dialog; font-size: 12;
              ↪ color: #333333\">" +
             "<tr>"
             + "<td><b>ROI: </b></td><td>"+roiLabel+"</td>"
             + "</tr>"+
             "</table>"
             + "<table align=\"center\" style = \"font-family: Dialog; font-size: 12;
                   ↪ color: #333333\">" +
             "<tr>"
             + "<td><b>Volume: </b></td><td>"+volume+"</td>"
             + "</tr>"
             +"</table>"
             + "<table align=\"center\" style = \"font-family: Dialog; font-size: 12;
                   ↪ color: #333333\">" +
             "<tr>"
             + "<td><b>Zmin: </b></td><td>"+Integer.toString(currentZmin)+"</td><td><b
                   ↪ >Zmax: </b></td><td>"+Integer.toString(currentZmax)+"</td>"
             + "</tr>"+
             "</table></html>"
       );
   RoiComment.setEnabled(true);
   chckbxCheckBox.setEnabled(true);
   RoiComment.setText(comment);
   if (obj.getName().equals("true")){
      chckbxCheckBox.setSelected(true);
   }
   else{
      chckbxCheckBox.setSelected(false);
   }
}

/**
 * This method runs merge channels function to merge channels of different colours.
 * It is useful to create a composite image combining images based on colour. This
 * feature requires images of the same bit-depth.
 */
protected static void buttonMergeChannelsActionPerformed(java.awt.event.ActionEvent evt
      ↪ ) {
   IJ.run("Merge Channels...");
   imp = Image.getImage();
   imp = WindowManager.getCurrentImage();
   RoiDisplay.showRoi(true, false);
}

/**
 * This method allows channels to be displayed by colour. It is useful to remove an
 * overlayed channel.
 */
protected static void buttonViewChannelsActionPerformed(java.awt.event.ActionEvent evt)
      ↪  {
   if (WindowManager.getCurrentWindow() == null){
      IJ.showMessage("No Image", "No images are open");
   }
   else{
      IJ.run("Channels Tool...", "");
   }
}

/**
 * This method displays the current stack in orthogonal view showing sagittal sections,
 * coronal (YZ projection image) window and transverse (XZ projection image) window.
 */
protected static void buttonOrthogonalViewsActionPerformed(java.awt.event.ActionEvent
      ↪ evt) {
   IJ.run("Orthogonal Views", "");
}

/**
 * This method creates a sequence of projections of a rotating volume on one plane.
 */
protected static void buttonView3DActionPerformed(java.awt.event.ActionEvent evt) {
   IJ.run("Select None", "");
   IJ.run(WindowManager.getCurrentImage(), "3D Project...", "projection=[Brightest
         ↪ Point] axis=Y-Axis slice=1 initial=0 total=360 rotation=10 lower=1 upper
         ↪ =255 opacity=0 surface=100 interior=50");
}

/**
 * This method adjusts the image based on the selection in the scroll bar.
 */
public static void addScrollListener(ImagePlus img, AdjustmentListener al,
      ↪ MouseWheelListener ml) {
   for (Component c : img.getWindow().getComponents()) {
      if (c instanceof Scrollbar) {
         ((Scrollbar) c).addAdjustmentListener(al);
      } else if (c instanceof Container) {
         for (Component c2 : ((Container) c).getComponents()) {
            if (c2 instanceof Scrollbar) {
               ((Scrollbar) c2).addAdjustmentListener(al);
            }
         }
      }
   }
   img.getWindow().addMouseWheelListener(ml);
}
```

```
   /**
    * This method removes the scroll listener so the image is not adjusted by selection in
          ↪  scroll bar.
    */
   public static void removeScrollListener(ImagePlus img, AdjustmentListener al,
         ↪ MouseWheelListener ml) {
      for (Component c : img.getWindow().getComponents()) {
         if (c instanceof Scrollbar) {
            ((Scrollbar) c).removeAdjustmentListener(al);
         } else if (c instanceof Container) {
            for (Component c2 : ((Container) c).getComponents()) {
               if (c2 instanceof Scrollbar) {
                  ((Scrollbar) c2).removeAdjustmentListener(al);
               }
            }
         }
      }
      img.getWindow().removeMouseWheelListener(ml);
   }

   protected static void buttonXmlImportActionPerformed(java.awt.event.ActionEvent evt) {
      if (org.annotationRoi3D.gui.InterfaceGUI.identified == false){
         IJ.showMessage("Identify ROIs", "Identify ROIs before importing data");
      }
      else{
         ImportFileChooser importWindow = new ImportFileChooser();
      }
   }

   /**
    * This method launches the method to save files in XML format.
    */
   protected static void buttonXmlExportActionPerformed(java.awt.event.ActionEvent evt) {
      if (obj != null){
         obj.setComment(RoiComment.getText());
      }
      ExportFileChooser exportWindow = new ExportFileChooser();
   }
}
```

C.2.4 ImportXML.java

```
package org.annotationRoi3D.io;

import ij.IJ;

import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;

import org.annotationRoi3D.gui.InterfaceGUI;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import org.w3c.dom.Element;

import java.io.File;
import java.util.Arrays;

public class ImportXML extends InterfaceGUI{
   /**
    * This method imports comments from an XML
    * file, comments are added to Objects3D
    * population. A check is performed to ensure
    * that the data sets match, this is performed
    * by checking the name and coordinates of each
    * ROI is the same.
    *
    * @param importFile the file to be imported.
    * @return Nothing.
    */
   public static void ImportXML(File importFile){

      try {
         DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
         DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
         Document doc = dBuilder.parse(importFile);

         doc.getDocumentElement().normalize();

         NodeList nList = doc.getElementsByTagName("ROI");

         String[] array0 = new String[model.getSize()];
         for (int i = 0; i < model.getSize(); i++){
            array0[i] = model.get(i).toString(); // creates array for identified ROIs
         }

         String[] array1 = new String[nList.getLength()];
         for (int i = 0; i < nList.getLength(); i++){
            Node nNode = nList.item(i);

            if (nNode.getNodeType() == Node.ELEMENT_NODE) {
               Element eElement = (Element) nNode;
```

```
            array1[i] = eElement.getElementsByTagName("name").item(0).getTextContent().
                ↪ substring(11);
            //creates array for ROIs in XML file
          }
        }

        if(Arrays.equals(array0, array1) == false){
          IJ.showMessage("Import XML", "Data sets to not match");
        }
        if (Arrays.equals(array0, array1) == true){
          for (int temp = 0; temp < nList.getLength(); temp++) {
            Node nNode = nList.item(temp);
            if (nNode.getNodeType() == Node.ELEMENT_NODE) {
              Element eElement = (Element) nNode;
              objects3D.getObject(temp).setComment(eElement.getElementsByTagName("
                  ↪ comment").item(0).getTextContent());
              objects3D.getObject(temp).setName(eElement.getElementsByTagName("checkbox
                  ↪ ").item(0).getTextContent());
              //sets the comment value and checkbox for identified ROIs
            }
          }
        }
      } catch (Exception e) {
        e.printStackTrace();
      }
    }

}
```

C.2.5 ImportFileChooser.java

```
package org.annotationRoi3D.io;

import java.awt.BorderLayout;
import java.io.File;

import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.filechooser.FileNameExtensionFilter;

public class ImportFileChooser extends JPanel {

   private static final long serialVersionUID = 1L;

   public static File ImportFile;
   JFileChooser fcImport;

   /**
    * This creates a dialog window for exporting
    * and importing XML files.
    */
   public ImportFileChooser() {
     super(new BorderLayout());
     fcImport = new JFileChooser();
     fcImport.addChoosableFileFilter(new FileNameExtensionFilter("XML Files", "xml"));
     fcImport.setFileSelectionMode(JFileChooser.FILES_ONLY);
     fcImport.setAcceptAllFileFilterUsed(true);

     int returnValImport = fcImport.showOpenDialog(ImportFileChooser.this);
     if (returnValImport == JFileChooser.APPROVE_OPTION) {
       ImportFile = fcImport.getSelectedFile();
       ImportXML.ImportXML(ImportFile);
     }
   }

   /**
    * Create the GUI and show it. For thread safety,
    * this method should be invoked from the
    * event dispatch thread.
    */
   public static void createAndShowGUI() {
     //Create and set up the window.
     JFrame frameImport = new JFrame("FileChooserImport");
     frameImport.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

     //Add content to the window.
     frameImport.add(new ImportFileChooser());

     //Display the window.
     frameImport.pack();
     frameImport.setVisible(true);
     }

}
```

C.2.6 ExportXML.java

```
package org.annotationRoi3D.io;
```

```java
import java.io.File;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import org.annotationRoi3D.gui.InterfaceGUI;
import org.w3c.dom.Document;
import org.w3c.dom.Element;

/**
 * This class parses ROI comments into an XML file
 * and streams the output, it is implemented by
 * the FileChooser class.
 *
 * @return Nothing.
 */
public class ExportXML extends InterfaceGUI{
   public static DOMSource source;
   public static Transformer transformer;

   public static void OutputXML() {

     try {
      DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();
      DocumentBuilder docBuilder = docFactory.newDocumentBuilder();

      // root elements
      Document doc = docBuilder.newDocument();
      Element rootElement = doc.createElement("ROIs");
      doc.appendChild(rootElement);

      for (int i = 0; i < objects3D.getNbObjects(); i++){
        //System.out.println(objects3D.getObject(i).toString());

        Element ROI = doc.createElement("ROI");
        rootElement.appendChild(ROI);

        //X-coordinate elements
        Element name = doc.createElement("name");
        name.appendChild(doc.createTextNode(objects3D.getObject(i).toString()));
        ROI.appendChild(name);

        Element x = doc.createElement("x");
        x.appendChild(doc.createTextNode(String.valueOf(objects3D.getObject(i).getCenterX
            ↪ ())));
        ROI.appendChild(x);

        // Y-coordinate elements
        Element y = doc.createElement("y");
        y.appendChild(doc.createTextNode(String.valueOf(objects3D.getObject(i).getCenterY
            ↪ ())));
        ROI.appendChild(y);

        // Z-coordinate elements
        Element lastname = doc.createElement("z");
        lastname.appendChild(doc.createTextNode(String.valueOf(objects3D.getObject(i).
            ↪ getCenterZ())));
        ROI.appendChild(lastname);

        // checkbox elements
        Element checkbox = doc.createElement("checkbox");
        checkbox.appendChild(doc.createTextNode(String.valueOf(objects3D.getObject(i).
            ↪ getName().toString())));
        ROI.appendChild(checkbox);

        // comment elements
        Element comment = doc.createElement("comment");
        comment.appendChild(doc.createTextNode(String.valueOf(objects3D.getObject(i).
            ↪ getComment().toString())));
        ROI.appendChild(comment);
      }

      // write the content into xml file
      TransformerFactory transformerFactory = TransformerFactory.newInstance();
      Transformer transformer = transformerFactory.newTransformer();
      DOMSource source = new DOMSource(doc);

      StreamResult result = new StreamResult(new File(ExportFileChooser.ExportFile.
          ↪ getAbsolutePath()));

      // Output to console for testing
      //StreamResult result = new StreamResult(System.out);
      //System.out.println("File saved!");

      transformer.transform(source, result);

     } catch (ParserConfigurationException pce) {
      pce.printStackTrace();
```

```
    } catch (TransformerException tfe) {
     tfe.printStackTrace();
    }
   }
}
```

C.2.7 ExportFileChooser.java

```
package org.annotationRoi3D.io;

import java.io.*;
import java.awt.*;

import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JPanel;

/**
 * This creates a dialog window for exporting
 * and importing XML files.
 */
public class ExportFileChooser extends JPanel {

   private static final long serialVersionUID = 1L;
   public static File ExportFile;
    JFileChooser fcExport;

    public ExportFileChooser() {
        super(new BorderLayout());
        fcExport = new JFileChooser();

        int returnValExport = fcExport.showSaveDialog(ExportFileChooser.this);
        if (returnValExport == JFileChooser.APPROVE_OPTION) {
            ExportFile = fcExport.getSelectedFile();
            org.annotationRoi3D.io.ExportXML.OutputXML();
        }
    }

    /**
     * Create the GUI and show it. For thread safety,
     * this method should be invoked from the
     * event dispatch thread.
     */
    public static void createAndShowGUI() {
        //Create and set up the window.
        JFrame frameExport = new JFrame("FileChooserExport");
        frameExport.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        //Add content to the window.
        frameExport.add(new ExportFileChooser());

        //Display the window.
        frameExport.pack();
        frameExport.setVisible(true);
    }
}
```

C.2.8 MetaImage_ Reader.java

```
package org.annotationRoi3D.io;

/**
MetaImage reader plugin for ImageJ.

This plugin reads MetaImage text-based tagged format files.

Author: Kang Li (kangli AT cs.cmu.edu)

Installation:
  Download MetaImage_Reader_Writer.jar to the plugins folder, or subfolder.
  Restart ImageJ, and there will be new File/Import/MetaImage... and
  File/Save As/MetaImage... commands.

History:
  2007/04/07: First version
  2008/07/25: Fixed two bugs (thanks to Jon Rohrer)

References:
  MetaIO Documentation (http://www.itk.org/Wiki/MetaIO/Documentation)
*/

/**
Copyright (C) 2007-2008 Kang Li. All rights reserved.

Permission to use, copy, modify, and distribute this software for any purpose
without fee is hereby granted, provided that this entire notice is included
in all copies of any software which is or includes a copy or modification of
this software and in all copies of the supporting documentation for such
```

```
software. Any for profit use of this software is expressly forbidden without
first obtaining the explicit consent of the author.

THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED
WARRANTY. IN PARTICULAR, THE AUTHOR DOES NOT MAKE ANY REPRESENTATION OR
WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY OF THIS SOFTWARE OR ITS
FITNESS FOR ANY PARTICULAR PURPOSE.
*/

import java.io.*;
import java.util.*;
import ij.*;
import ij.plugin.*;
import ij.process.*;
import ij.io.*;

public class MetaImage_Reader implements PlugIn {

public boolean littleEndian = false;

public void run(String arg) {
    OpenDialog od = new OpenDialog("Open MetaImage...", arg);
    String dir = od.getDirectory();
    String baseName = od.getFileName();
    if (baseName == null || baseName.length() == 0)
        return;
    int baseLength = baseName.length();
    String lowerBaseName = baseName.toLowerCase();
    boolean mhd = lowerBaseName.endsWith(".mhd");
    boolean mha = lowerBaseName.endsWith(".mha");
    boolean raw = lowerBaseName.endsWith(".raw");
    String headerName;
    if (mha || mhd) {
        headerName = baseName;
        baseName  = baseName.substring(0, baseLength - 4);
    }
    else {
        baseName  = baseName.substring(0, baseLength - 4);
        headerName = baseName + ".mhd";
    }
    IJ.showStatus("Opening " + headerName + "...");
    ImagePlus imp = load(dir, baseName, headerName, mha);
    if (imp != null)
        imp.show();
    IJ.showStatus(baseName + " opened");
}

private ImagePlus load(String dir,
                       String baseName,
                       String headerName,
                       boolean local)
{
    ImagePlus impOut = null;
    try {
        FileInfo fi = readHeader(dir, baseName, headerName, local);
        if (fi.fileName.equals("LIST")) {
            // Loads a sequence of files.
            BufferedReader in = new BufferedReader(new FileReader(dir + headerName));
            ImageStack stackOut = new ImageStack(fi.width, fi.height);
            boolean autoOffset = (fi.longOffset < 0);
            boolean fileNamesBegin = false;
            String line = in.readLine();
            int index = 0, numImages = fi.nImages;
            while (null != line) {
                line = line.trim();
                if (fileNamesBegin) {
                    String[] parts = line.split("\\s+");
                    for (int i = 0; i < parts.length; ++i) {
                        // Adjust the fields of FileInfo.
                        fi.nImages = 1;
                        fi.fileName = parts[i];
                        if (autoOffset)
                            fi.longOffset = getOffset(fi);
                        IJ.showStatus("Reading " + fi.fileName + "...");
                        FileOpener opener = new FileOpener(fi);
                        ImagePlus imp = opener.open(false);
                        ImageStack stack = imp.getStack();
                        for (int j = 1; j <= stack.getSize(); ++j) {
                            // Load the first image only even if there are more.
                            ImageProcessor ip = stack.getProcessor(j);
                            stackOut.addSlice(fi.fileName, ip);
                            break;
                        } // for j
                        if (++index >= numImages)
                            break;
                    } // for i
                }
                else {
                    if (line.startsWith("ElementDataFile"))
                        fileNamesBegin = true;
                }
                if (index >= numImages)
                    break;
                line = in.readLine();
            }
```

```
    impOut = new ImagePlus(baseName, stackOut);
    impOut.setStack(null, stackOut);
}
else if (fi.fileName.indexOf('%') >= 0) {
  String[] parts = fi.fileName.split("\\s+");
  int imin = 1;
  int imax = fi.nImages;
  int step = 1;
  if (parts.length > 1) {
    imin = Integer.parseInt(parts[1]);
    if (parts.length > 2) {
      imax = Integer.parseInt(parts[2]);
      if (parts.length > 3)
        step = Integer.parseInt(parts[3]);
    }
  }
  boolean autoOffset = (fi.longOffset < 0);
  ImageStack stackOut = new ImageStack(fi.width, fi.height);
  int index = 0, numImages = fi.nImages;
  fi.nImages = 1;
  for (int i = imin; i <= imax; i += step) {
    Formatter formatter = new Formatter();
    formatter.format(parts[0], i);
    fi.fileName = formatter.toString();
    if (autoOffset)
      fi.longOffset = getOffset(fi);
    IJ.showStatus("Reading " + fi.fileName + "...");
        FileOpener opener = new FileOpener(fi);
        ImagePlus imp = opener.open(false);
        ImageStack stack = imp.getStack();
        for (int j = 1; j <= stack.getSize(); ++j) {
            // Load the first image only even if there are more.
            ImageProcessor ip = stack.getProcessor(j);
            stackOut.addSlice(fi.fileName, ip);
            break;
        } // for j
        if (++index >= numImages)
break;
  } // for i
  impOut = new ImagePlus(baseName, stackOut);
    impOut.setStack(null, stackOut);
}
else {
    if (fi.longOffset < 0)
        fi.longOffset = getOffset(fi);
    IJ.showStatus("Reading " + fi.fileName + "...");
    FileOpener opener = new FileOpener(fi);
    impOut = opener.open(false);
        }
    }
    catch (IOException e) {
        IJ.error("MetaImage Reader: " + e.getMessage());
    }
    catch (NumberFormatException e) {
        IJ.error("MetaImage Reader: " + e.getMessage());
    }
    return impOut;
}


private FileInfo readHeader(String dir,
                            String baseName,
                            String headerName,
                            boolean local) throws IOException, NumberFormatException
{
    FileInfo fi = new FileInfo();
    fi.directory = dir;
    fi.fileFormat = FileInfo.RAW;

    Properties p = new Properties();
    p.load(new FileInputStream(dir + headerName));
    String strObjectType = p.getProperty("ObjectType");
    String strNDims = p.getProperty("NDims");
    String strDimSize = p.getProperty("DimSize");
    String strElementSize = p.getProperty("ElementSize");
    String strElementDataFile = p.getProperty("ElementDataFile");
    String strElementByteOrderMSB = p.getProperty("ElementByteOrderMSB");
    if (null == strElementByteOrderMSB)
      strElementByteOrderMSB = p.getProperty("BinaryDataByteOrderMSB");
    String strElementNumberOfChannels = p.getProperty("ElementNumberOfChannels", "1");
    String strElementType = p.getProperty("ElementType", "MET_NONE");
    String strHeaderSize = p.getProperty("HeaderSize", "0");

    if (strObjectType == null || !strObjectType.equalsIgnoreCase("Image"))
        throw new IOException("The specified file does not contain an image.");
    int ndims = Integer.parseInt(strNDims);
    if (strDimSize == null)
        throw new IOException("The image dimension size is unspecified.");
    else {
        String[] parts = strDimSize.split("\\s+");
        if (parts.length < ndims)
            throw new IOException("Invalid dimension size.");
        if (ndims > 1) {
            fi.width = Integer.parseInt(parts[0]);
            fi.height = Integer.parseInt(parts[1]);
            fi.nImages = 1;
```

```
        if (ndims > 2) {
            for (int i = ndims - 1; i >= 2; --i)
                fi.nImages *= Integer.parseInt(parts[i]);
        }
    }
    else {
        throw new IOException("Unsupported number of dimensions.");
    }
}
if (strElementSize != null) {
    String[] parts = strElementSize.split("\\s+");
    if (parts.length > 0)
        fi.pixelWidth = Double.parseDouble(parts[0]);
    if (parts.length > 1)
        fi.pixelHeight = Double.parseDouble(parts[1]);
    if (parts.length > 2)
        fi.pixelDepth = Double.parseDouble(parts[2]);
}
int numChannels = Integer.parseInt(strElementNumberOfChannels);
if (numChannels == 1) {
    if (strElementType.equals("MET_UCHAR"))   { fi.fileType = FileInfo.GRAY8;       }
    else if (strElementType.equals("MET_SHORT")) { fi.fileType = FileInfo.
        ↪ GRAY16_SIGNED; }
    else if (strElementType.equals("MET_USHORT")) { fi.fileType = FileInfo.
        ↪ GRAY16_UNSIGNED; }
    else if (strElementType.equals("MET_INT")) { fi.fileType = FileInfo.GRAY32_INT; }
    else if (strElementType.equals("MET_UINT")) { fi.fileType = FileInfo.
        ↪ GRAY32_UNSIGNED; }
    else if (strElementType.equals("MET_FLOAT")) { fi.fileType = FileInfo.GRAY32_FLOAT
        ↪ ; }
    else {
        throw new IOException(
            "Unsupported element type: " +
            strElementType +
            ".");
    }
}
else if (numChannels == 3) {
    if (strElementType.equals("MET_UCHAR_ARRAY")) fi.fileType = FileInfo.RGB;
    else if (strElementType.equals("MET_USHORT_ARRAY"))
        fi.fileType = FileInfo.RGB48;
    else {
        throw new IOException(
            "Unsupported element type: " +
            strElementType +
            ".");
    }
}
    else {
        throw new IOException("Unsupported number of channels.");
    }

    if (strElementDataFile != null && strElementDataFile.length() > 0) {
      if (strElementDataFile.equals("LOCAL"))
        fi.fileName = headerName;
      else
        fi.fileName = strElementDataFile;
    }
    else {
        if (!local)
          fi.fileName = baseName + ".raw";
        else
          fi.fileName = headerName;
    }

    if (strElementByteOrderMSB != null) {
        if (strElementByteOrderMSB.length() > 0
            && (strElementByteOrderMSB.charAt(0) == 'T' ||
                strElementByteOrderMSB.charAt(0) == 't' ||
                strElementByteOrderMSB.charAt(0) == '1'))
            fi.intelByteOrder = false;
        else
            fi.intelByteOrder = true;
    }

    fi.longOffset = (long)Integer.parseInt(strHeaderSize);

    return fi;
}


private int getBytesPerPixel(FileInfo fi) {
    int bpp = 0;
    switch (fi.fileType) {
    case FileInfo.GRAY8:        return 1;
    case FileInfo.GRAY16_SIGNED: return 2;
    case FileInfo.GRAY16_UNSIGNED: return 2;
    case FileInfo.GRAY32_INT:   return 4;
    case FileInfo.GRAY32_UNSIGNED: return 4;
    case FileInfo.GRAY32_FLOAT: return 4;
    case FileInfo.RGB:          return 24;
    case FileInfo.RGB48:        return 48;
    default:
        break;
    }
    return bpp;
```

```
}


private long getOffset(FileInfo fi) {
    // Automatically calculate the header size.
    int bpp = getBytesPerPixel(fi);
    long dataBytes = bpp * fi.width * fi.height * fi.nImages;
    File file = new File(fi.directory + fi.fileName);
    return file.length() - dataBytes;
}
}
```

C.2.9 Image.java

```
package org.annotationRoi3D.methods;

import java.util.ArrayList;

import ij.IJ;
import ij.ImagePlus;
import ij.WindowManager;
import ij.measure.Calibration;

import mcib3d.geom.Object3D;
import mcib3d.geom.Object3DVoxels;
import mcib3d.geom.Objects3DPopulation;
import mcib3d.geom.Voxel3D;
import mcib3d.image3d.ImageHandler;
import mcib3d.image3d.ImageInt;

import org.annotationRoi3D.gui.EventAction;
import org.annotationRoi3D.gui.InterfaceGUI;


/**
 * This class contains methods relating to Image processes
 * used within Annotation_ROI_3D
 */
public class Image extends InterfaceGUI{
   /**
    * Returns a reference to the active image
    * or null if there isn't one.
    *
    * @return imp image reference
    */
public static ImagePlus getImage() {
   imp = WindowManager.getCurrentImage();
   if (imp == null) {
      return null;
   } else {
      return imp;
   }
}

/**
 * Returns a reference to 3D image
 *
 * @return ImageHanlder
 */
public static ImageHandler getImage3D() {
   ImagePlus imp = getImage();
   return ImageHandler.wrap(imp);
}

/**
 * Sets current image as active image
 */
public static void registerActiveImage() {
   activeImage = Image.getImage();

   if (activeImage != null && activeImage.getProcessor() != null && activeImage.
         ↪ getImageStackSize() > 1) {
      if (currentImage != null && currentImage.getWindow() != null && currentImage !=
            ↪ activeImage) {
         EventAction.removeScrollListener(currentImage, al, ml);
         EventAction.removeScrollListener(currentImage, al, ml);
         currentImage.killRoi();
         currentImage.updateAndDraw();
         currentImage = null;
      }
      if (currentImage != activeImage) {
         //System.out.println("add listener:"+activeImage.getTitle());
         EventAction.addScrollListener(activeImage, al, ml);
         currentImage = activeImage;
      }
   }
}

/**
 * Adds a feature to the Image attribute of the RoiManager3D_ object
 */
public static void addImage() {
```

```
   plus = Image.getImage();
   ImagePlus contours;
   String title = plus.getTitle();
   objects3D.setCalibration(plus.getCalibration());

   ImageHandler seg = ImageHandler.wrap(plus);
   int min = (int) seg.getMinAboveValue(0);
   int max = (int) seg.getMax();
   // iterate in image and constructs objects
   ArrayList<Voxel3D>[] objects = new ArrayList[max - min + 1];
   for (int i = 0; i < max - min + 1; i++) {
      objects[i] = new ArrayList<Voxel3D>();
   }
   float pix;
   for (int k = 0; k < seg.sizeZ; k++) {
      for (int j = 0; j < seg.sizeY; j++) {
         for (int i = 0; i < seg.sizeX; i++) {
            pix = seg.getPixel(i, j, k);
            if (pix > 0) {
               objects[(int) (pix) - min].add(new Voxel3D(i, j, k, pix));
            }
         }
      }
   }
   // ARRAYLIST
   ArrayList listObjects = new ArrayList();
   for (int i = 0; i < max - min + 1; i++) {
      if (!objects[i].isEmpty()) {
         listObjects.add(objects[i]);
      }
   }
   // add objects
   addListVoxels(listObjects, plus);
   list.updateUI();
}

/**
 *   List voxels in the image with values > threshold
 *
 * @param list ArrayList of obj
 * @param plus image added by addImage()
 */
public static void addListVoxels(ArrayList list, ImagePlus plus) {
   ArrayList<Voxel3D> objList;
   java.util.Iterator it = list.iterator();
   Object3D obj;
   Calibration cal = plus.getCalibration();
   int i = 1;
   ImageInt seg = ImageInt.wrap(plus);
   while (it.hasNext()) {
      // Object3D
      objList = (ArrayList<Voxel3D>) it.next();
      if (!objList.isEmpty()) {
         obj = new Object3DVoxels(objList);
         obj.setCalibration(cal);
         obj.setLabelImage(seg);
         obj.setName("false");
         obj.setComment(" ");
         obj.computeContours();
         if (obj.getAreaPixels() == 0) {
            IJ.log("area 0 " + obj);
         }
         // RoiManager
         addObject3D(obj);
         i++;
      }
   }
}


/**
 * Adds object3D to model and updates list
 *
 * @return nothing
 * @param obj 3d object to be added
 */
public static void addObject3D(Object3D obj) {
   objects3D.addObject(obj);
   model.addElement(obj.toString().substring(11));
   list.updateUI();
   list.repaint();
   list.revalidate();
}

/**
 * Gets the allIndexes attribute of the RoiManager object
 *
 * @return The allIndexes value
 */
public static int[] getAllIndexes() {
   int count = model.getSize();
   int[] indexes = new int[count];
   for (int i = 0; i < count; i++) {
      indexes[i] = i;
   }
   return indexes;
```

```
   }

}
```

C.2.10 RoiDisplay.java

```
package org.annotationRoi3D.methods;

import java.awt.Frame;

import ij.gui.*;
import ij.IJ;
import ij.ImagePlus;
import ij.Macro;
import ij.gui.MessageDialog;
import ij.plugin.filter.ThresholdToSelection;
import ij.process.ByteProcessor;
import ij.process.ImageProcessor;
import mcib3d.image3d.ImageHandler;

import org.annotationRoi3D.gui.InterfaceGUI;

/**
 * This class contains all methods relating
 * to ROI display and updating.
 */
public class RoiDisplay extends InterfaceGUI{

   /**
    * Shows selected Roi on image and defines
    * parameters of selected Roi to be displayed
    * in Roi information.
    */
   public static void showRoi(boolean force, boolean all) {
      //ImageHandler ima = Image.getImage3D();
      //ImagePlus imp = Image.getImage();
      //Image.registerActiveImage();
      if (imp == null) {
         error("There are no images open.");
         return;
      }
      int zmin = imp.getNSlices() + 1;
      int zmax = -1;
      if (force) {
      // draw mask of rois
      indexes = list.getSelectedIndices();
      if ((indexes.length == 0) || all) {
         indexes = Image.getAllIndexes();
      }
      arrayRois = new Roi[imp.getNSlices()];
      // get zmin and zmax

      //Object3D obj = null;
      for (int i = 0; i < indexes.length; i++) {
         obj = objects3D.getObject(indexes[i]);
         if (obj.getZmin() < zmin) {
            zmin = obj.getZmin();
         }
         if (obj.getZmax() > zmax) {
            zmax = obj.getZmax();
         }
      }
      currentZmin = zmin;
      currentZmax = zmax;
      roiCmX = (int) obj.getMassCenterX(ima);
      roiCmY = (int) obj.getMassCenterY(ima);
      roiCmZ = (int) obj.getMassCenterZ(ima);
      volume = obj.getVolumeUnit();
      comment = obj.getComment();
      roiLabel = (String) model.get(indexes[0]);

      for (int zz = zmin; zz <= zmax; zz++) {
         IJ.showStatus("Computing Roi " + zz);
         ByteProcessor mask = new ByteProcessor(imp.getWidth(), imp.getHeight());
         for (int i = 0; i < indexes.length; i++) {
            obj = objects3D.getObject(indexes[i]);
            obj.draw(mask, zz, 255);
         }

         mask.setThreshold(1, 255, ImageProcessor.NO_LUT_UPDATE);
         ImagePlus maskPlus = new ImagePlus("mask " + zz, mask);

         ThresholdToSelection tts = new ThresholdToSelection();
         tts.setup("", maskPlus);
         tts.run(mask);

         arrayRois[zz] = maskPlus.getRoi();
      }
   }
   int middle = (int) (0.5 * zmin + 0.5 * zmax);
   imp.setSlice(middle + 1);
   imp.setZ(middle + 1);
```

```
      imp.setRoi(arrayRois[middle]);
      imp.updateAndDraw();
   }

   public static void updateRois() {
      // synchronized (this) {
      plus = Image.getImage();
      if (plus != null) {
         int sl = plus.getSlice() - 1;
         if ((sl >= currentZmin) && (sl <= currentZmax)) {
            plus.setRoi(arrayRois[sl]);
         } else {
            plus.killRoi();
         }
         plus.updateAndDraw();
      }
   }

   /**
    * Description of the Method
    *
    * @param msg Description of the Parameter
    * @return Description of the Return Value
    */
   public static boolean error(String msg) {
      Frame frame = null;
      new MessageDialog(frame, "ROI Manager", msg);
      Macro.abort();
      return false;
   }
}
```

C.2.11 Segmentation.java

```
package org.annotationRoi3D.methods;

import org.annotationRoi3D.gui.InterfaceGUI;

import mcib3d.image3d.ImageHandler;
import mcib3d.image3d.ImageLabeller;
import ij.ImagePlus;
import ij.gui.GenericDialog;
import ij.measure.Calibration;

public class Segmentation extends InterfaceGUI {

   public static String title;
   static ImagePlus plus = Image.getImage();
   int low = 128;
   int high = 255;

    public static void setTitle() {
        title = plus.getTitle();
    }
    public String getTitle() {
        return title;
    }

    /**
     * Creates optional dialog box to set threshold
     * for segmentation, with default values: low 128
     * and high 255.
     *
     * @param low The low threshold parameter
     * @param high The high threshold parameter
     */
    public static void segmentation3D() {
        GenericDialog gd = new GenericDialog("Threshold 3D");
        gd.addNumericField("Low_Threshold", 128, 0);
        gd.addNumericField("High_Threshold", 255, 0);
        //gd.showDialog();
        int low = (int) gd.getNextNumber();
        int high = (int) gd.getNextNumber();

        segmentation3D(low, high);
    }


    /**
     * segmentation3D method identifies 3D
     * regions within image stack
     *
     * @param low The low threshold parameter
     * @param high The high threshold parameter
     */
    public static void segmentation3D(int low, int high) {
        plus.killRoi();
        setTitle();
        Calibration cal = plus.getCalibration();
        ima = ImageHandler.wrap(plus);
        ima = ima.threshold(low, false, true);
        ImageLabeller labels = new ImageLabeller(false);
```

```
ima = labels.getLabels(ima);
if (cal != null) {
    ima.setScale(cal.pixelWidth, cal.pixelDepth, cal.getUnits());
}
        ima.show(title);
    }
}
```