INTERACTIVE WEB-BASED TRACK EDITING AND MANAGEMENT

A Thesis presented to the

Faculty of the Graduate School

University of Missouri

In Partial Fulfillment of the

Requirements of the Degree

Master of Science

by

PHANEENDRA MADALA

Dr. Kannappan Palaniappan

MAY 2012

The undersigned, appointed by the Dean of the Graduate School, have examined the thesis entitled

INTERACTIVE WEB-BASED TRACK EDITING AND MANAGEMENT

Presented by Phaneendra Madala,

A candidate for the degree of Master of Science and hereby certify that in their opinion it is worthy of acceptance.

Dr. Kannappan Palaniappan

Dr. Jianlin Cheng

Dr. Filiz Bunyak

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Kannappan Palaniappan for his support and guidance during this project. I completely utilized every opportunity I had during my study at University of Missouri with his help and encouragement. I would also like to thank all my colleagues at the lab for their assistance and advice for this project.

I would like to thank Dr. Jianlin Cheng and Dr. Filiz Bunyak for giving me high-quality knowledge and also for taking time to serve on my project committee.

I should thank my parents for constantly motivating me during this project. I thank my brother for guiding me in a right path in all aspects.

Finally, I thank all my friends, without their support at the right moment, this wouldn’t be possible.

Table of Contents

List of Figures

List of Tables

Abstract

Image and video analysis is the process of extracting useful information from an image. With advances in **optical** and computational technologies, image and video analysis is becoming an increasingly important tool in biological, medical and defense applications. **Motion** characteristics and behaviors of objects can be studied from image sequences, such as the spatio-temporal behavior of cells and organisms in microscopy videos or vehicles and people in surveillance videos. Manual supervised and automatic object tracking are used to study migration, lineage, cellular events and population scale dynamics in microscopy applications and multi-target behavior in defense applications.

FireFly is a rich multimedia web-based tool based on Adobe Flash and Flex with server side PHP and MySQL, for managing image collections, collaborative ground truth generation, manual and supervised analysis of images and video including labeling, annotation, ground truthing, algorithm output inspection, comparison and correction, etc. The main objective of this project is to extend FireFly and add interactive editing and updating tracking information by automatically propagating the track information to upstream and downstream frames in the annotation/ video event database.

Chapter 1

Introduction

One of the main objectives of computational image/ video analysis is to automatically extract useful content information about objects/events in images and video and use this information in further reasoning, discovering and decision making. Applications range from tracking object in wide-area motion imagery [22] to studying bacterial behavior in microscopy video. The process of retrieving information is made faster by image and video analysis algorithms. The collection of data produced by these algorithms for a set of images is used by the researcher to study the spatio-temporal characteristics and behaviors of objects, such as the spatio-temporal behavior of cells and organisms in microscopy videos. During algorithm design, it is often highly desirable to have annotated ground truth information to facilitate development and testing image/ video analysis algorithms. However, due to the complexity of visual information [23], data generated by image and video analysis algorithms may produce errors that need to be manually corrected. Therefore, the data from image and video analysis needs to be validated against ground truth during the algorithm design stage [24].

In general, researchers work with large video datasets to study the behavior of objects and events, where each image may consist of tens to thousands of objects. Ground truth generation for such large video datasets will be time consuming and require an expert to perform the annotation process with few errors.

There are a few automated tools such as LabelMe [2], LabelMe video [6], Kolam [7], ViPER [1] and Allen Brain Atlas [8] available to enable faster and more accurate ground truth generation. In this thesis we discuss version 2 of **FireFly** [3] tool which is used for managing video collections, collaborative ground truth generation, and manual and supervised analysis of images and video including object/ event labeling, annotation, ground truthing, algorithm output inspection, comparison and correction and other annotation capabilities. FireFly is developed at the Multimedia and Visualization Laboratory at University of Missouri is a generic tool which allows users to work with different datasets. It is a rich interactive media web-based tool, which can be accessed from anywhere on any platform through a web browser and helps researchers from multiple domains to work collaboratively on annotating video datasets on a remote (secure) archive.

1.1A Brief Description of FireFly

Figure 1.1 represents microscopic images of bacteria and second column shows the zoomed and annotated polygon objects visualized with different colors to identify them easily. Zoomed portion is same as red colored portion of an image in the first column. Each object is tagged with their associated labels. Label represents with class ‘B’ (Bacteria) and unique ObjectID.

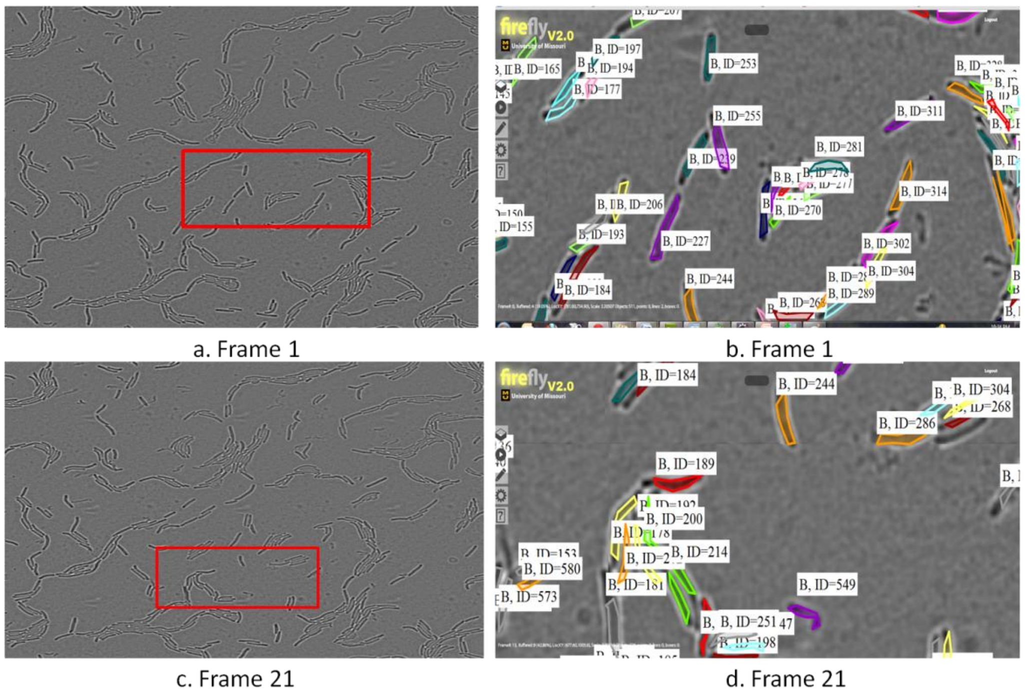
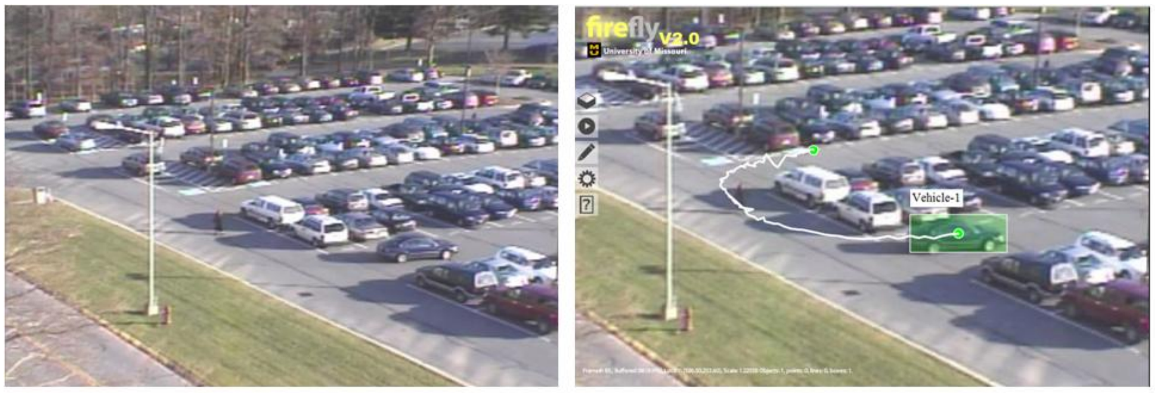
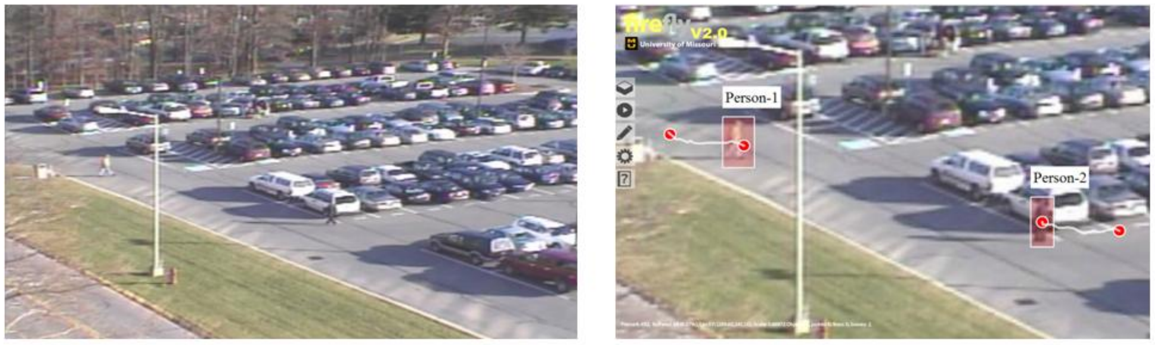


Figure 1.1: Microscopic image of Bacteria, annotated in second column

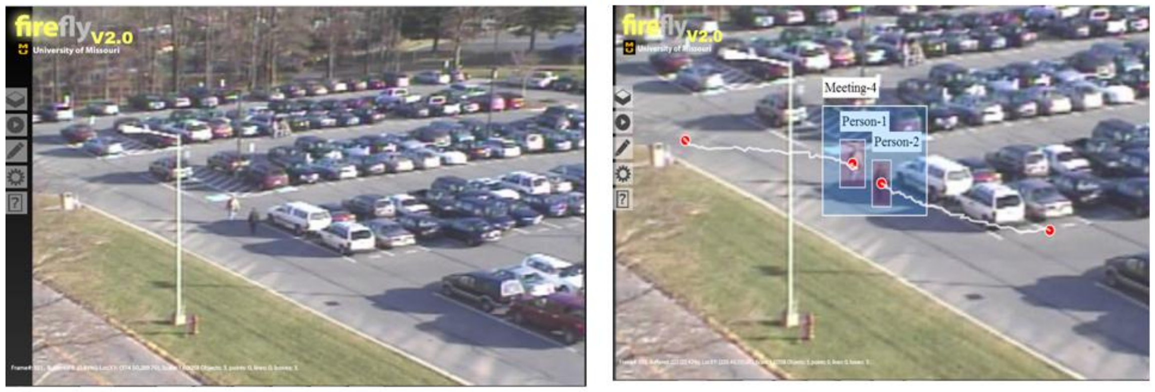
Figure 1.2 shows the annotated objects and tracking of vehicle and persons in FPSS dataset. The small circles in the trajectory/track represent the position of an object at starting and current frame of the track. The static objects in this dataset are not annotated such as parked vehicles which are not involved in any activities.



a. Vehicle parking



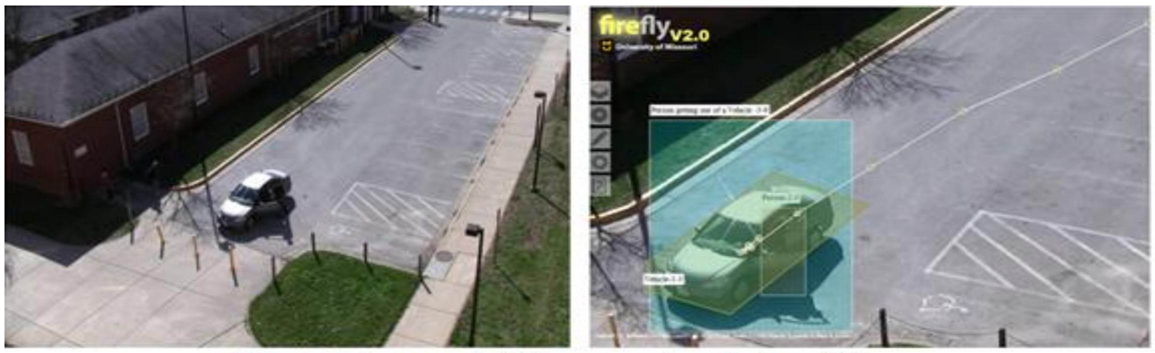
b. Persons Walking



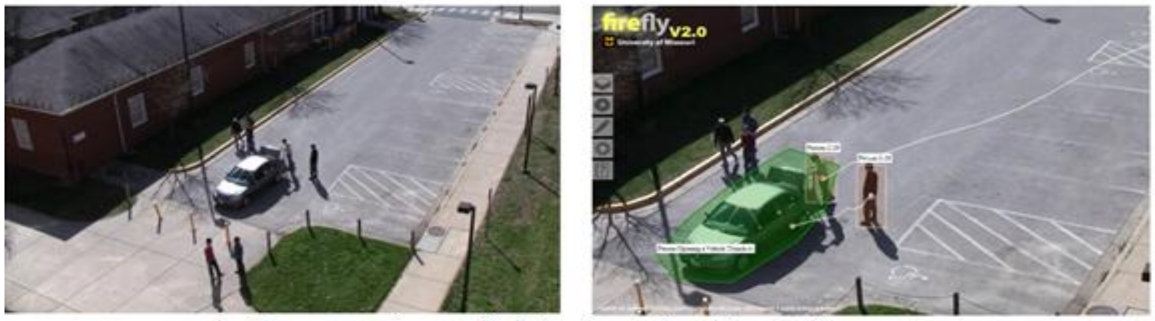
c. Persons Meeting

Figure 1.2: Annotated and tracking of objects in FPSS Dataset

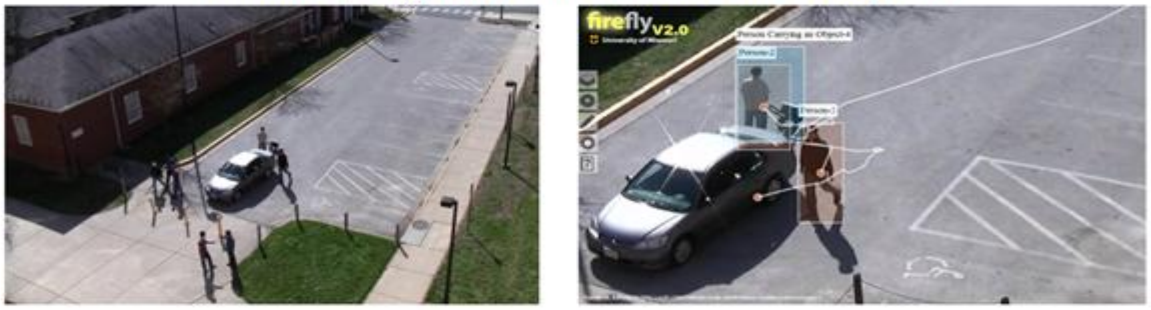
Figure 1.3 shows the annotation of 4 events and tracking of persons and vehicles in Virat dataset. Firefly classifies the events in Virat dataset into 14 catagories where each event is represented with different color. Figure 1.3a shows annotation of an event “person getting out of vehicle” in blue colored box, annotations of person, annotation of vehicle and also track of vehicle. small circles in the trajectory/track of vehicle represents the position of vehicle in its previous frames. Figure 1.3b shows annotation of an event “person opening vehicle trunk”, annotation of two persons and trajectory/track of two persons. The small circle on trajectory represents the position of that person at starting and current frames. Figure 1.3c shows annotation of an event “person carrying an object”, annotation of two persons and trajectory/track of two persons. The small circle on trajectory represents the position of that person at starting and current frames. Figure 1.3d shows annotation of an event “person getting into vehicle”, annotation of an event person carrying an object”, annotation of two persons.



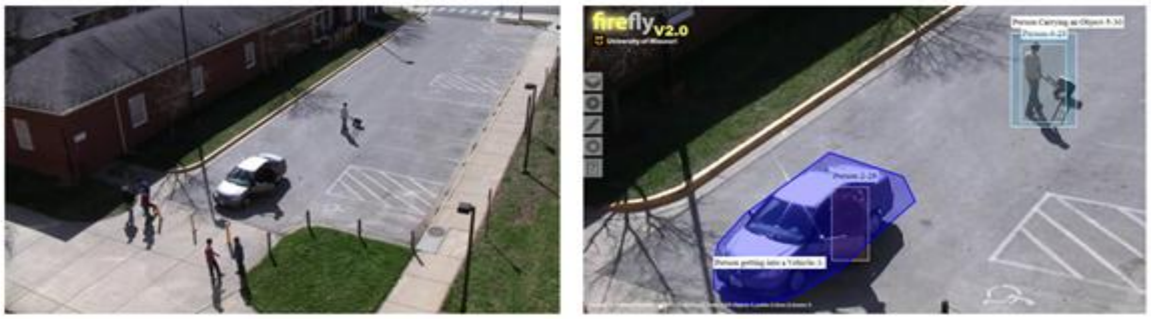
a. Person getting out of vehicle event and track of a vehicle



b. Person opening a vehicle trunk event and track of two persons



c. Person carrying an object event and track of two persons



d. Person getting into a vehicle event

Figure 1.3: Annotated and tracking of objects in Virat dataset

Figure 1.4 shows sample frames from video surveillance dataset. Objects in this dataset are buildings, road, metro and cars. Second column shows these objects annotated with colors. Third column in the Figure 1.4 shows the zoomed view of second column.

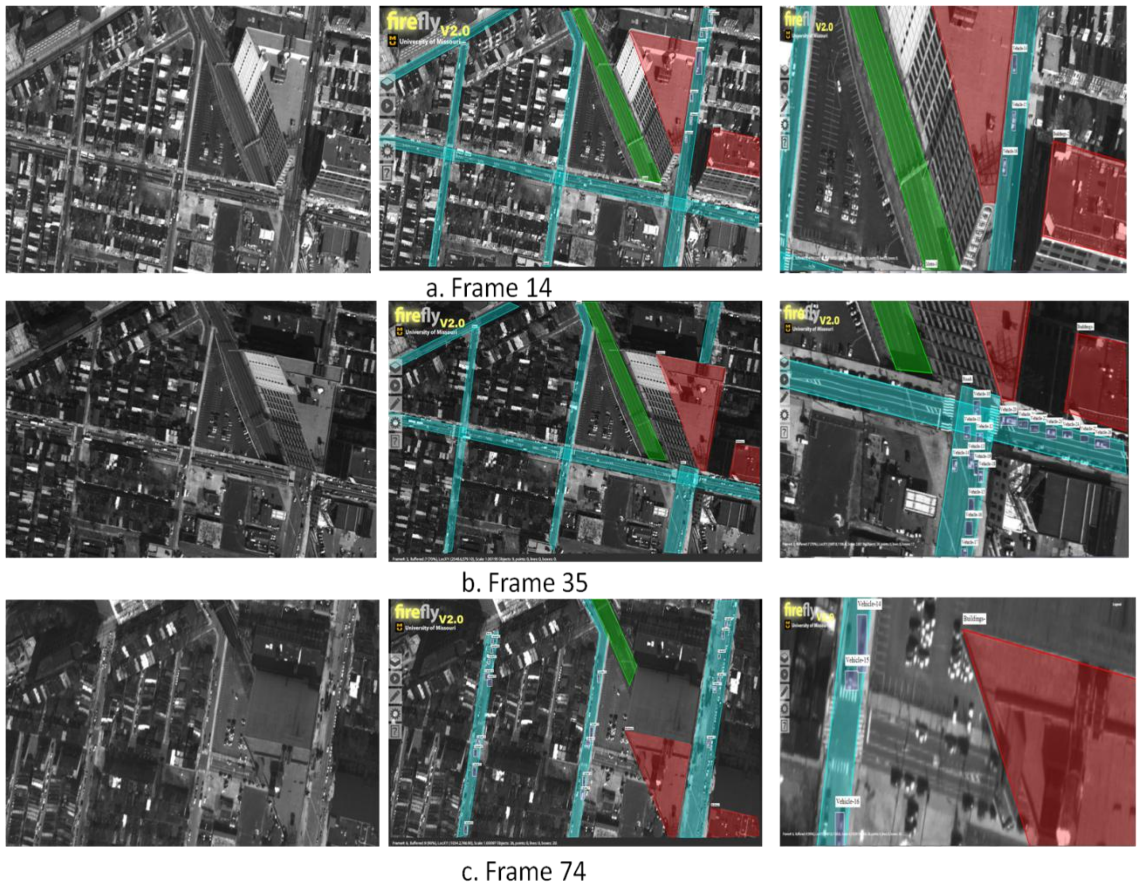


Figure 1.4: Annotated objects of Video surveillance

In general, Image and video analysis algorithms are used to identify and generate data of each object in these images. Due to the complexity of the image and video analysis task, automatic algorithms are expected to produce incorrect data. Therefore, a professional manually corrects the outcome of image and video analysis algorithms by comparing it with corresponding objects in an original image (ground truth). Each of these images may contain hundreds to thousands of objects and correcting each object will not only be time consuming but also error prone. FireFly helps the researcher to visualize the annotations (tracks, classes, segmentation and contours/mask) on the original image and provides a set of tools to edit them interactively, which reduces the time used for correction and gives more accurate data.

The general system of FireFly is represented in Figure 1.5a. FireFly front end is developed in Adobe Flex 3 and it is compiled to produce .swf file. As FireFly is a web application, it runs inside the web browser with the help of Adobe Flash Player plug-in to run the .swf format applications. All the requests on the server side are processed by Zend Framework which is open source software framework for PHP to provide reusability and modularity through Model View Controller (MVC) design. Video event database consists of tables developed and maintained using MySQL software and PHP script performs reading and writing operations on database based on the request received. The complete communication between the client and server is through binary format called Action Message Format (AMF) introduced by Adobe to maintain serialization of objects between server (PHP) and client (ActionScript) applications. Video datasets are located on the server and used by client side Flex application by sending HTTPRequest.

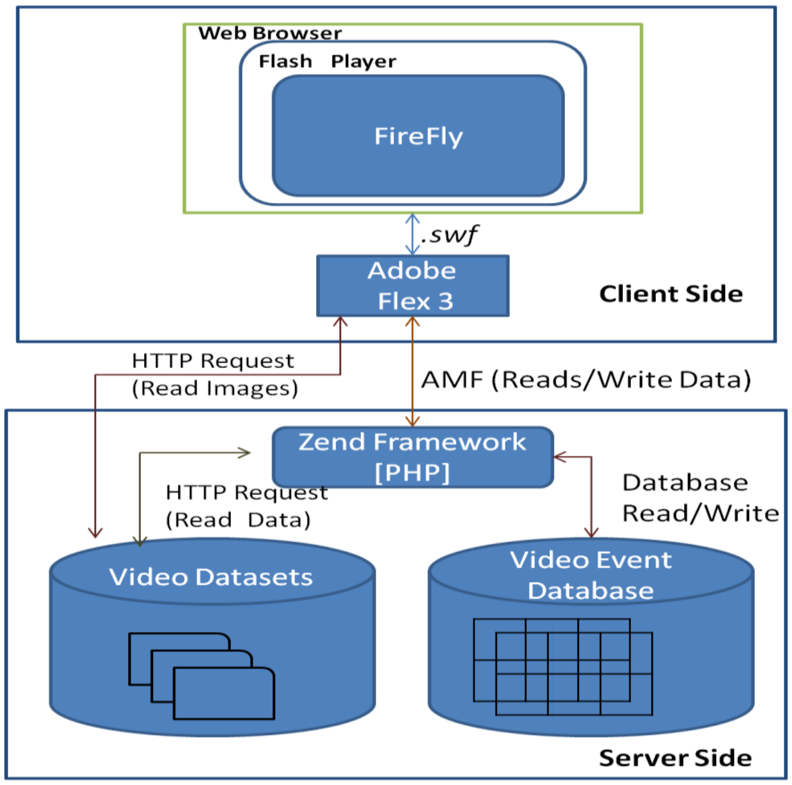


Figure 1.5a FireFly System Design

Figure 1.5b represents the hierarchical database organization of data elements in FireFly. The top most level in the hierarchy is Labs, where each user in the FireFly belongs to at least one lab. Examples of Labs would be biomedical lab, surveillance lab, etc. Each Lab consists of collection of projects related to their field and each project consist of different ImageSets (is the name used in the FireFly graphical user interface but in the database tables this element is referred to as Frameset_ImageData to be more expressive). Figure 1.5c represents the user access architecture for FireFly video/image datasets. FireFly allows multiple users to create their own annotations for a dataset and each of these annotations of a specific dataset share a same image/video sets. The creator of the annotation is called owner of the annotation and owner of the annotation can share his annotation to any FireFly user by providing read only or read/write access permissions their user ID’s. Read only permission limits user to view the annotations and restricts user to perform any edit/save/delete operations. Read/Write permissions allow user to perform edit/delete/save operations on annotations and these modifications are not restorable. In the previous versions of FireFly, Annotation is implemented with the name Dataset and Imageset is implemented with the name Frameset. But the new naming convention, Annotation is found to be appropriate for representing annotations with respect to the Imageset and similarly Imageset is observed to be appropriate for representing video/image datasets.

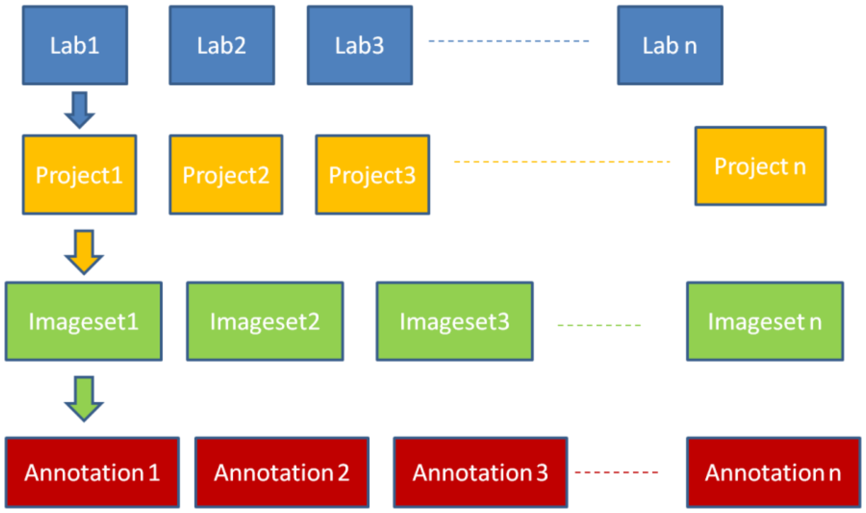


Figure 1.5b Hierarchical database organization of the data elements. ImageSet (sometimes referred to as Frameset) is the name used in the FireFly graphical user interface but in the database tables this element is referred to as Frameset_ImageData to be more expressive.

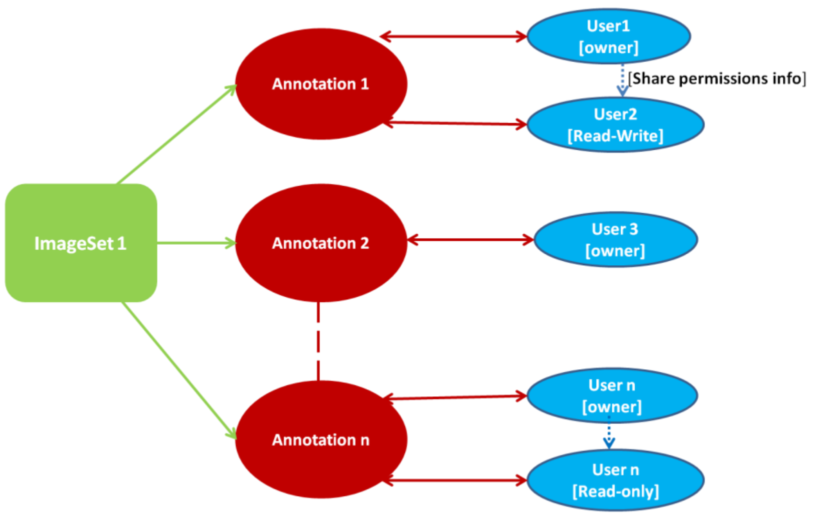


Figure 1.5c FireFly system user access

The main functionalities of FireFly include:

- Visualization: FireFly visualizes the images of original image/video datasets and also visualizes the data and properties of objects generated by image/ video analysis algorithms using graphical tools.
- Data Management: FireFly maintains central database server to provide collaborative environment. Database consists of information related to user authentication, authorization, datasets and object and event properties. FireFly effectively updates the database for every user update operation.
- Manual Segmentation: It is a process of partitioning an image into small segments, where each segment provides meaningful information from an image. Each of such segments is visualized in the FireFly using labels and boundaries. FireFly provides various object structures like point, line, box,

polygon and polyline which allow the user to interactively segment the image.

- Manual Classification: It is the process of assigning partitioned segments to one of the categories. The classification is done based on the characteristics of the object. FireFly maintains classification attributes of each object [4] and allows the user to enter these attributes for newly added objects.
- Manual Tracking of an object: The main objective of tracking is to find the spatial-temporal association of an object or group of objects in a dataset. Manual supervised and automatic object tracking are used to study migration, lineage, cellular events and population scale dynamics and also helps in understanding the classification cycle [4]. In general, track information is generated by various automated trackers, e.g. [5] and FireFly, provides interactive interface to visualize and edit this track information for ground truth generation or for correction.

Automated track generation for a critical dataset is expected to be a challenging task and requires manual edit operations by user to correct the association relationships among the objects. The manual edit operation on an object will eventually effect the track information of that object and requires propagating these updates to upstream and downstream frames in a dataset for consistency. Manual propagation of the track updates is a tedious and time consuming process. In this project, efficient algorithms were introduced, to automatically update and propagate the track information based on the user edit operation on one object.

1.2 Video Annotation Tools

Video annotation tools help the researchers to gather ground truth and also help in validating the image/video analysis algorithms. Some of these tools such as NeuronJ and DCellIQ were discussed in [3]. In this section we discuss other tools such as LabelMe [2], LabelMe Video [6], Kolam [7], Viper [1], Allan Brain Atlas [8], OMERO.Web [25] and Bisque [26].

1.2.1 LabelMe

LabelMe [2] is a database and an online annotation tool for the purpose of object detection and recognition research. LabelMe was created by Computer Science and Artificial Intelligence Laboratory at the Massachusetts Institute of Technology. Large sets of images or video are needed for object detection and recognition study therefore; they introduced an online tool which helps in building annotation for datasets from large population of users. This tool is designed in Javascript to provide an online drawing interface. It provides polygon drawing tool to segment the objects and allows a user to label the segmented object. The resulting labels are stored in XML file format [2]. A sample interface is shown in Figure 1.6.

A Matlab toolbox is also provided to perform operations on annotations, query datasets, communicate with online tool and etc.

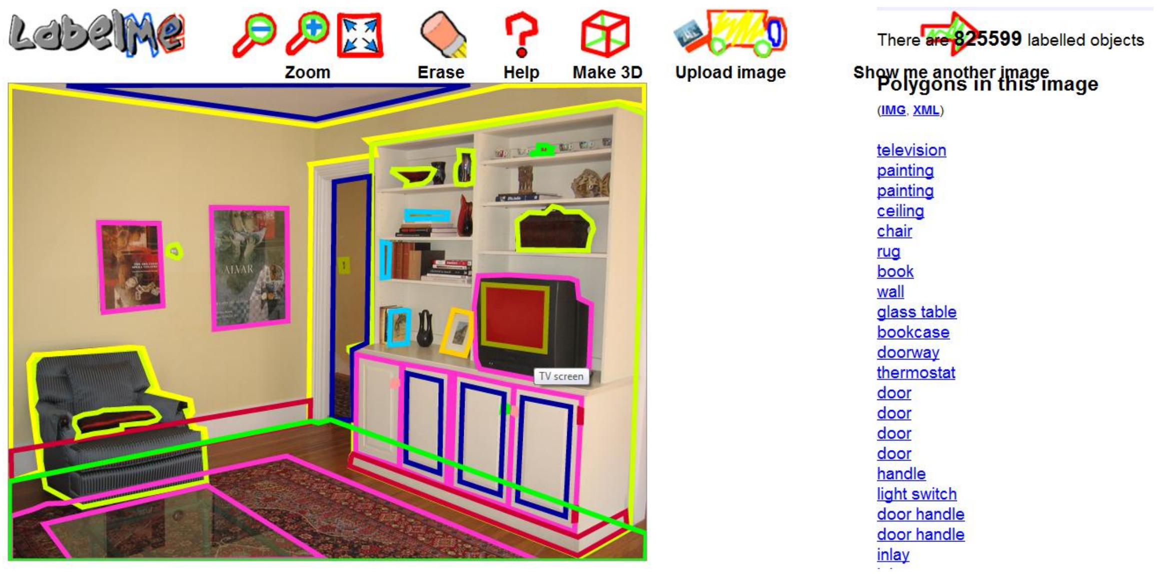


Figure 1.6 Interface of LabelMe: Shows the segmented objects [5]

However, there are many concerns in this annotation collection method. One of them is the complexity of polygons provided by the use, i.e. user may provide simple or complex boundaries for an object. Another issue is labeling, e.g. there are multiple ways to label an object. These decisions are left to the user to know the different ways of thinking in segmenting an image. This tool is used for ground truth generation. This tool is not used for object classification and object tracking.

1.2.2 LabelMe video

LabelMe video [6] is an online video annotation tool which is used to create an open database of videos where users are allowed to upload, annotate and download. This annotated video database is used to obtain statistics of moving and static objects, and information regarding their interactions, which are helpful in tracking and activity recognition applications, e.g. video surveillance. It consists of two main features, object annotation and event annotation. Object annotation is similar to the annotation functionality in LabelMe, which consists of segmentation and its motion.

User is allowed to annotate the object using polygon drawing and these polygon boundaries are propagated throughout the sequence. The user is also allowed to navigate through the video and edit the polygons propagated across different frames. Event annotation is designed to annotate the interaction between the objects. LabelMe video interpolates the missing polygons between key frames using interpolation techniques [6]. LabelMe video provides tracking of objects in a frame but it doesn't support the lineage tracking, such as tracking in case of cell division. FireFly helps in maintaining and propagating the track information to daughter cells when there is a cell division. Figure 1.7 shows the interface of LabelMe video. There are three objects which are annotated in the video with their description.

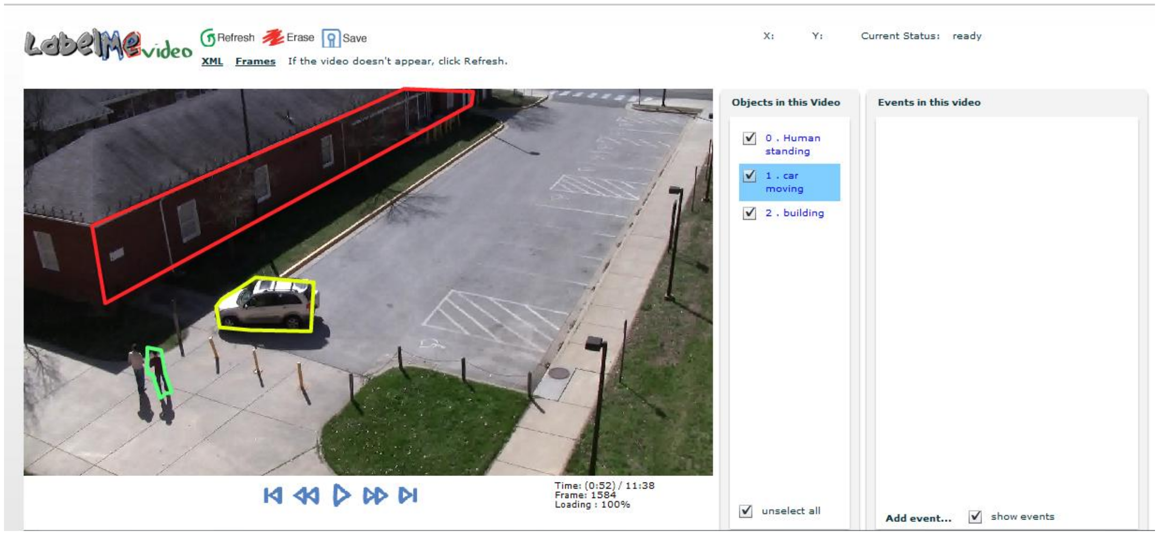
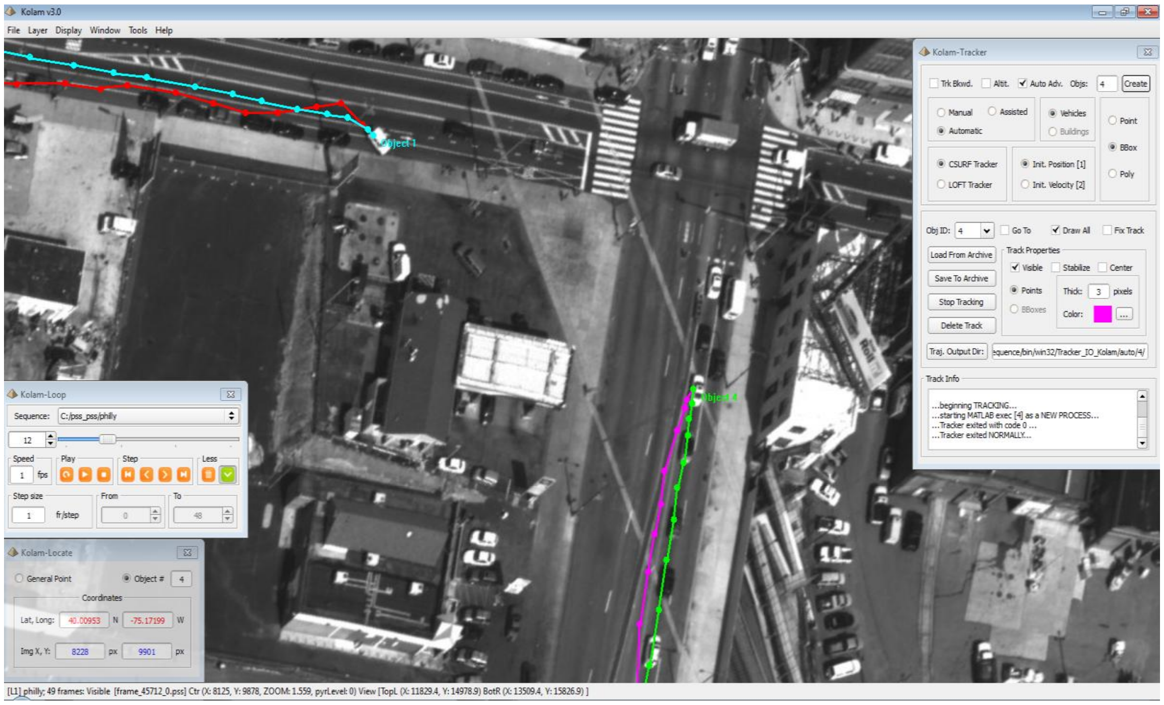


Figure 1.7 Interface of LabelMe video [6]

1.2.3 KOLAM

KOLAM [7] is an open source, extensible architecture for visualization and tracking wide-area motion imagery created by the University of Missouri. KOLAM uses Qt API and UI framework to provide interactive tracking and trajectory management for WAMI datasets. Tracking in KOLAM can be accomplished in 3 ways: automated object tracking by interfacing with various external tracking algorithms and visualizing the resulting trajectories; manual tracking used for generating ground truth, and assisted object tracking for editing and trajectory drawing [7]. A sample object tracking using KOLAM is shown in the Figure 1.8.



Trk Blwd. [ ] Altit. [ ] Auto Adv. [✓] Objs: 4
Manual, Assisted, [✓ Automatic]; [✓ Vehicles], Buildings; Point, [✓ BBox], Poly
[✓ CSURF Tracker], LOFT Tracker; [✓ Init. Position [1]], Init. Velocity [2]
Obj ID: 4; Go To [ ]; Draw All [✓]; Fix Track [ ]
Visible [✓]; Stabilize [ ]; Center [ ]; [✓ Points], BBoxes; Thick: 3 pixels; Color: 🟪
Traj. Output Dir: equence/bin/win32/Tracker_IO_Kolam/auto/4/

```
...beginning TRACKING...
...starting MATLAB exec [4] as a NEW PROCESS...
...Tracker exited with code 0 ...
...Tracker exited NORMALLY...
```

Sequence: C:/pss_pss/philly; 12; Speed 1 fps; Step size 1 fr/step; From 0; To 48
General Point, [✓ Object #] 4
Lat, Long: 40.00953 N -75.17199 W
Img X, Y: 8228 px 9901 px

```
[L1] philly; 49 frames: Visible [frame_45712_0.pss] Ctr (X: 8125, Y: 9878, ZOOM: 1.559, pyrLevel: 0) View [TopL (X: 11829.4, Y: 14978.9) BotR (X: 13509.4, Y: 15826.9) ]
```

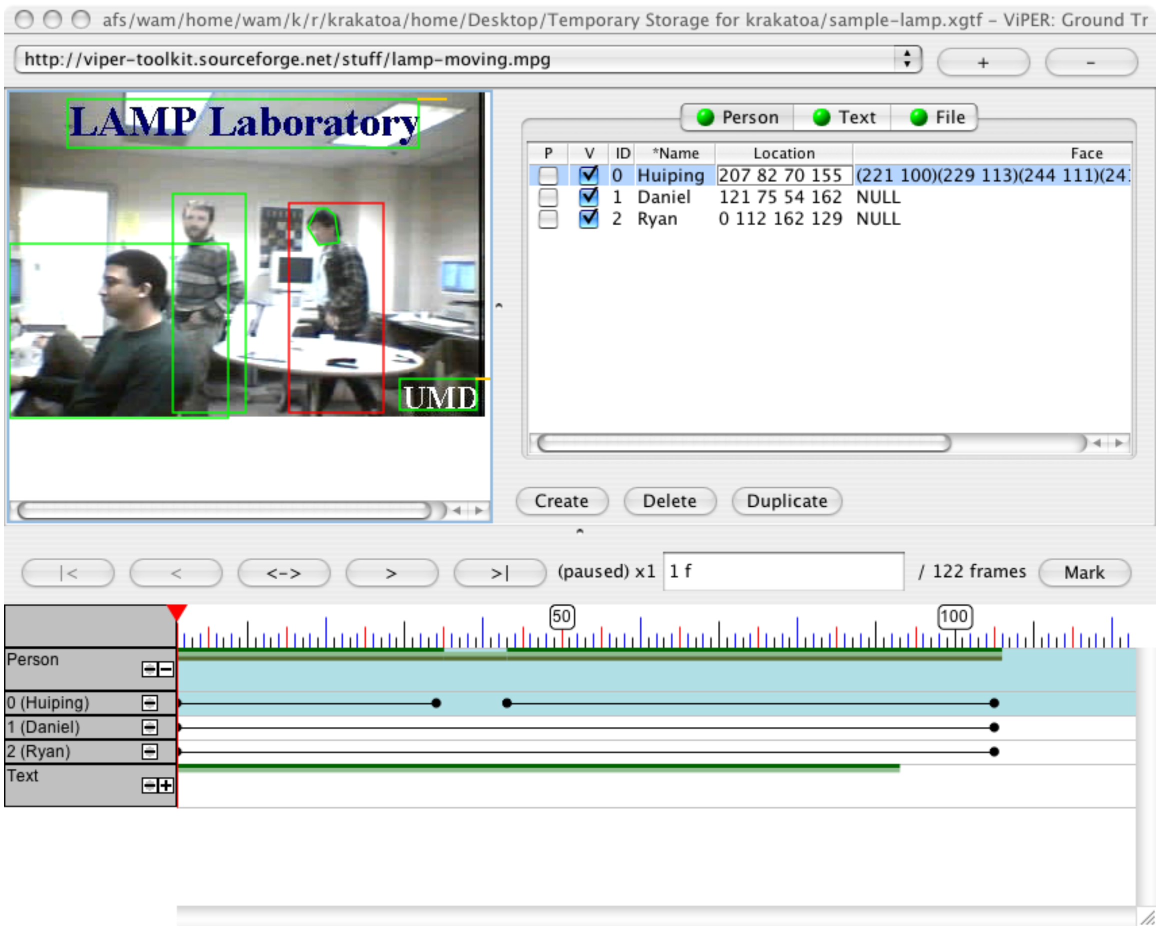
Figure 1.8 Interface of KOLAM [7]

KOLAM is mainly designed to support large datasets scalable from hundreds of gigabytes to petabytes in size. To display these images efficiently, KOLAM partitions or tiles the images to subimages of the larger dataset through the process of a quad-tree like regular tiling. These tiles represent the partitioned non-overlapping regions of the image and allow on-demand access to those tiles that are required to display. KOLAM is a desktop application, that limits the users to install software locally and the extracted data are saved locally which again limits researchers to collaborate on a single dataset.

1.2.4 ViPER

Language and Media Processing Lab, at university of Maryland introduces a video analyzing tool called Video Performance Evaluation Resource, ViPER [1]. It provides interface for tracking people and detecting text. ViPER-Ground

Truth and ViPER-Performance Evaluation are two tools available to provide ground truth and compare the results of analysis with ground truth. It provides various shapes like rectangle, ellipse, point, etc to annotate the ground truth [1]. Sample video tracking interface of ViPER-GT is shown in the Figure 1.9



afs/wam/home/wam/k/r/krakatoa/home/Desktop/Temporary Storage for krakatoa/sample-lamp.xgtf – ViPER: Ground Tr

http://viper-toolkit.sourceforge.net/stuff/lamp-moving.mpg

P	V	ID	*Name	Location	Face
☐	✓	0	Huiping	207 82 70 155	(221 100)(229 113)(244 111)(24
☐	✓	1	Daniel	121 75 54 162	NULL
☐	✓	2	Ryan	0 112 162 129	NULL

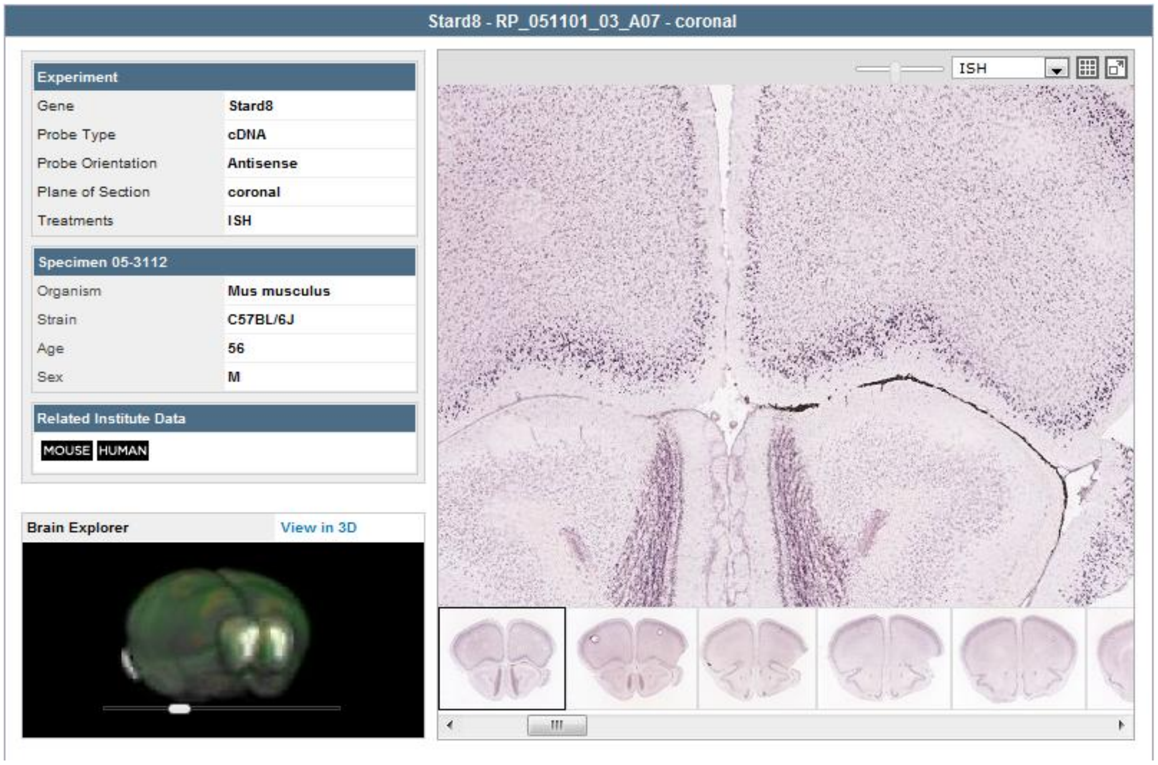
(paused) x1 1 f / 122 frames

Figure 1.9 Interface of ViPER [1]

This tool is developed in Java and the data is stored in XML format which is later used by the ViPER-PE to compare the analysis data. It provides an API for other applications to interface and use this data. However, Viper is a desktop application, which requires local installation of the software and need the videos to be on a local host to process.

1.2.5 Allen Brain Atlas

Allen Brain Atlas, ABA [8] is a web-based, three-dimensional atlas of gene expression. It provides visualization tools for studying *in situ* hybridization-based (ISH) expression patterns. It enables users to visualize gene expression data from the mouse brain in 3D at a resolution of 100μm^3. User can download ISH data which is in XML format to their own computational environment via API or via freely available software, Brain Explorer [9] provided by ABA. It segments the anatomic regions using smooth overlay of polygons and provides shading regions for better visualization. It enables mouse operations on high resolution images such drag, rotate, zoom in, zoom out, and toggle operation on segmentation [8].


Stard8 - RP_051101_03_A07 - coronal

Experiment	
Gene	Stard8
Probe Type	cDNA
Probe Orientation	Antisense
Plane of Section	coronal
Treatments	ISH

Specimen 05-3112	
Organism	Mus musculus
Strain	C57BL/6J
Age	56
Sex	M

Figure 1.10 Interface of Allen Brain Atlas [8]

In comparison with FireFly, ABA visualizes the images and provides better interface for viewing the data in 3D but, does not provide tools for a user to manually annotate the objects. Whereas, FireFly visualizes the data and provides various tools for a user to annotate the image.

1.2.6 OMERO.web

OMERO (OME Remote objects) [25] is open source software introduced by Open Microscopy Environment (OME) for the storage and manipulation of microscopy data. OMERO.web is a web application used for annotation, visualization and management of microscopy data. OMERO.web supports many of the features from their desktop application called OMERO.insight. OMERO.web also supports the visualization of big images which can be panned and zoomed and provides good interface for user to browse the images and details the metadata of each image. OMERO.web allows user to open multiple images which helps in comparing the data [25]. The sample interface of OMERO.web is shown in the Figure 1.11.

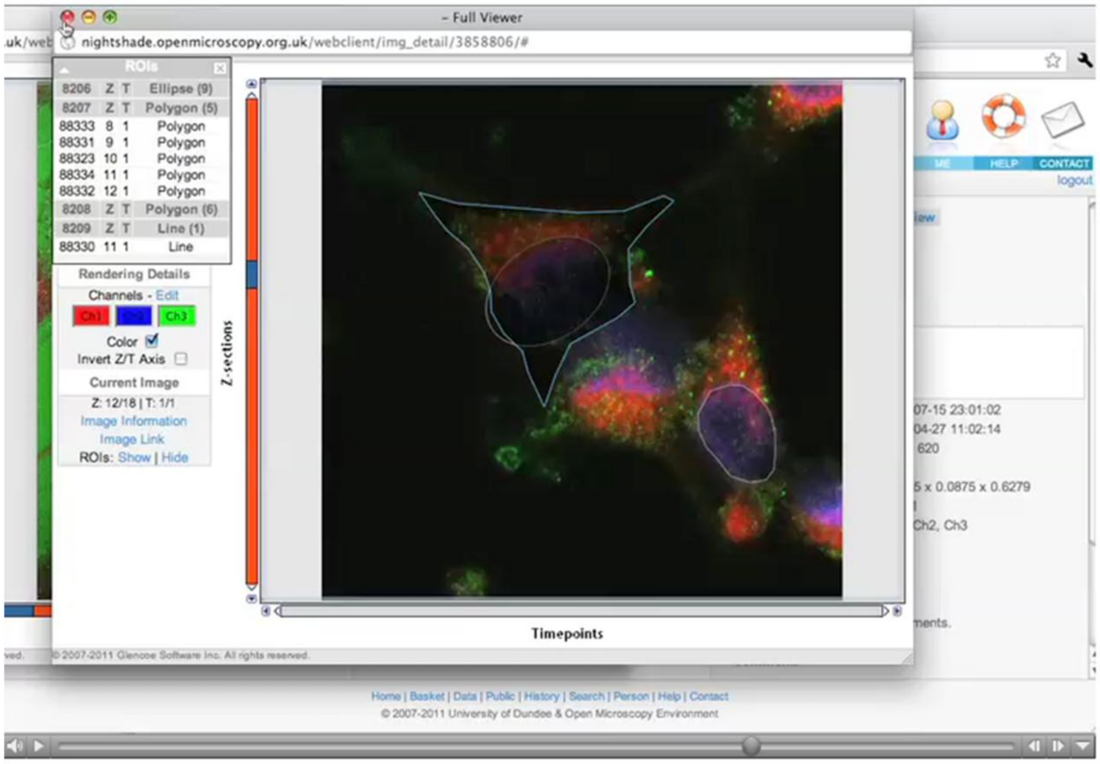
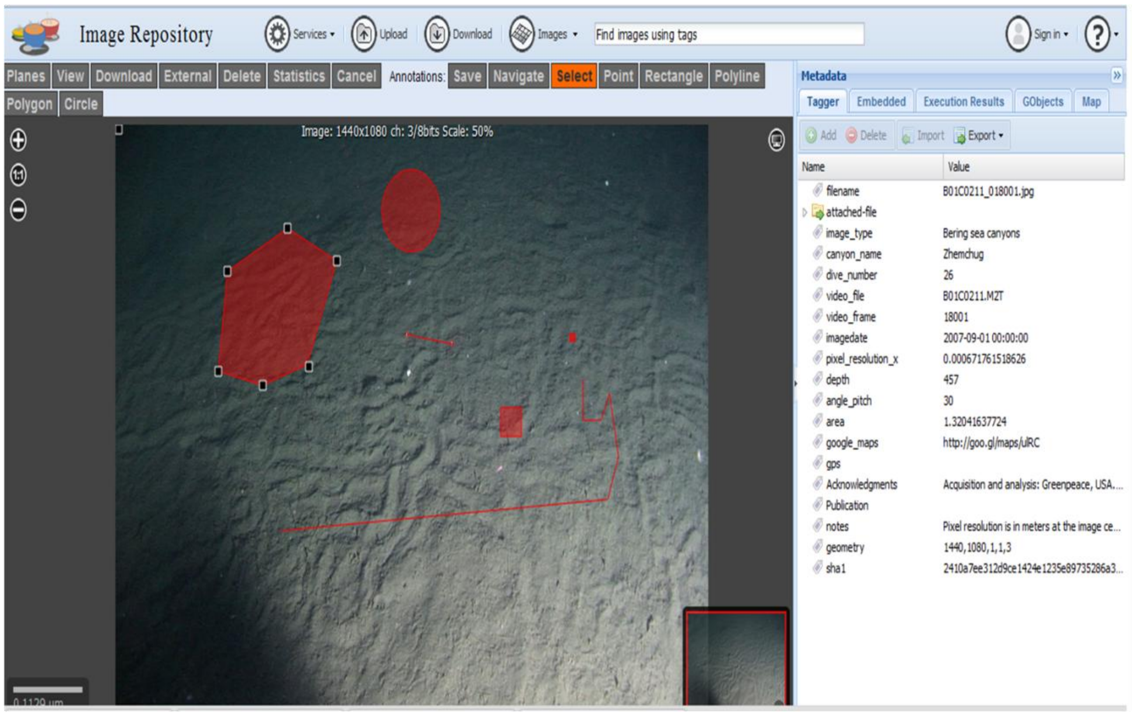


Figure 1.11 Sample interface of OMERO Web [25]

OMERO.web is an up-and-coming tool which currently visualizes the server data and ROI (Region of Interest) which are annotated in OMERO.insight and editing of ROI is yet to be supported.

1.2.7 Bisque (Bio-Image Semantic Query User Environment)

Bisque [26] is introduced by University of California Santa Barbara. It is a web tool which provides researchers with organizational and quantitative analysis tools for 5D image data. In addition to image database management, tool also provides imaging, annotation of text and graphics, analysis and visualization of results. Bisque provides internal and external analysis and visualization tools. Internal analysis methods are executable through web whereas external analysis requires additional computation resources and feature which are not accessible through the web [26]. Results of analysis are stored in the form of tags, images and graphical objects. Sample interface of Bisque is shown in the Figure 1.12.



Name	Value
filename	B01C0211_018001.jpg
attached-file	
image_type	Bering sea canyons
canyon_name	Zhemchug
dive_number	26
video_file	B01C0211.M2T
video_frame	18001
imagedate	2007-09-01 00:00:00
pixel_resolution_x	0.000671761518626
depth	457
angle_pitch	30
area	1.32041637724
google_maps	http://goo.gl/maps/uIRC
gps	
Acknowledgments	Acquisition and analysis: Greenpeace, USA....
Publication	
notes	Pixel resolution is in meters at the image ce...
geometry	1440,1080,1,1,3
sha1	2410a7ee312d9ce1424e1235e89735286a3...

Figure 1.12 Sample interface of Bisque [26]

Bisque web components are built in AJAX and the communication among various components is achieved using REST services like HTTP requests with XML and JSON.

Chapter 2

FireFly User Interface (Version 2)

FireFly is a tool used for 3 purposes: visualization, manual ground truth generation and editing/correction.

2.1 Visualization

Firefly visualizes the original dataset/image sequence along with various associated annotations (tracks, classes, segmentation and contours/mask). FireFly visualizes these annotations using graphical objects/structures such as points, lines, boxes, polylines and polygons.

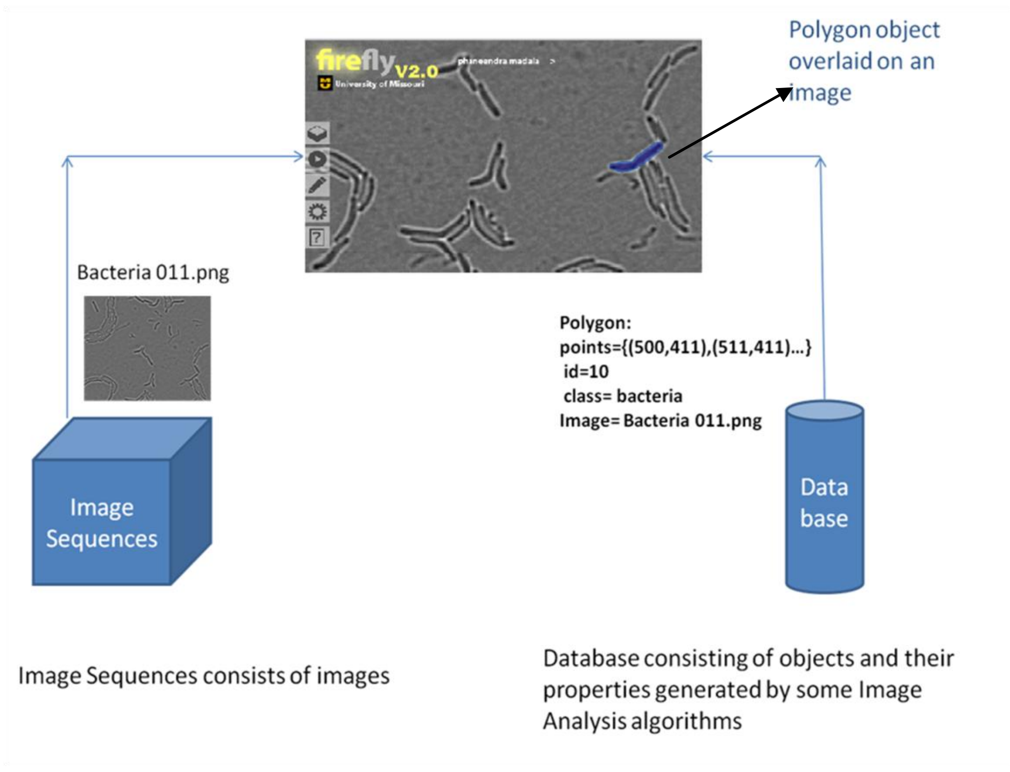

Polygon object overlaid on an image
firefly V2.0
University of Missouri
phaneendra madala
Bacteria 011.png
Polygon:
points={(500,411),(511,411)...}
id=10
class= bacteria
Image= Bacteria 011.png
Image Sequences
Data base
Image Sequences consists of images
Database consisting of objects and their properties generated by some Image Analysis algorithms

Figure 2.1 Visualization of data using FireFly

Figure 2.1 shows the visualization of data using FireFly. Image sequences consist of original images required for the validation and database consists of segmentation information of the objects, such as co-ordinates of contours, classification information and track information. FireFly tool reads the image from image sequences and uses interactive tools to represent the data in the database on an image.

2.2 Manual Ground Truth Generation

FireFly provides all the required graphical tools, point, line, box, polygon and polyline to represent the objects as shown in the Figure 2.2. These objects help the user to generate the ground truth manually. Every object consists of an attribute window which stores the attributes related to the classification and tracking information of an object such as temporal, difficulty, class name, etc.

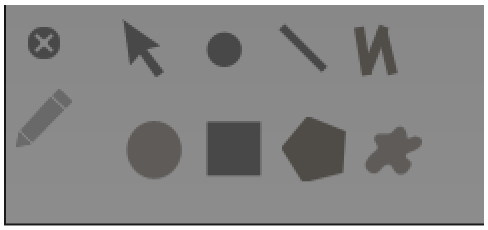


Figure 2.2 FireFly graphical tools to represent objects

2.3 Editing/Correction

Sometimes the data generated by automated trackers or algorithms may have errors, in which case, the user is provided with an interactive interface to make corrections to the existing annotations.

Correction of data using FireFly involves:

- Segmentation Correction: In segmentation process each object is clearly located by drawing contours/skeleton of an object. Table 2.1 lists all possible errors in segmentation and the operations required to correct them. Figure 2.3 depicts the segmentation error of incorrect merge, where two objects are segmented as one object. Split operation is not yet provided, and therefore, the entire segment is deleted and two objects are created to correct the segment.

Segmentation Problem	Action	FireFly operation
Spurious point	Delete	Delete key or Delete button
Missing object	Add	Select and Drop necessary object from tool widget
Incorrect location	Edit – Move objects	Click operation to select and edit by dragging the object
Incorrect shape	Edit – Move points in the object	Select point in a object and edit by dragging the point. Available for point, line and box objects
Incorrect Split	Edit – Select two objects to merge	Not yet implemented
Incorrect Merge	Edit – Select object and split	Not yet Implemented

Table 2.1 Possible errors and operations in segmentation (implemented solutions are shown in bold)

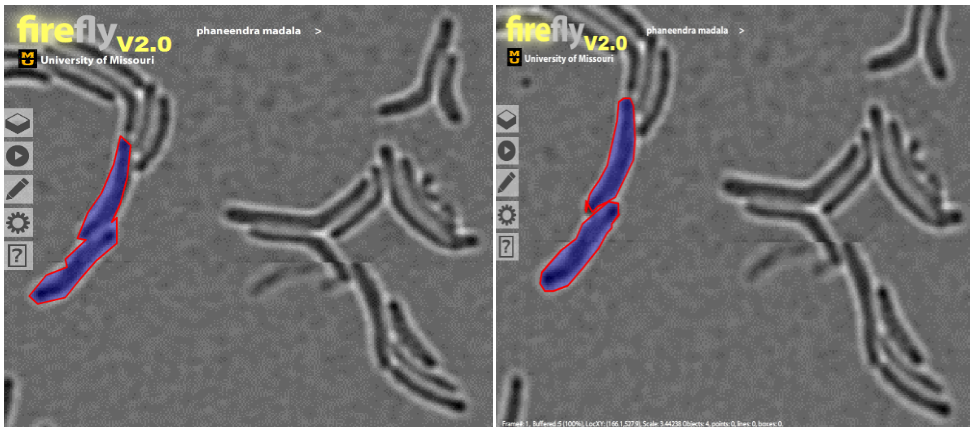
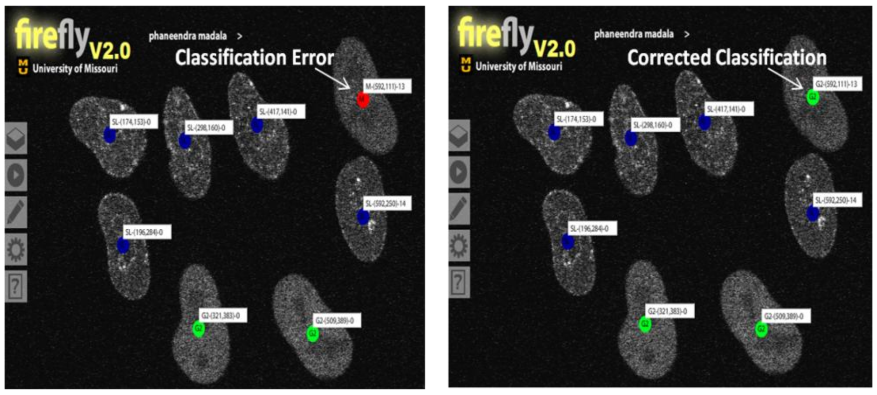


Figure 2.3: Segmentation correction in Bacteria dataset

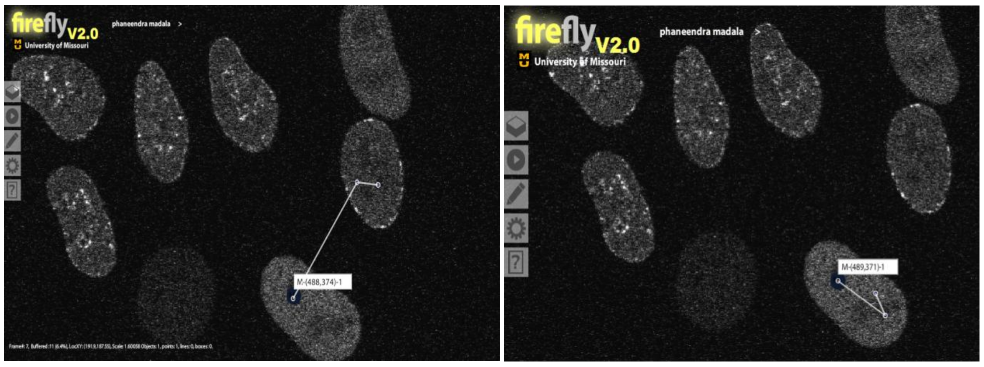
- Classification Correction: For classifying objects, FireFly automatically visualizes the objects using different class labels and colors based on the data from automated classification methods [5]. FireFly allows the user to interactively edit the class of an object. Figure 2.4 shows the classification correction on FireFly.



(a) (b)

Figure 2.4 Classification correction in HeLa cells dataset

- Tracking Correction: Tracking is done by linking segmented objects from Frame To frame in the image sequence. This tracking information is stored in the object attributes and is used in drawing the trajectory of an object. Therefore, it is important for a user to verify the track of an object in a frameset. When a user makes changes to the object attribute information, FireFly will update the information automatically and propagate it through to previous and succeeding frames to maintain correct track data. The track correction visualization in FireFly is shown in Figure 2.5.



(a) (b)

Figure 2.5 Track corrections in HeLa cell dataset

2.4 Types of Datasets and Objects

The main characteristic of FireFly is, it being a generic tool. In order to be a generic tool it needs to supports different datasets. To begin with, FireFly was developed to support the datasets which are primarily required by Multimedia and Visualization Laboratory at University of Missouri. Some of the datasets supported by the applications are listed in table 2.2.

DATASETS	**OBJECTS**
Cell Labeling/Tracking	Points
Cell Segmentation	Contours
Bacteria Labeling	Polylines, Polygons
Bacteria segmentation	Polylines, Polygons
Vessel Segmentation	Lines, Contours, Boxes
Histopathology	Filled Polygons, Contours
Cell Segmentation	Contours
Satellite Image Classification	Points, Boxes, Lines, Polylines and

	Polygons
Surveillance Video Labeling	Points, Boxes, Lines, Polylines and Polygons
Wide-area Video Labeling	Points, Boxes, Lines, Polylines and Polygons

Table 2.2: List of datasets supported by FireFly [3] (currently supported datasets are shown in bold)

FireFly supports labeling and annotation of biological, medical and defense imagery. It is also flexible to add new components to this tool to support more datasets. A sample user interface of FireFly is shown in Figure 2.6

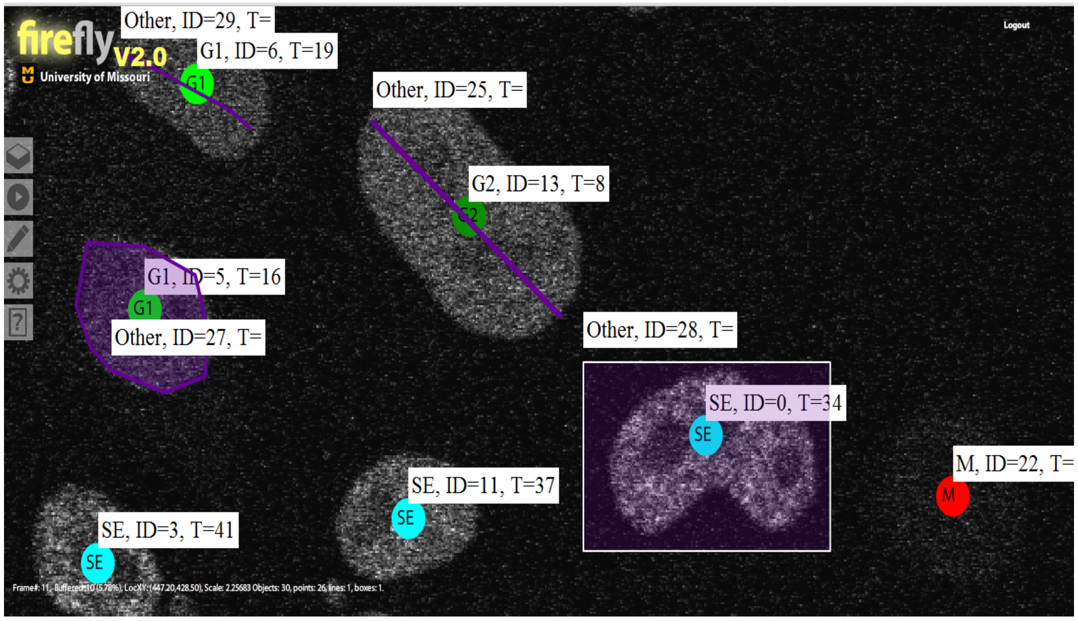


Figure 2.6 Sample FireFly interface

2.5 Technology

FireFly allows researchers from various domains to use the tool for ground truth generation. If FireFly was to be designed as a desktop application, multiple platform support is required, which repeatedly increases the cost of maintenance. Considering this issue, FireFly was developed as a web application. Web applications offer many other advantages over desktop applications namely:

- Web applications can be used anywhere in the world by connecting to the internet.
- No local installation of software required
- It is easy to add or upgrade features without disturbing the existing version
- It is platform independent and
- Easy to develop

But one of the primary advantages of developing a desktop based application is, it provides a very rich set of user interface as compared to web applications. Also, desktop applications will use the same screen and exchanges the data behind it, which reduces the time for displaying the output.

However, this limitation is overcome by Rich Internet Applications (RIA) introduced by Macromedia in the year 2002. Macromedia addresses some of the important features that rich client technologies must include [10]. Rich Internet Applications is defined as “combining the best user interface functionality of desktop software applications, with the broad reach and low-cost deployment of Web applications and the best of interactive, multimedia communication” [11]. But, most of these applications require the installation of software, which runs the applications on a browser. The biggest advantage of using this software is ease of development. This type of approach is generally referred to as “Sandbox” approach, it allows developers to understand the platform for development and assure the user to run applications in the same way on different browsers [12]. Some of the platforms which have a major market are Adobe, HTML 5, Java and Microsoft Silverlight. With the advantages of Adobe over other platforms, FireFly is developed using Adobe Flex. The advantages of Adobe Flex over other platforms are discussed in section 2.5.1 [13].

2.5.1 Comparing Platforms with Adobe Flex

- **HTML/JavaScript/Ajax:**

  HTML / JavaScript are not consistent across browsers and require more complex programming as compared to Adobe Flex. Whereas, Flex runs the same applications on all web browsers using Adobe Flash Player plug-in [13].

- **JSF:**

  Java Server Faces (JSF) works on server side, whereas Adobe Flex works on client side. Also, it is very complicated to develop complex UI using JSF as it requires additional component libraries [13].

- **Microsoft Silverlight:**

  Microsoft Silverlight is a framework for rich Internet applications. It uses XML based language XAML. However, it is owned by Microsoft and is not an open source framework as compared to Adobe Flex. Third party Flex has the advantages of components that are regularly contributed by the open source community [13].

Tools like FireFly must provide rich user interface with different widgets to experience the same usability as a desktop application to provide:

- Interactivity for the user to generate the ground truth
- Support multimedia to provide video streaming, images and graphical content and finally
- Faster performance

FireFly incorporates the characteristics of RIA to meet the above requirements.

2.5.2 Adobe Flex

Adobe Flash is a platform used to support animation, video and rich internet applications. Flash supports Shockwave Flash (SWF) file format, which has .swf file extension. Adobe Flex is a software developing kit (SDK) that allows developers to build rich internet applications easily and rapidly on Flash platform. Flex is expected to perform well in all the foundations listed in [14]. One of the main advantages of Flex is it separates the presentation and data access layers. It is designed around XML and data can be read and written using simple HTTPService call, which separates it from backend server technology like PHP, J2EE, .NET, etc.

2.5.3 Flex Framework

Figure 2.7 shows Flex framework architecture which mainly consists of class libraries, MXML, ActionScript. MXML is the heart of the Flex framework.

MXML is a markup language based XML introduced by Adobe. Flex uses two languages ActionScript and MXML to build rich Internet applications. MXML is used for layout, structuring and ActionScript [15] that controls the MXML components [13].

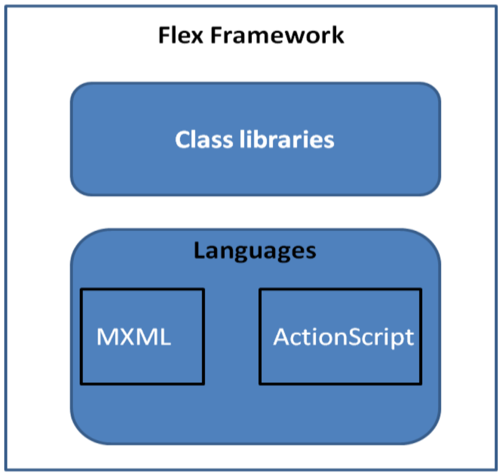


Figure 2.7 Flex Framework Architecture [13]

The libraries provide APIs for components such as controls, containers and remote service objects. In MXML compilation process, MXML file gets converted to ActionScript class and this is compiled to .swf file format as shown in Figure 2.8. Sample code of action script and MXML code is shown in Figure 2.9.

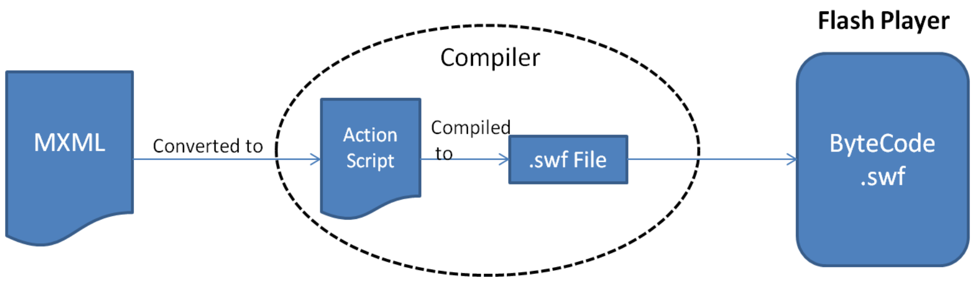


Figure 2.8 Flex Compilation Processes [13]

```
<?xml version="1.0" ?>
<mx:Application xmlns:mx="http://www.macromedia.com/2003/mxml">
<mx:Panel id="reader" title="Sample Application" width="500">
    <mx:Script>
            <![CDATA[
                        private function display():void{

                                    body.Text= 'you said' +body.Text;
                        }
            ]]>
    </mx:Script>

 <mx:TextArea id="body" editable="false" height="300" text="Hello World" />
<mx:Button label="submit" horizontalCenter="90" verticalCenter="21" click="display();"/>
</mx:Panel>
</mx:Application>
```

Figure 2.9: Sample MXML and action script code

2.5.4 Working with the server

Flex applications are client based, therefore they require access to a Web server. The Flex communication with the server is shown in Figure 2.10.

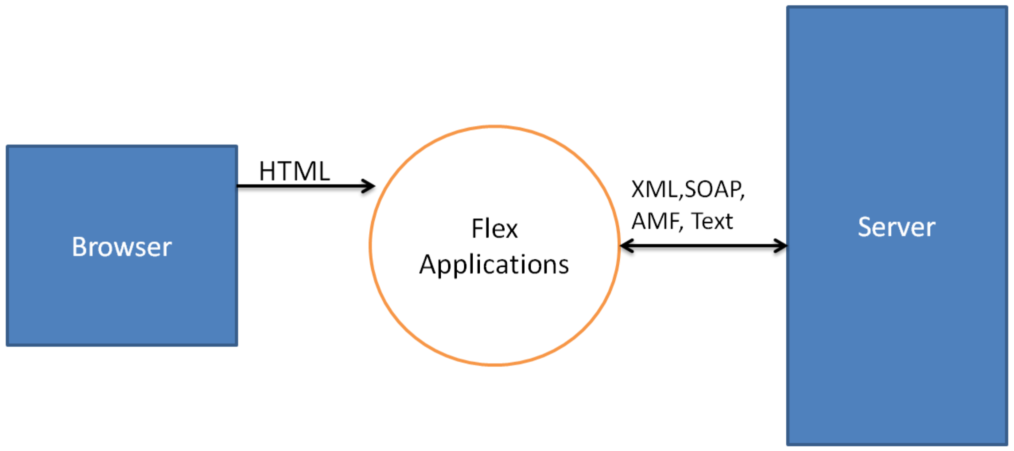


Figure 2.10: Flex communication with the server [13]

In the above figure, Flex applications which are embedded within the HTML file contains the logic to communicate with the server using Web services

(SOAP and XML) or HTTP Services using simple text or XML. Flex uses Action Message Format (AMF) to directly communicate with Web servers by remote object invocation APIs.

2.5.5 Frameworks for Flex

Frameworks improve the efficiency of creating new software. Developers can increase productivity by concentrating on the requirements of the application rather than the application infrastructure. Frameworks provide a set of libraries, intended for reuse. While the libraries and their methods are invoked by the user, the framework defines flow of control for applications. Cairngorm and Swiz are the two frameworks used in FireFly. Cairngorm is one of the main open source frameworks for Flex applications [16]. It is based on MVC model architecture. Model layer in Cairngorm holds data objects and their state and the View layer represents the graphical user interfaces. View layer communicates with controllers through events and binds the data to match Model layer [21]. This framework allows designers, developers and data service developers to work in parallel. Micro architecture of Cairngorm is shown in the Figure 2.11.

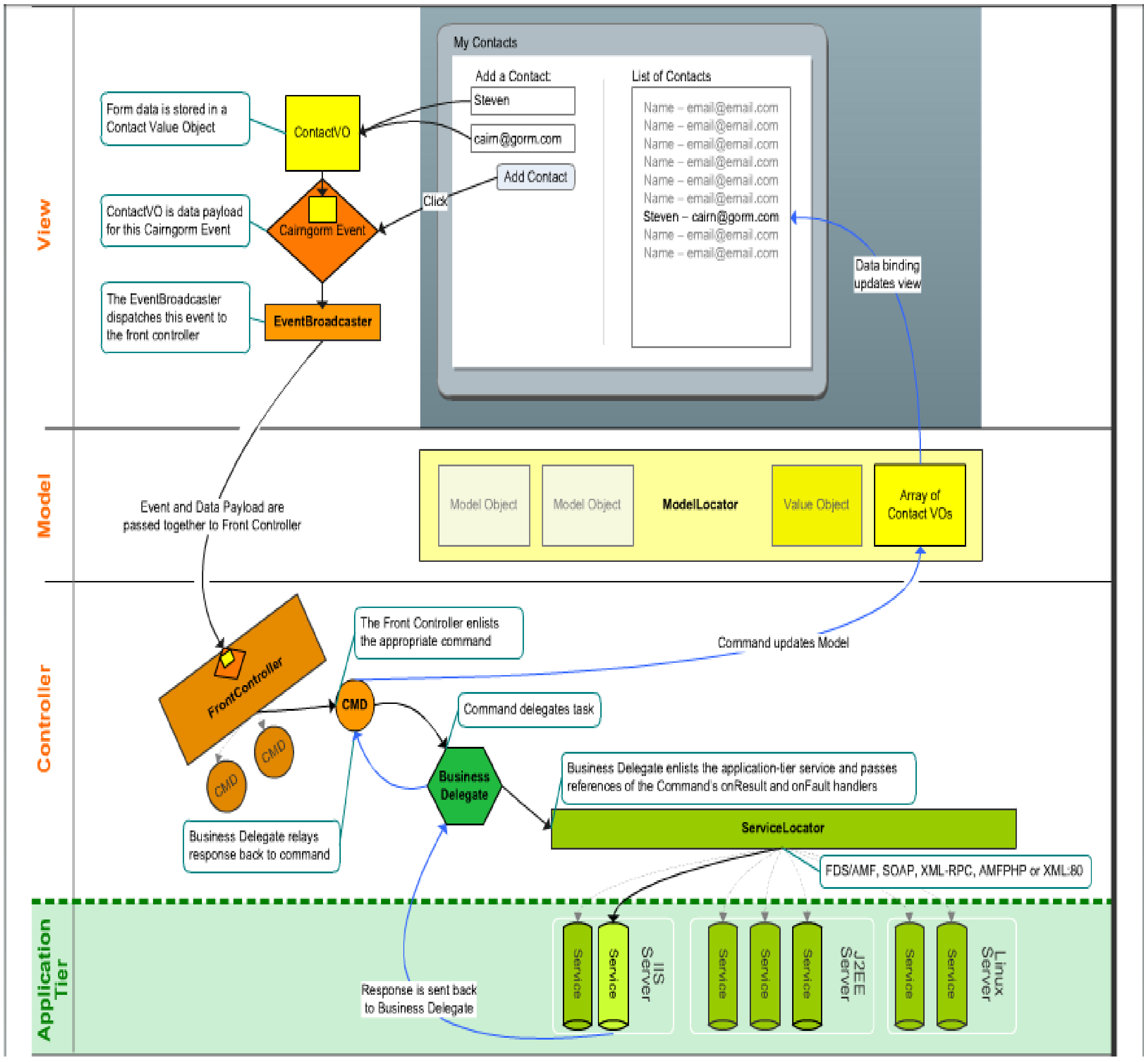

View
Model
Controller
Application Tier
My Contacts
Add a Contact:
Steven
cairn@gorm.com
Add Contact
List of Contacts
Name – email@email.com
Steven – cairn@gorm.com
Form data is stored in a Contact Value Object
ContactVO
ContactVO is data payload for this Cairngorm Event
Cairngorm Event
Click
The EventBroadcaster dispatches this event to the front controller
EventBroadcaster
Data binding updates view
Event and Data Payload are passed together to Front Controller
Model Object
Model Object
ModelLocator
Value Object
Array of Contact VOs
FrontController
The Front Controller enlists the appropriate command
Command updates Model
CMD
Command delegates task
Business Delegate
Business Delegate enlists the application-tier service and passes references of the Command's onResult and onFault handlers
ServiceLocator
FDS/AMF, SOAP, XML-RPC, AMFPHP or XML:80
Business Delegate relays response back to command
Response is sent back to Business Delegate
Service
IIS Server
J2EE Server
Linux Server

Figure 2.11: Cairngorm Micro Architecture [21]

Swiz Framework is for Adobe Flex to simplify the RIA development [17]. Swiz offer Inversion of Control, which is an object oriented programming practice where objects are bounded automatically by framework at run time rather than compile time, this will reduce the complexity of development in an interactive application. Swiz also offers event handling, simple life cycle for asynchronous remote calls and doesn't follow folder structure. It is easy to understand and develop. Swiz provides two main tags: Autowire and Mediate. While Autowire tags are used to represent the dependency of an object, Mediate tags are responsible for handling the events, anywhere in the application. This helps the developer to concentrate entirely on the functionality.

2.5.6 Communication using Zend-AMF

Flex has various ways of communicating with the server like Remote Object calls, HTTPService calls, and WebService calls. A simple GET and POST method which require retrieving the information using HTTPService call or XML can be used for transfer of data using WebService. However for large applications the communication of data structures will be complex and the serialization of these structure using WebServices and HTTPService call will be complicated. Remote provides an easy way of serialization using Action Message Format (AMF).

AMF is a binary format introduced by Macromedia. It is used for communication between client and server and also used to serialize the objects. As AMF is in binary format, it uses many optimization techniques to reduce it size. AMF was design to serialize and deserialize quickly using low memory and less CPU computation. AMF in ActionScript-2 is called AMF0 and for later versions it is called AMF3. The major advantage of AMF3 is that the object is compressed in zlib for faster transfer.

Zend is an open source application framework based on PHP. It consists of package called Zend AMF [18] and is used for communication between Flex application and PHP.

Chapter 3

Track Database Representation

3.1 Cell Tracking

Cell is the fundamental and basic structural unit of all living organisms. Cell proliferation and migration are common behaviors of cells required for the development and maintenance of any living organism. Understanding their dynamic behavior helps a biologist to predict physiological process in health, diseases and drug reactions. The behavior of a cell is therefore captured in a microscopic image at regular time intervals and these image sequences are used by the researchers to study spatio-temporal behavior of cells, their migration, and their ancestral information. This involves tracking of a cell in the image sequence [19].

Tracking basically requires connecting the segmented cells from Frame To frame. Track information helps in drawing a trajectory of an object in the image sequence and gives more accurate analysis for the researcher to study the behavior. The most common behaviors of these cells observed during image and video analysis are

a. Cell migration: Every cell in a living organism will migrate from one place to another, trajectories of this migration will help the researchers to understand the direction, speed and other important characteristics

b. Cell proliferation: This is the increase in number of cell population where a cell grows and divides to reproduce by binary fission [20]. We refer to this scenario as object splitting. Object split in a HeLa dataset and is shown in the Figure 3.1

c. Cells entering and leaving the field of view: Since the objects migrate from place to place, new objects may add to the frame and also existing objects may leave the frame.

d. Visual overlap: While cells do not merge, they may appear to merge due to segmentation problems or projection from 3D to 2D (i.e. cell sliding under another cell). We refer to this scenario as objects merge.

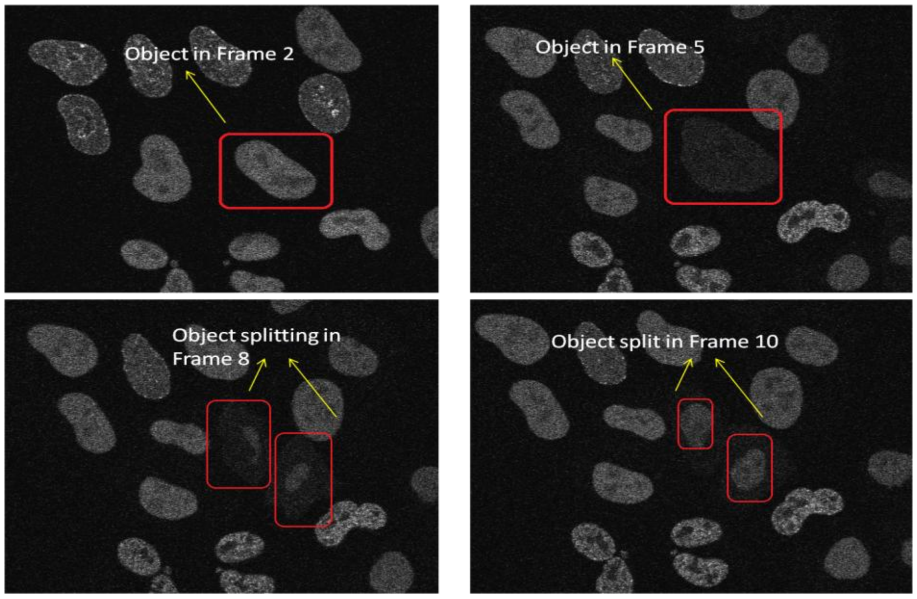


Figure 3.1 Object split in a frame sequence

However the object tracking is not limited to cells, and can be used in other biological dataset/image sequence such as microorganisms, tissues, etc. and non-biological datasets, e.g. Vehicle tracking in surveillance imaging.

We use parent and child relationships to link different instances of the same object in time and also to link parent cells to its daughter cells in case of cell division. Consequently this linkage encodes both trajectory and lineage information of an object. The parent and child association is represented in a lineage tree data structure where each object resembles a node in a tree and edges are labeled in lineage data representation. For example in the Figure 3.2, label of the edge from root node 1 to its child is represented as 1. When there is a split in the path, label is appended with an alphabet like ‘1a’, ‘1b’, ‘1aa’, ‘1ab’, etc.

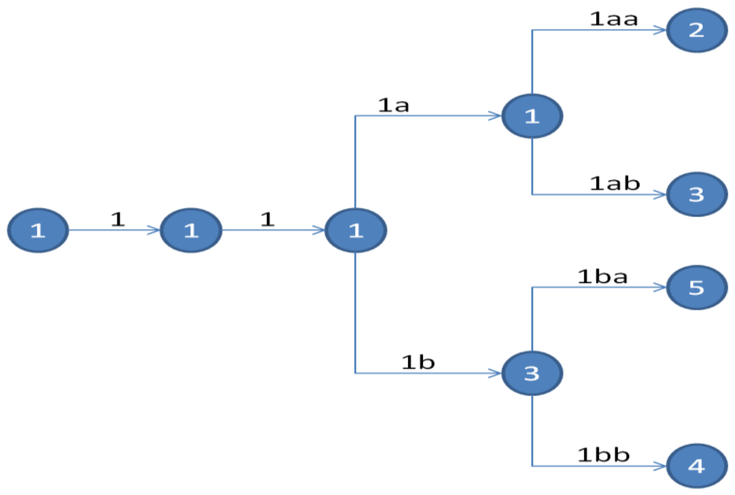


Figure 3.2 Lineage tree representation of an object split in a frameset

In the Figure 3.3, Object 5 is a merge of Object 2 and Object 3. The identity of Object 5 path is represented as (1a -1b), which contains both parent’s path information.

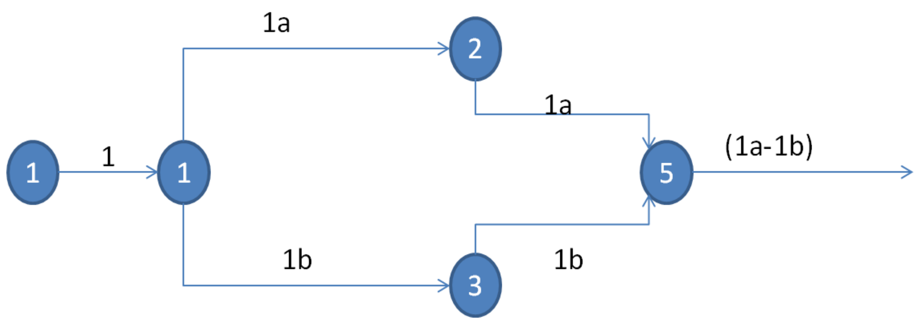


Figure 3.3 Lineage tree representation of an object merge in a frameset

The tracking information is generated by algorithms [5] and Firefly adds this information to the object attributes.

3.2 Database

Figure 3.4 represents ERD of FireFly tables associated with track information. FireFly uses seven tables such as Attribute, MarkedObject, PointObject, LineObject, BoxObject, FreeFormPoint and FreeFormPoly tables for storing track information and for drawing trajectory. MarkedObject is significant table in FireFly which consists of markedObjectID as a primary key, classification attributes and other object related information. Basically, trajectory is drawn using the centroid of an object which is calculated from the co-ordinates of an object stored in the tables' specific to an object type (PointObject, LineObject, BoxObject, FreeFormPoint and FreeFormPoly). Since, attributes of different dataset may vary; Attribute table uses key-value format to store the attributes values. This will helps FireFly to be general and allow storing multiple attribute formats. Typically, for every object, FireFly stores 4 attributes for track information: Parent, Childrens, Lbl and SegID as keys and their associated values. The database attribute names such as Parent, Childrens, Lbl and SegID represents Object attribute names- ParentID, Childrens, RootLabel and Lineage/SegmentID in the Firefly Attribute Panel. Since the user interface of Firefly is improvised with each of its versions, there is a need to change the database attribute names in the future to make it consistent with the user interface names. FireFly uses unique auto increment number for every object in the database called markedObjectID. markedObjectID is a primary key in MarkedObject table and is a foreign key in other tables to relate object with its track and segment information.

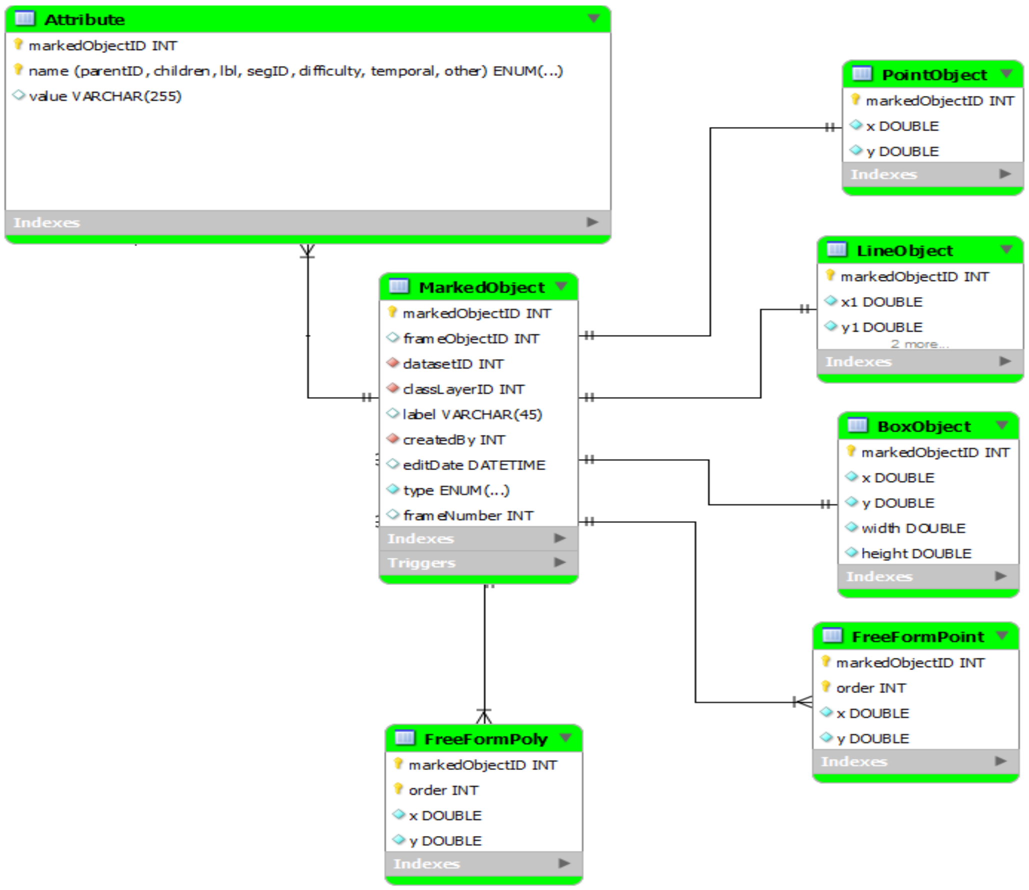


Figure 3.4 FireFly tracking Database ERD

3.3 Object Attributes

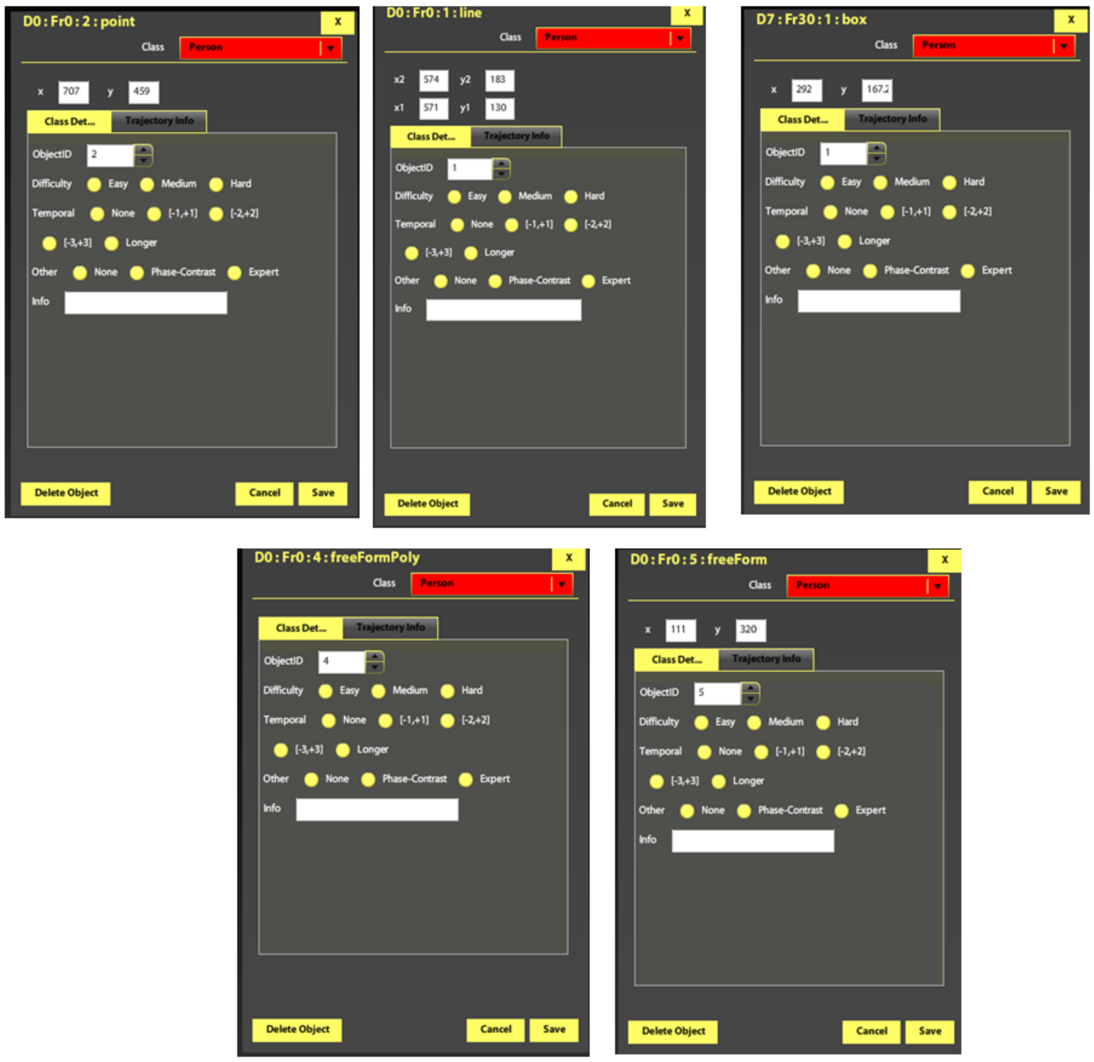
From the previous chapter we have seen the various types of datasets and types of objects such as point, line, box, polygon, etc required to represent the dataset. These objects consist of various attributes for object classification and tracking purposes as shown in the Figure 3.5. The classification attributes are

a. Class Label: Class label represents the class to which an object belongs to. This is the common attribute for all types of objects
b. Co-ordinates: This is the x, y co-ordinates on the image where the object is located. This attribute varies with the type of object as listed in table 3.1

Object Type	**Co-ordinates**
Point	X, Y
Box	X, Y, Width, height
Line	X1, Y1, X2, Y2
Polyline	CENTROID X, Y
Polygon	CENTROID X, Y

Table 3.1 Co-ordinates of different types of objects

c. Difficulty: This attribute conveys the difficulty that an expert experienced to classify the object
d. Temporal: This attribute conveys the number of frames that are required to compare to classify the objects
e. Other: This provides and other information that may be used to make a decision on classification



D0 : Fr0 : 2 : point — Class Person; x 707, y 459; ObjectID 2

D0 : Fr0 : 1 : line — Class Person; x2 574, y2 183; x1 571, y1 130; ObjectID 1

D7 : Fr30 : 1 : box — Class Person; x 292, y 167.2; ObjectID 1

D0 : Fr0 : 4 : freeFormPoly — Class Person; ObjectID 4

D0 : Fr0 : 5 : freeForm — Class Person; x 111, y 320; ObjectID 5

Figure 3.5: Attribute panel for different objects including point, box, line, polyline and polygon

FireFly uses five attributes ObjectID, ParentID, ChildrenID, Root Label, Lineage/SegmentID and TrackletID (yet to implement) to store the track information and these are explained in the following sections.

3.3.1 ObjectID

Every object in a frame is assigned with a unique identification number called ObjectID. ObjectID cannot be zero or less than zero. FireFly, by default assigns the ObjectID to the maximum ObjectID value in that frame added one (Maximum+1) for a new object created. Figure 3.6 represents the visualization of an image and associated objects in FireFly. Each object is visualized with a color and label. Color represents the class the object belongs to and label consists of class name, x and y coordinates and a unique ObjectID that is used to identify an object quickly in a frame.

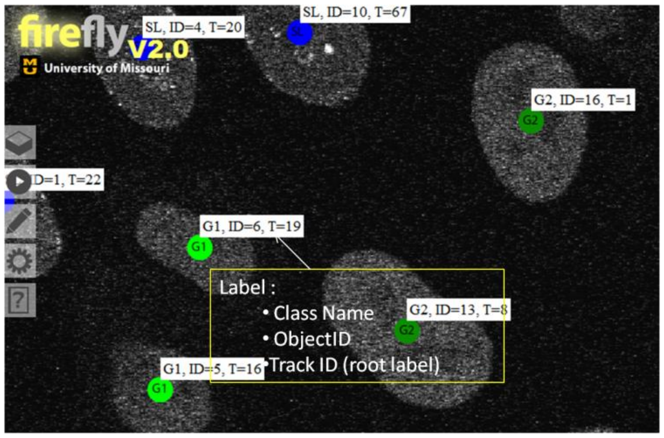


Figure 3.6 Point objects in a frame

A single object instance can hold different ObjectID numbers in different frames. For example, in Figure 3.7, Object '1' in Frame 'T', Object '1' in Frame 'T+1', Object '3' in Frame 'T+2', Object '4' in Frame 'T+3' and Object '4' in Frame 'T+4' are instances of the same object in different frames.

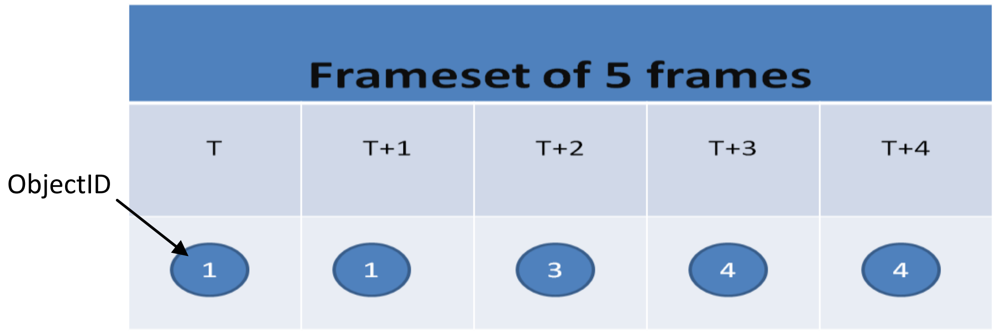


Figure 3.7 ObjectID of an object in different frames

3.3.2 ParentID

Parent for an object is its instance in the previous frame and ParentID of an object represents the ObjectID of its parent. For an object which does not have any instance in the previous frame is considered as a root object (similar to the root node in a tree data structure) and its ParentID is assigned to '-1'. Figure 3.8 represents parent association for the objects in the frames Frame T to Frame T+4.

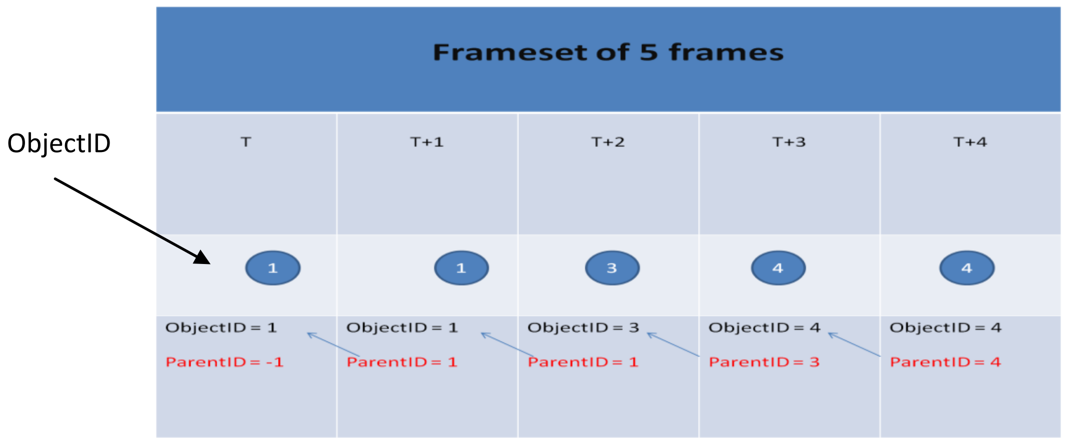


Figure 3.8 ObjectID and ParentID of an object in a frame sequence

In some scenarios, object may have more than one parent (i.e. object has more than one instance in the previous frame) which is referred as an object merge. ParentID of an object in this case consists of ObjectID of its parents separated by a comma.

In Figure 3.9 Object 1 in Frame T+1 is an example of object split scenario, where it consists of two parents: Object 1 and Object 2 in Frame T. Therefore, the ParentID of object 1 in Frame T+1 consists of ObjectID 1 and ObjectID 2 with a comma separated.

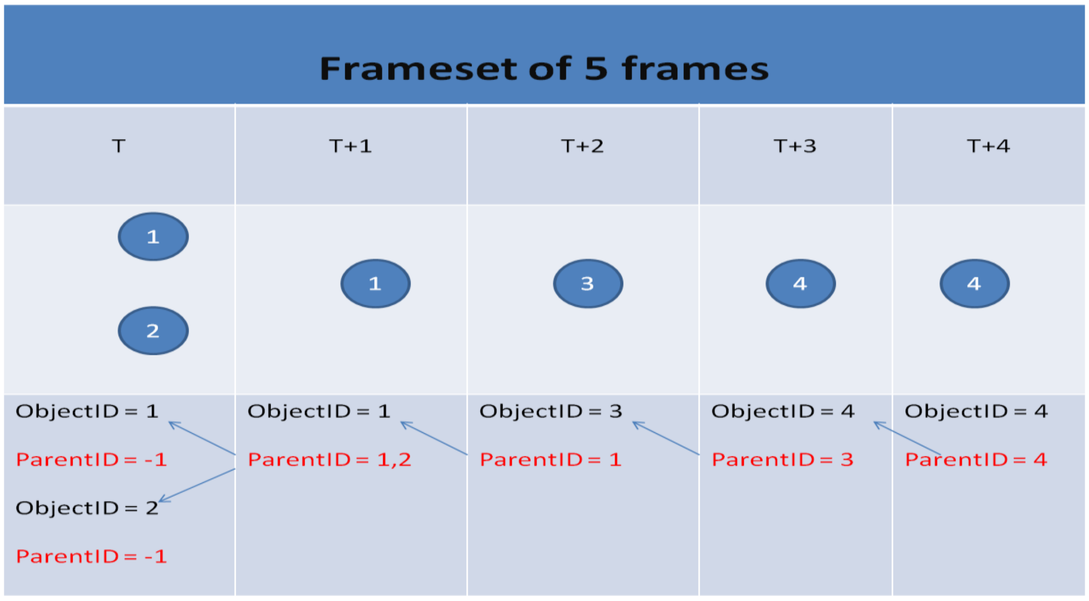


Figure 3.9 ObjectID and ParentID of an object in a frame sequence in case of merge

3.3.3 ChildrenID

A child for an object is its instance in the subsequent frame and ChildrenID of an object represents the ObjectID of its child. An object which does not have any instance in the subsequent frame is considered as a leaf object (similar to the leaf node in a tree data structure) and its ChildrenID is assigned to '-1'. Figure 3.10 represents children association for the objects in the frames Frame T to Frame T+4.

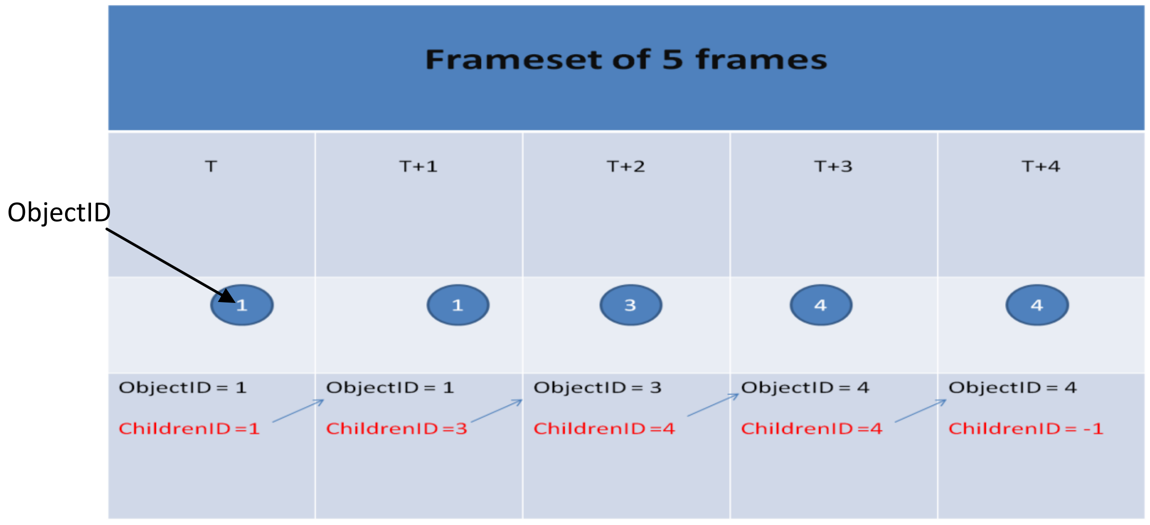


Figure 3.10 ObjectID and ChildrenID of an object in a frame sequence

In some scenarios, object may have more than one child (i.e. object has more than one instance in the subsequent frame) which is referred as an object split. ChildrenID of an object in this case consists of ObjectID of its children separated by a comma.

In Figure 3.11 Object 4 in Frame T+3 is an example of object split scenario, where it consists of two children: Object 4 and Object 6 in Frame T+4. Therefore, the ChildrenID of object 4 in Frame T+3 consists of ObjectID 4 and ObjectID 6 with a comma separated.

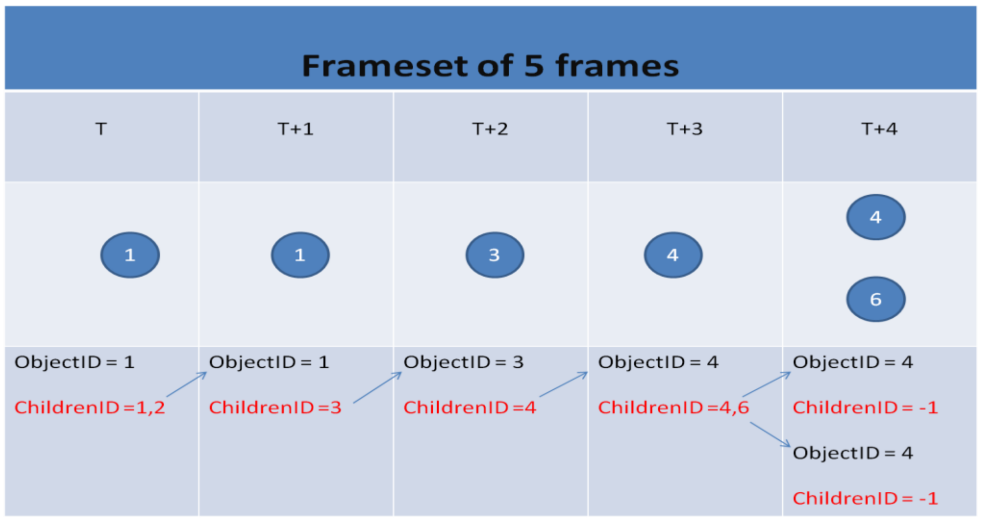


Figure 3.11 ObjectID and ChildrenID of an object in a frame sequence

3.3.4 Root Label

Tracking of an object involves connecting the instances of an object from frame to frame. This is accomplished by associating parent and children of an object as discussed in the sections 3.2.2 and 3.2.3.

Figure 3.12 shows the visualization of trajectories of various objects in a dataset, drawn using their corresponding track information. In a critical dataset, each frame is expected to have numerous objects and differentiating each object's track is difficult. Therefore, every individual track is assigned with a unique identification number called Root Label.

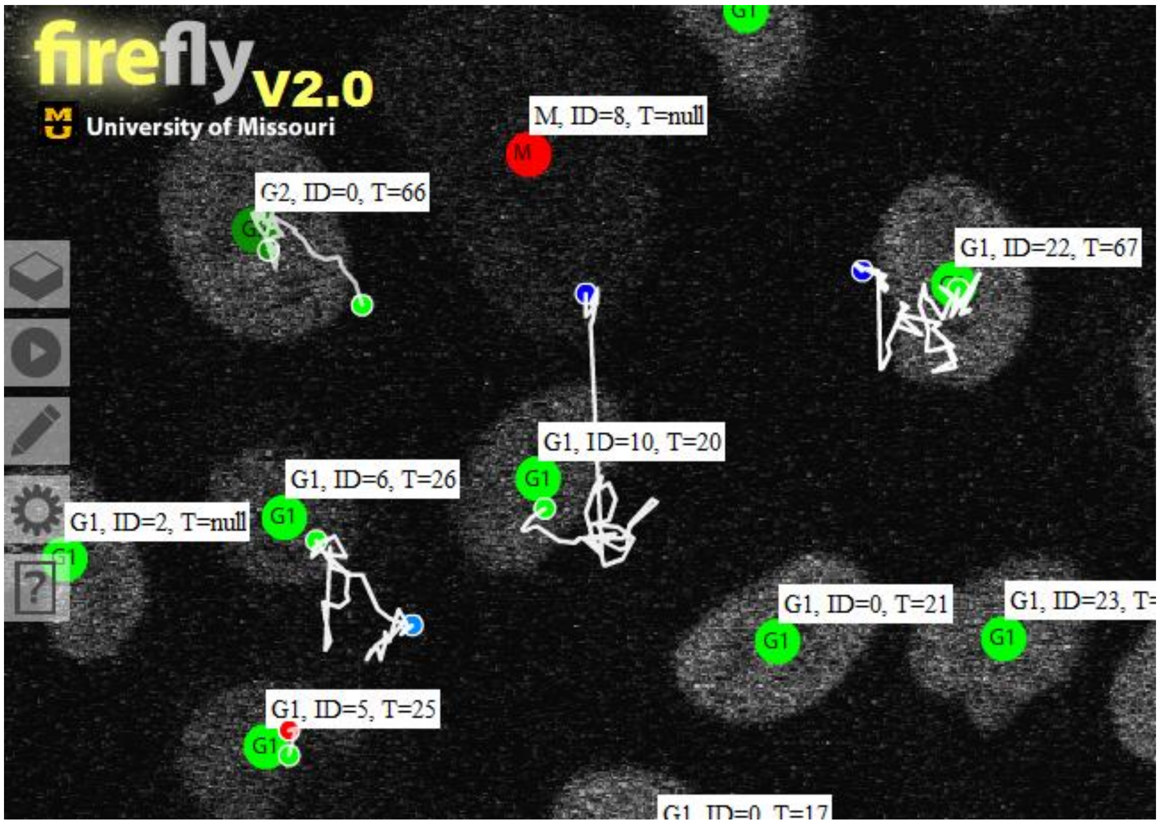
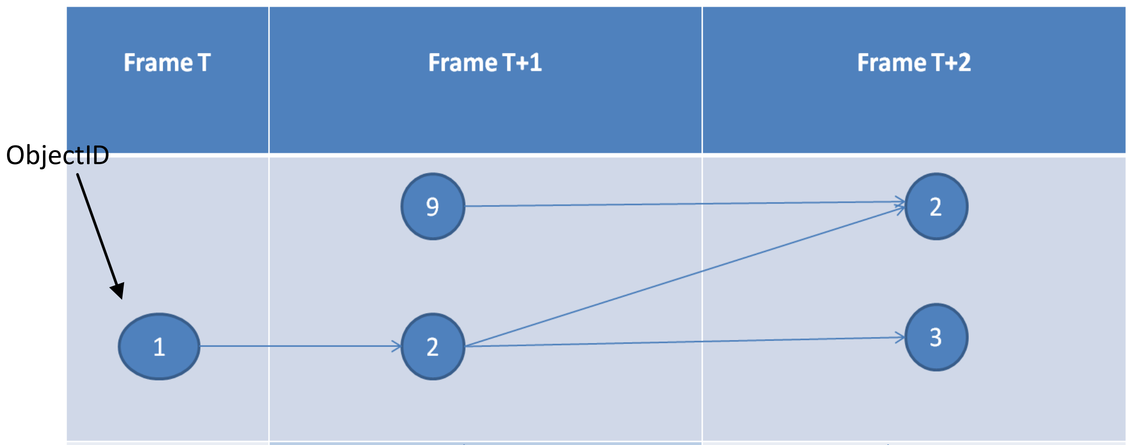


Figure 3.12 Trajectories of objects in a frame

Root Label is initiated for every root object (i.e. ParentID is -1) and propagated through its descendants. Root Label initiation must be a unique number; therefore each root object is initiated with the maximum Root Label value in a dataset added one (Maximum +1).

In an object merge scenario (i.e. object has more than one instance in the previous frame), object contains more than one parent and each parent object has a unique Root Label. Root Label of one of the parent object must be chosen to propagate to maintain unique and consistent track information. Therefore, minimum of parents Root Label values is chosen to be propagated through its descendants.

In Figure 3.13, root object ‘Object 1’ in Frame T is initiated with the Root Label 12 and it is propagated through all its descendants. Object 2 in Frame T+2 is an example of object merge scenario, which has two parents: Object 9 and Object 2 in Frame T+1. Minimum of the two Root Labels of Object 9 and Object 2 (i.e. minimum of 12, 14 is 12) is chosen as its Root Label.



ObjectID = 1	ObjectID = 2	ObjectID = 9	ObjectID = 3	ObjectID =2
ParentID = -1	ParentID = 1	ParentID = -1	ParentID = 2	ParentID = 2,9
ChildrenID = 2	ChildrenID = 2,3	ChildrenID = 2	ChildrenID = -1	ChildrenID = -1
Root Label = 12	Root Label = 12	Root Label = 14	Root Label = 12	Root Label = Minimum(12,14)=12

Figure 3.13 Root Labels of objects in a frame sequence

3.3.5 Lineage (or SegmentID)

Lineage attribute is used for drawing the trajectory of an object. It contains tree edge labels as shown in the figures 3.2 and 3.3. Lineage for the root object will have the same Lineage as its Root Label. For example; in the Figure 3.14, Object 1 at Frame T, is a root object. Its Lineage will be same as its Root Label – ‘12’. Object 2 at Frame T+1 will split into two objects: 2 and 3 at Frame T+2.

Therefore, in Frame T+2 the Lineage for Object 3 is 12a, but for Object 2, it is a merger of two objects, thus it acquires both Lineage from its parents with a ‘-’ separator (12b-14).

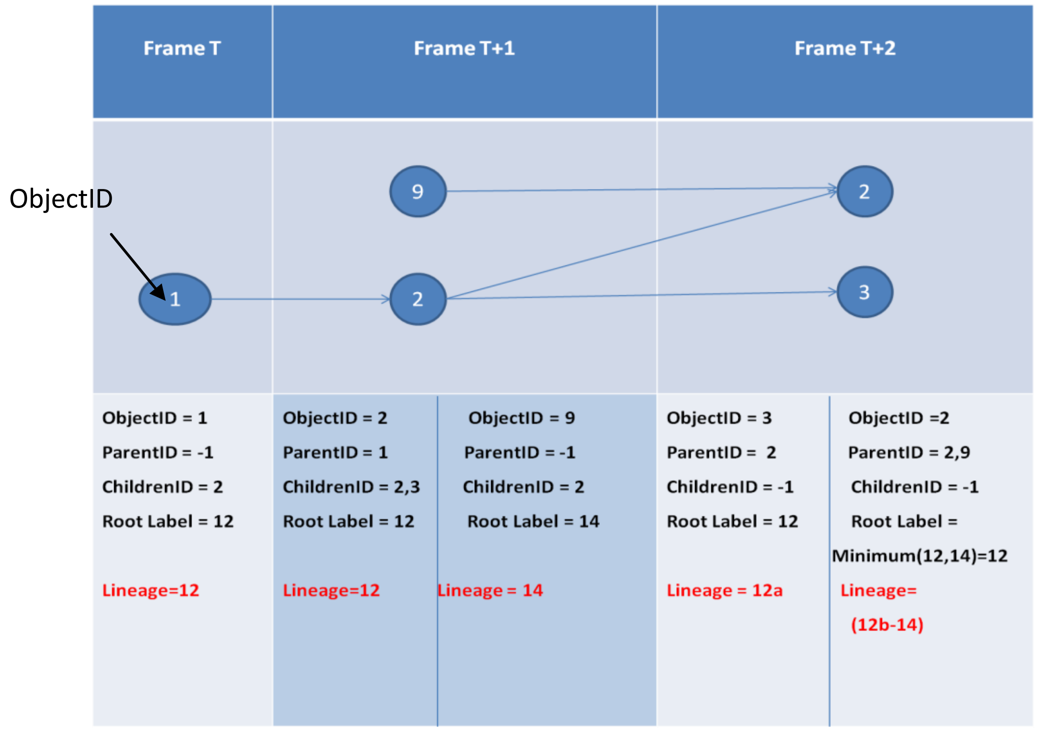
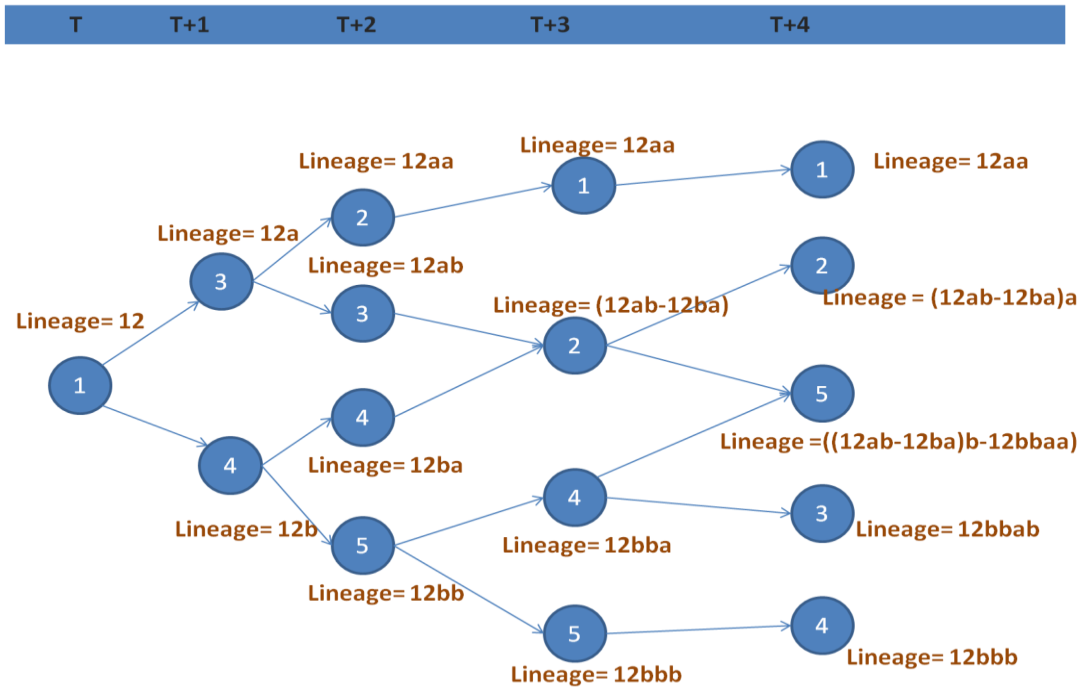


Figure 3.14 Lineage of objects in a frame sequence

Figure 3.15 depicts the Lineages of objects in a complex scenario with multiple split and merge situations. Lineage attribute is implemented using SegmentID name in the previous versions of Firefly, however, this attribute is changed to Lineage as most suitable naming convention to represent complete history of its parents.



Frame 3.15 Lineages of objects in a complex scenario

Figure 3.16 depicts the object track information and lists all track attributes of an object in a frameset.

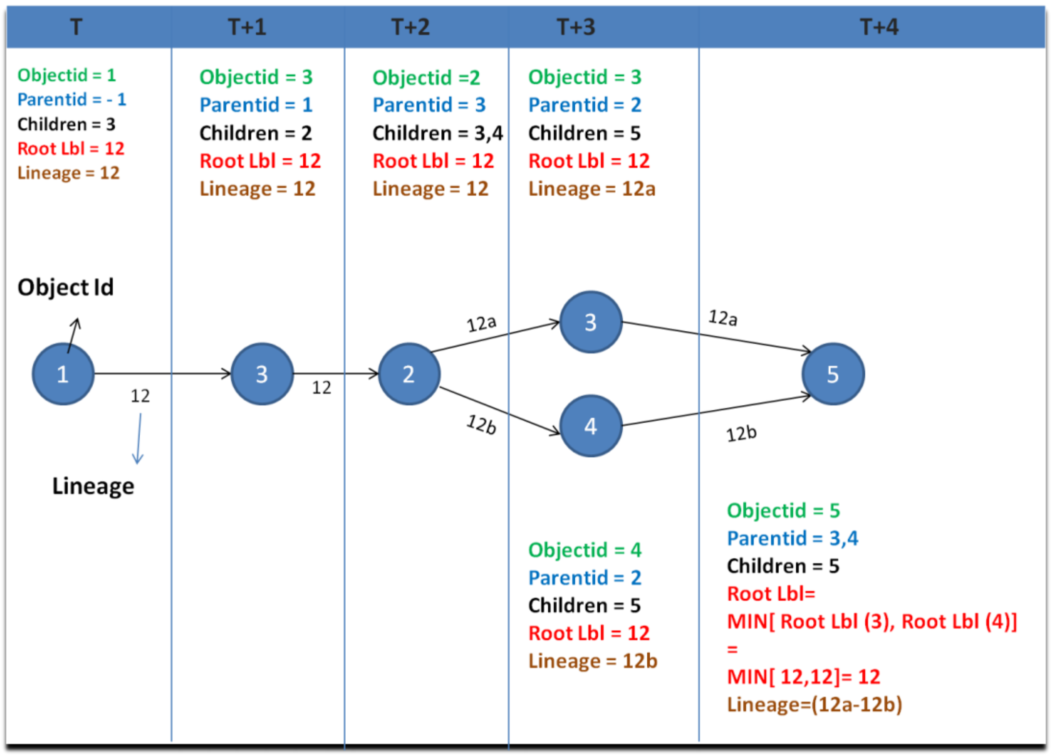


Figure 3.16 Root Labels of an object in a frame sequence

3.3.6 TrackletID

In order to efficiently access, display and modify trajectory information it may be useful to maintain a separate table of TrackletID information. A tracklet is a sequence of objects with one-to-one relationships without any splits or merges. That is the in- and out-degree of each node in a tracklet is one (one parent and one child) or zero (source or sink node). If there is any splits or merges then one TrackletID ends and a new TrackletId is started (ie each TrackletID can be associated with a single line for drawing without branching or forks). Using TrackletIDs enables efficient drawing, maintaining track level information such as the total number of points in the tracklet, etc. Figure 3.17 represents trackletIDs of an object in a frame sequence. Note that TrackletID database field is not yet implemented in the FireFly database and it is shown in the Figure 3.17 for illustration.

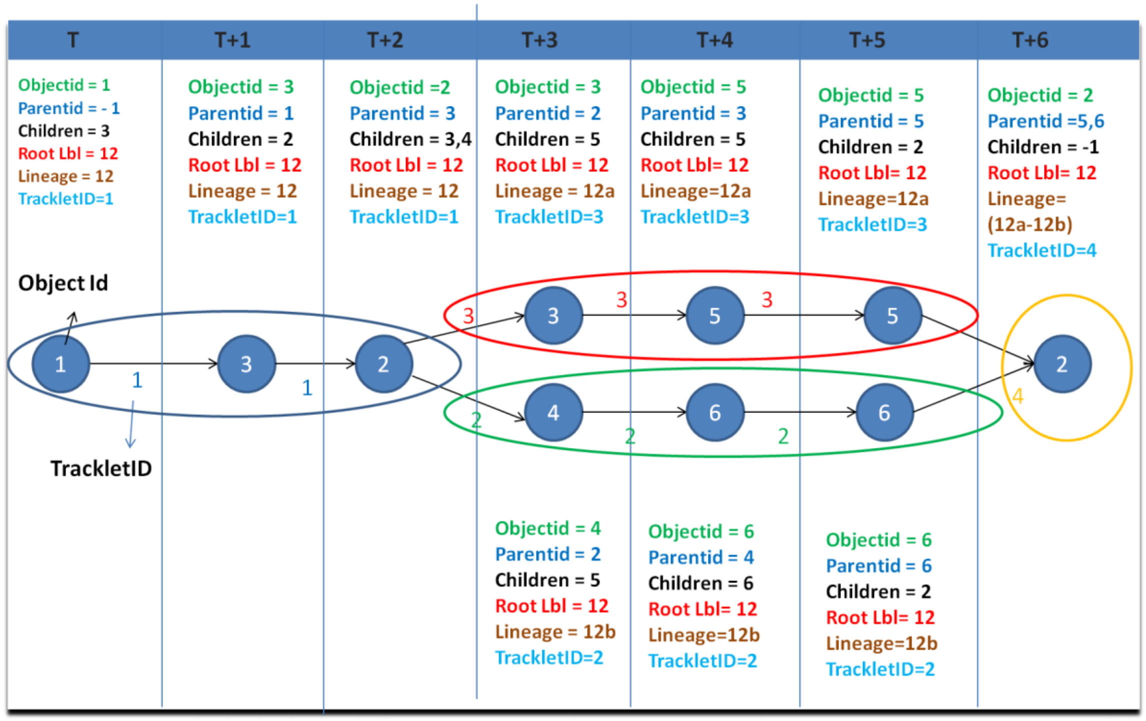


Figure 3.17 TrackletID of an object in a frame sequence. Note that the TrackletID database field is not yet functional in the FF database and is shown here for illustration

Chapter 4

User Track Editing Operations

4.1 Need for Automatic Track Label Propagation

For a critical dataset, tracking is a challenging task which requires additional manual editing operations to correct the track data generated from automated tracker/software. Track editing involves connecting objects from different frames by associating the parent, ChildrenID and label properties. Figure 4.1 shows the track information from Frame T to Frame T+4. During ground truth correction, Object 2 at Frame T+2 is found to be a spurious point and requires a 'manual delete' operation by the user to delete the object. The delete operation eventually changes track information for objects in the frames T+3 and T+4. User must edit the parent, ChildrenID and label properties for these to maintain consistent track information. Practically, this manual edit operation for track label propagation is time consuming and infeasible for large populations of objects.

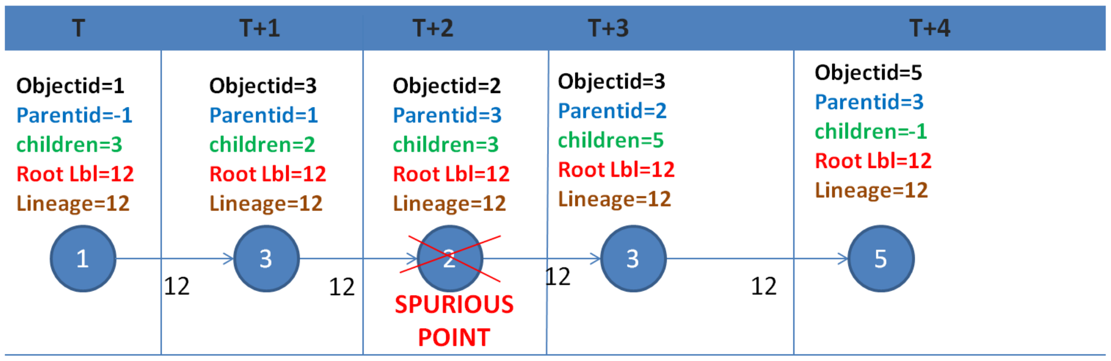


Figure 4.1 Spurious point in Frame T+1

Therefore, an automatic track updating algorithm is required when an edit operation is made on an object. This requires studying all possible scenarios of manual track edit operations to implement feasible algorithm. The possible instances, when manual correction is required are:

- Creating track information
- Adding or Extending track information
- Deleting spurious points / Splitting tracks
- Joining tracks
- False merge
- False split
- ID switches

4.2 Creating (Missing) Track Information

In this case, a frameset is missing the complete track information of an object or an object itself is missing. In both the scenarios user must provide the ParentID and ChildrenID attribute details for that object in every Frame to build the track information.

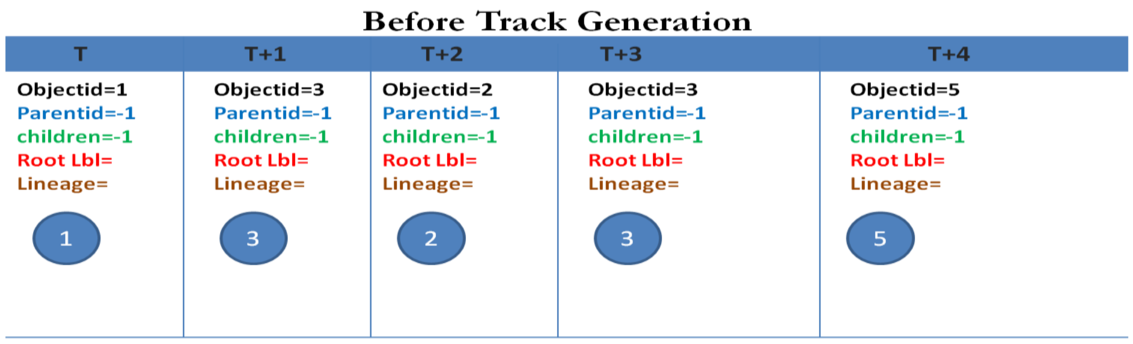


Figure 4.2 Missing track information in a frameset **before** edit operation

Figure 4.2 explains the scenario where an object is created in every frame and the track information has to be provided manually. When a new object is created, ParentID and ChildrenID are assigned to '-1' and Root Label and Lineage will be null by default. To build the track for an object, the user must manually edit the parent and children attributes. At the point of saving the track information, Root Label of new object will be the maximum Root Label in the database added one and Lineage is the same as Root label. Track information can be built by editing either ParentID or ChildrenID of an object. In Figure 4.3, the Root Label and Lineage is propagated automatically after editing the ParentID and ChildrenID information of an object. Operations on FireFly involve, editing the ParentID and ChildrenID attributes in the attribute window. Root Label of the parent object will be propagated to the current object and it descendants.

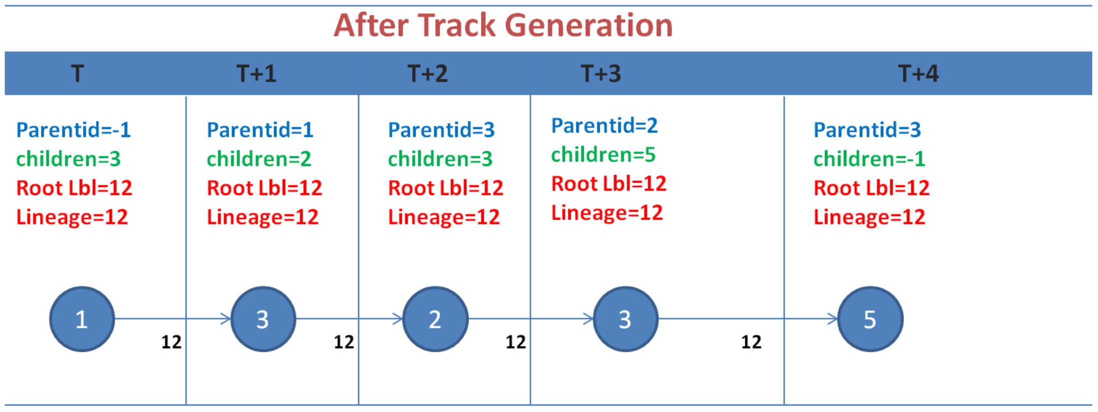
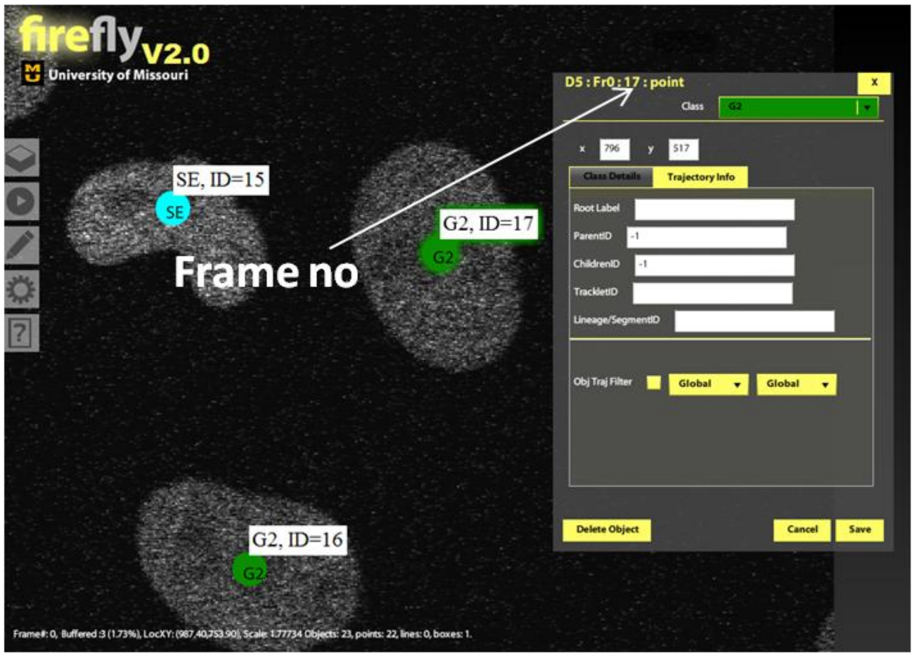


Figure 4.3 Track information **after** user edit operation in a frameset

Operation on FireFly:

FireFly uses rich internet application flex to visualize the objects and track information. It interfaces with the user via a window to manually edit the classification and tracking properties of an object. When a user double clicks on an object, an attribute window will pop up with all the attributes information of that object. Figure 4.4 shows the attribute window of Object 17 with a label on the top representing dataset number which the user is working on, frame number of the object, ObjectID and the type of graphical shape. Since object 17 in frame 0 is missing track information, its ParentID and ChildrenID are represented with -1 and Root label and Lineage are null. Trajectory is drawn for any object by selecting the Auto Trajectory or Draw Trajectory in the attribute window. Auto Trajectory draws the trajectory for the complete path of an object in a dataset. Draw Trajectory draws the trajectory between the start and end frames mentioned in the attribute window. Since there is no track information presented, trajectory is not being drawn for the object 17 from frame 0 to 172 as selected in the attribute window.



Class G2

x 796 y 517

Root Label

ParentID -1

ChildrenID -1

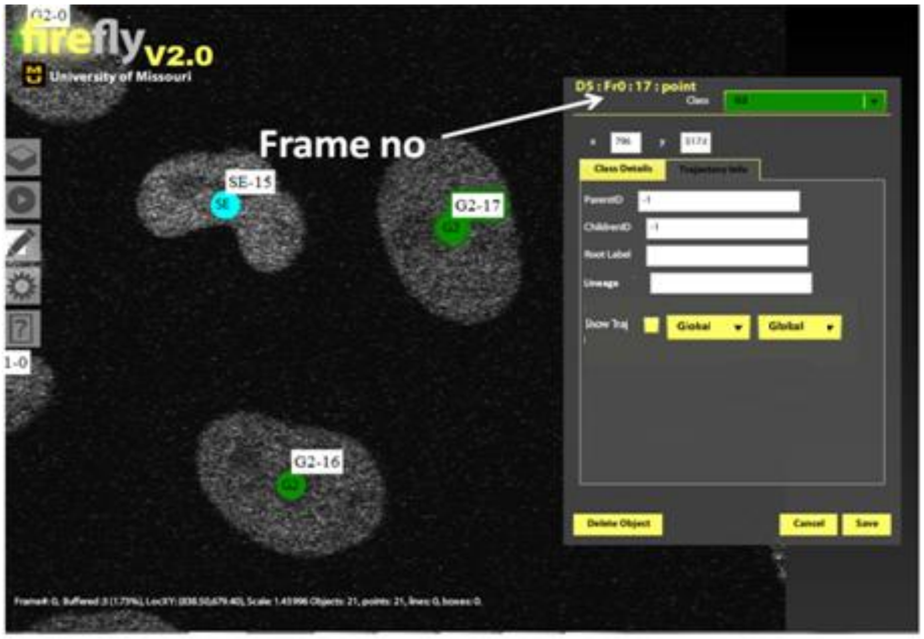
TrackletID

Lineage/SegmentID

Obj Traj Filter Global Global

Frame#: 0, Buffered :3 (1.73%), LocXY: (987.40,753.90), Scale: 1.77734 Objects: 23, points: 22, lines: 0, boxes: 1.

(a)



Class G2

x 796 y 517

ParentID -1

ChildrenID -1

Root Label

Lineage

Show Traj Global Global

Frame#: 0, Buffered :3 (1.73%), LocXY: (838.50,679.40), Scale: 1.43996 Objects: 21, points: 21, lines: 0, boxes: 0.

(b)

Figure 4.4 Trajectory of Object 17 from frame 0 to frame 172 (a) New Attribute panel (b) Old Attribute Panel. Note that other figures in this chapter may show the old version of the attribute panel.

To generate track information for any object, user has to manually link the parent and children of that object throughout the dataset; this involves manually entering the ObjectID of its parent and children in the attribute window as shown in the Figure 4.5. When user links the object 17 at frame 0 with its children, algorithm will automatically updates the ParentID and also propagates the root label and Lineage for object 17 at frame 1 as shown in the Figure 4.6.

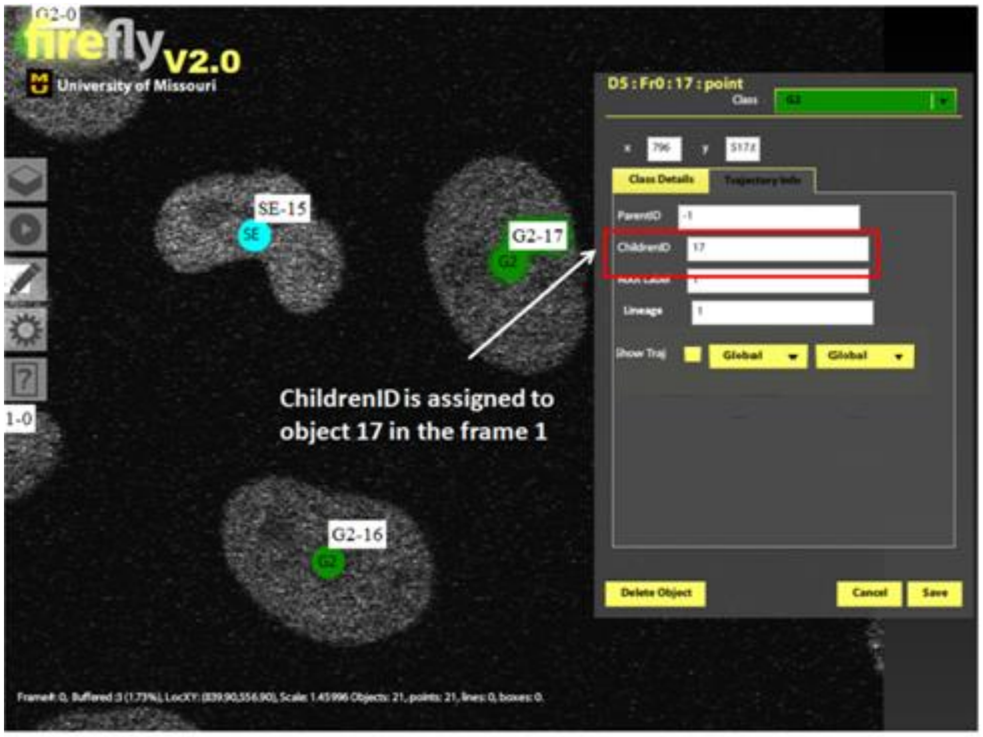


Figure 4.5 User linking Object 17 at frame 0 with new child

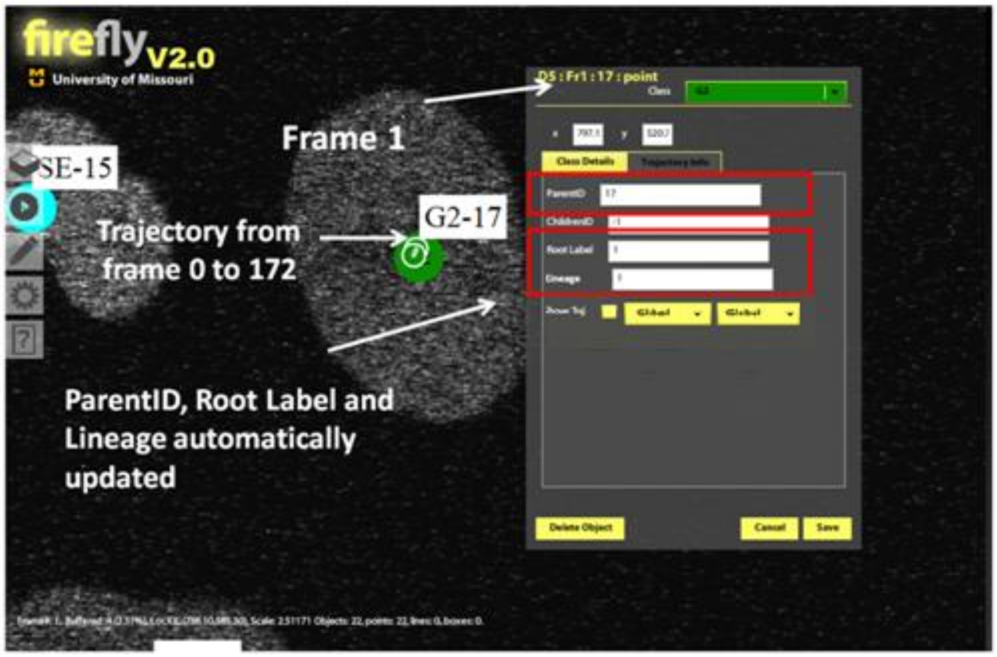
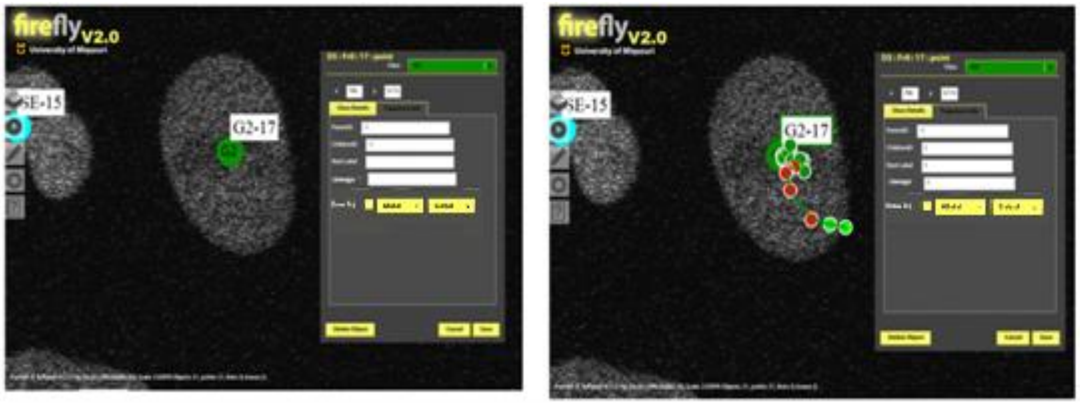


Figure 4.6 Trajectory of object 17 and automatic ParentID, Root label and Lineage update

Similarly, user must manually link the children of object 17 in every frame to automatically propagate the root label and Lineage in a dataset. Figure 4.7 shows the trajectory of object 17 before and after creating link.



(a) Before creating link operation **(b) After creating link operation**

Figure 4.7 Trajectories of Object 17 **before and after** creating link operation

4.3 Adding or Extending Track Information

In this case, track information is edited to extend the existing track. In Figure 4.8, there exists a track with a Root Label 12, but, objects in the Frame T and Frame T+4 which belong to the same track are missing in the track information. This missing object information can be added to the existing track by appending missing object information to ParentID or ChildrenID in the track. When an object is added to the ChildrenID list, Root Label and Lineage are automatically propagated to the child object and its descendants.

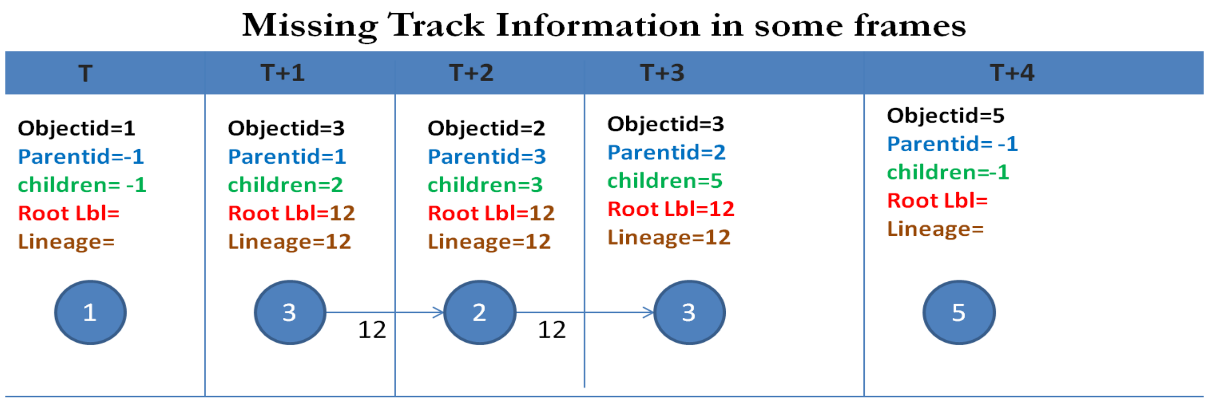


Figure 4.8 Missing track information in some frames in a frameset at the beginning and end of the track **before** track extend operation

Figure 4.9 depicts the Root Label and Lineage propagation after joining new objects to the existing track.

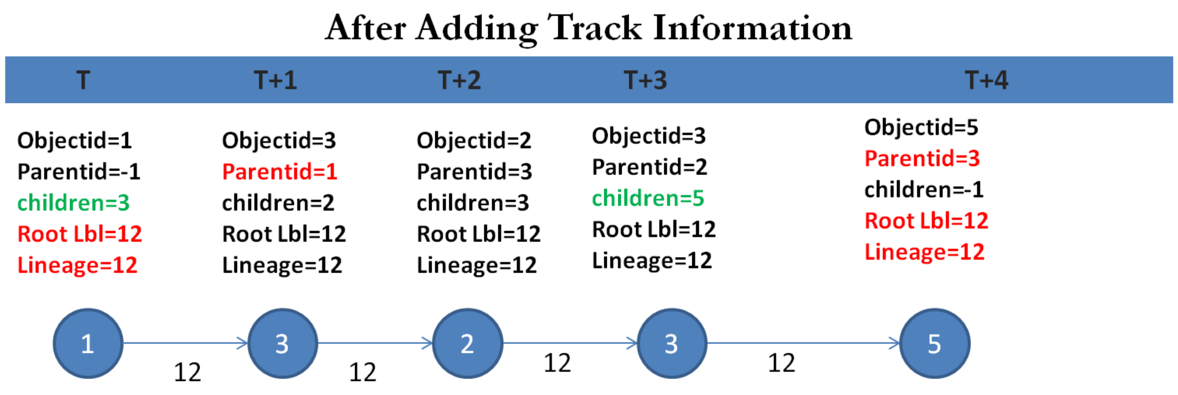


Figure 4.9 Track information **after** user edit operation in a frameset to add missing nodes

Operation on FireFly:

Figure 4.10 shows two frames 13 and 14 with trajectory visualization of object 23 at frame 13 and missing track information of object 24 at frame 14. To join the object 24 to the track requires manual edit operation of entering ObjectID 23 to the ParentID attribute of object 24 at frame 14. Algorithm automatically propagates the Root Label 1 to object 24 as shown in the figure 4.11.

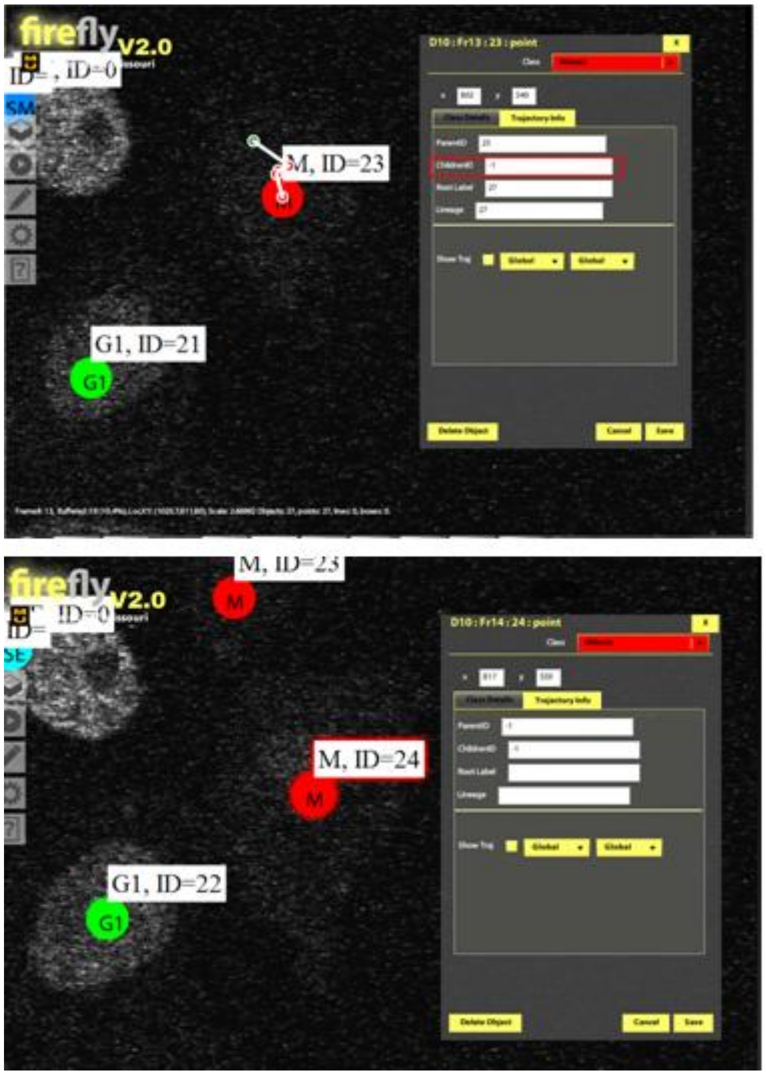


Figure 4.10 Visualization of trajectory of object 23 at Frame 13 and object 24 at Frame 14 **before** adding object 24 to the track

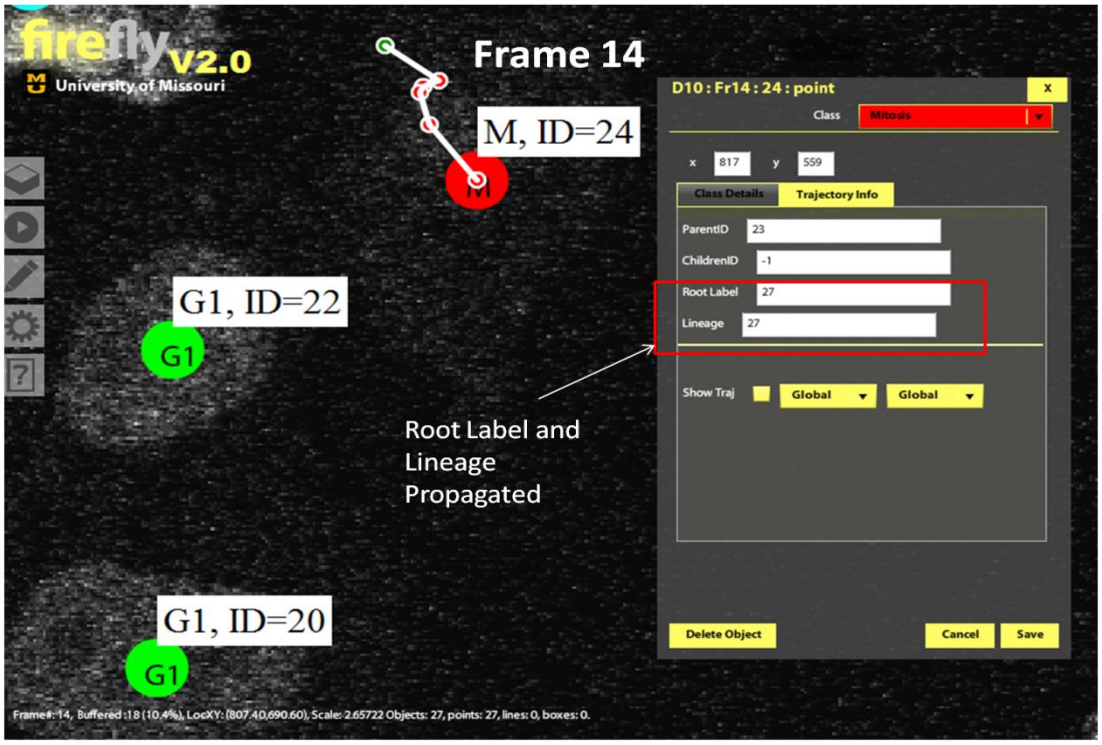
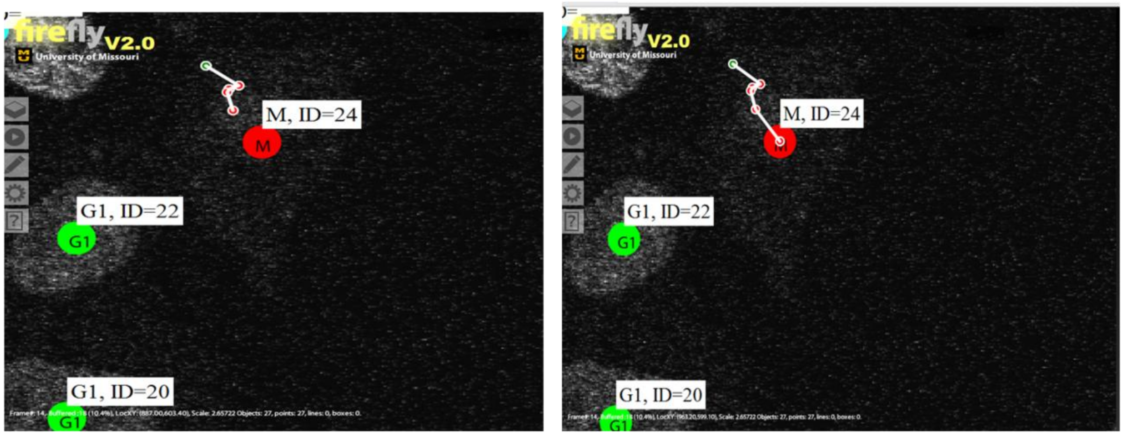


Figure 4.11 Trajectory visualization of object 24 at Frame 14 **after** adding object to the track



(a) Before extending track **(b) After extending track**

Figure 4.12 Trajectories visualization **before and after** extending track

4.4 Joining Tracks

In some instances, two or more different tracks are joined together due to the error in an algorithm or due to a missing object in one of the frames. These two tracks with different Root Labels can be joined together by a leaf object and a root object of the two tracks. In the Figure 4.13, the track with Root Label 12 and the track with Root Label 14 originally belong to the same track with Root Label 12.

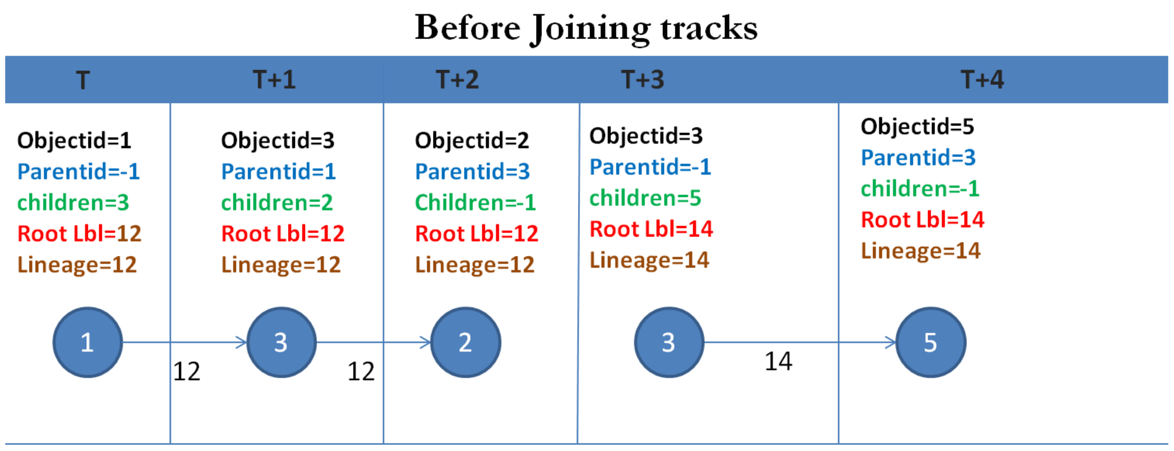


Figure 4.13 Two different tracks of the same object in a frameset before **join** operation

The two tracks are manually joined by adding Object 3 at Frame T+3 to the ChildrenID list of Object 2 at Frame T+2 as shown in Figure 4.14. When ChildrenID are edited manually, the ParentID of Object 3 is updated automatically and Root Label of Object 2 at Frame T+2 is propagated to Object 3 and its descendants.

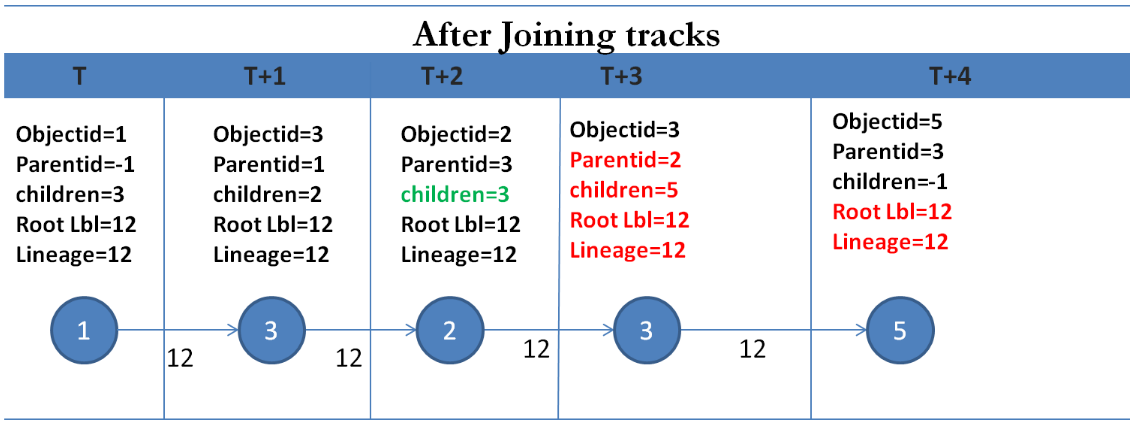


Figure 4.14 Track information after joining two tracks in a frameset **after** join operation

Operation on FireFly:

Figure 4.15 shows the visualization of two different tracks. To join these two tracks requires user to link the leaf node of track 1 and root node of track 2 as shown in Figure 4.16.

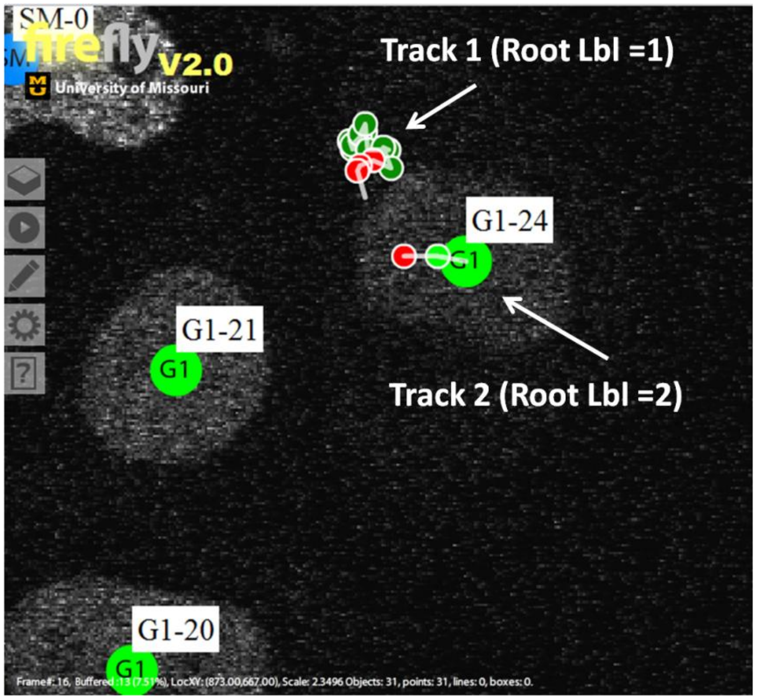

SM-0
firefly v2.0
University of Missouri
Track 1 (Root Lbl =1)
G1-24
G1
G1-21
G1
Track 2 (Root Lbl =2)
G1-20
G1
Frame#: 16, Buffered :13 (7.51%), LocXY: (873.00,667.00), Scale: 2.3496 Objects: 31, points: 31, lines: 0, boxes: 0.

Figure 4.15 Trajectory visualization of two different tracks

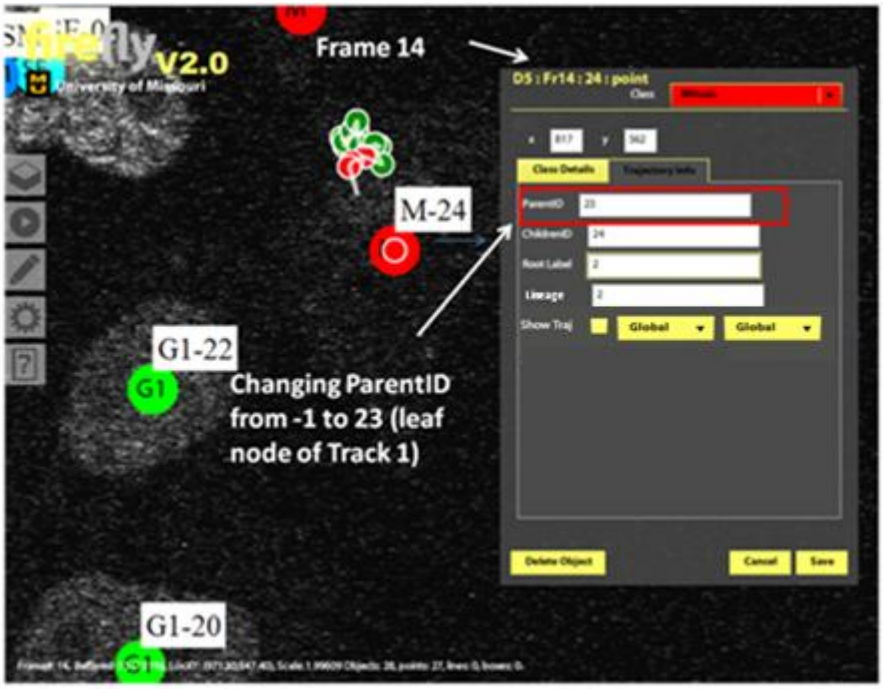


Figure 4.16 Joining object 24 at frame 14 and track 1

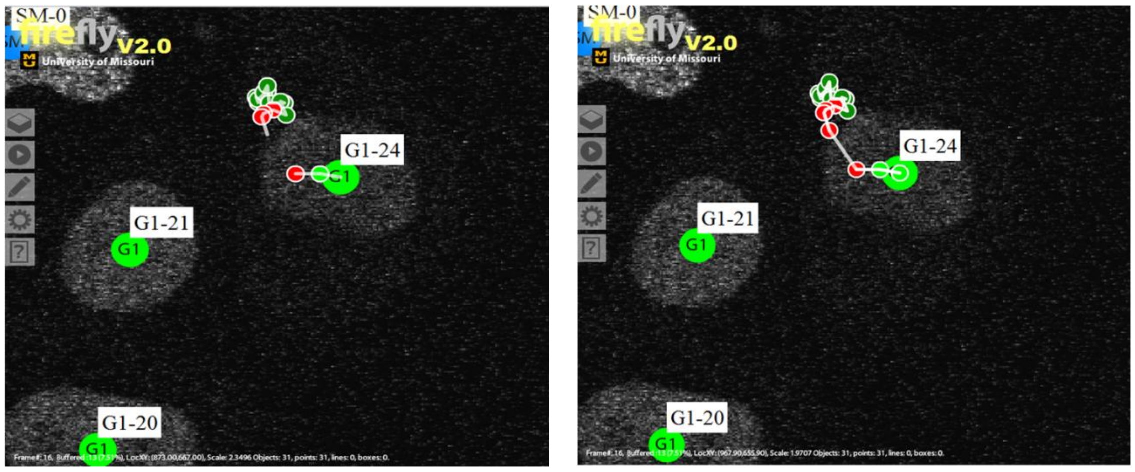


Figure 4.17 Trajectory visualization **before and after** joining track 1 and track 2

4.5 Deleting Spurious Points/ Splitting Tracks

In the process of track editing, spurious objects are identified and deleted. When an object is deleted the track is updated automatically by splitting the track into two tracks. In Figure 4.18, Object 2 at Frame T+2, is identified as spurious and track information needs to be updated.

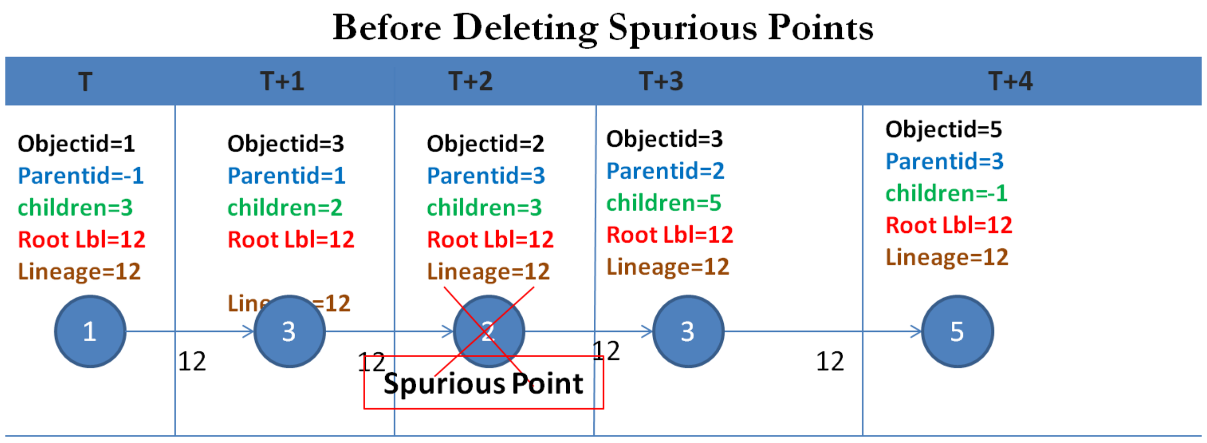


Figure 4.18 Track information **before** deleting spurious objects in a frameset

Figure 4.19, shows the track information update after deleting Object 2. The ParentID and ChildrenID list of parents and ChildrenID objects are updated automatically. Object 3 at Frame T+3 ParentID is updated to -1 and becomes the root object for its descendants. Root Label is assigned to the maximum Root Label added one.

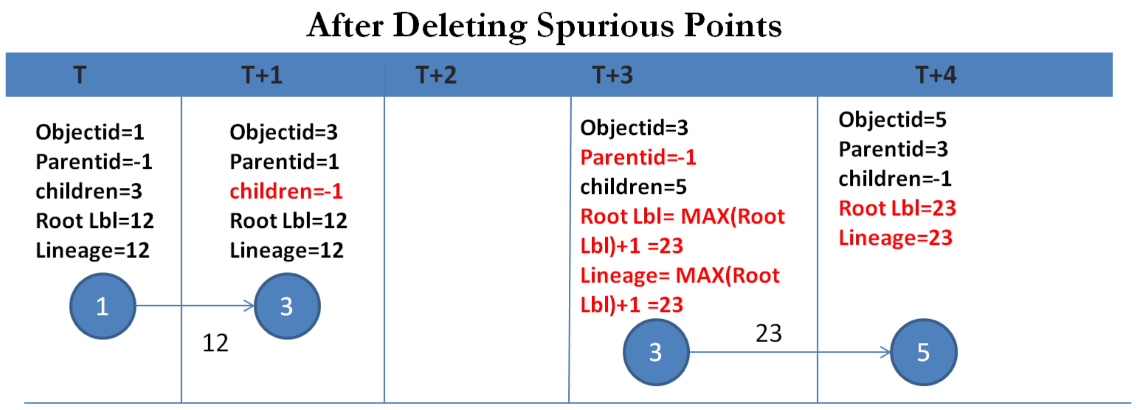
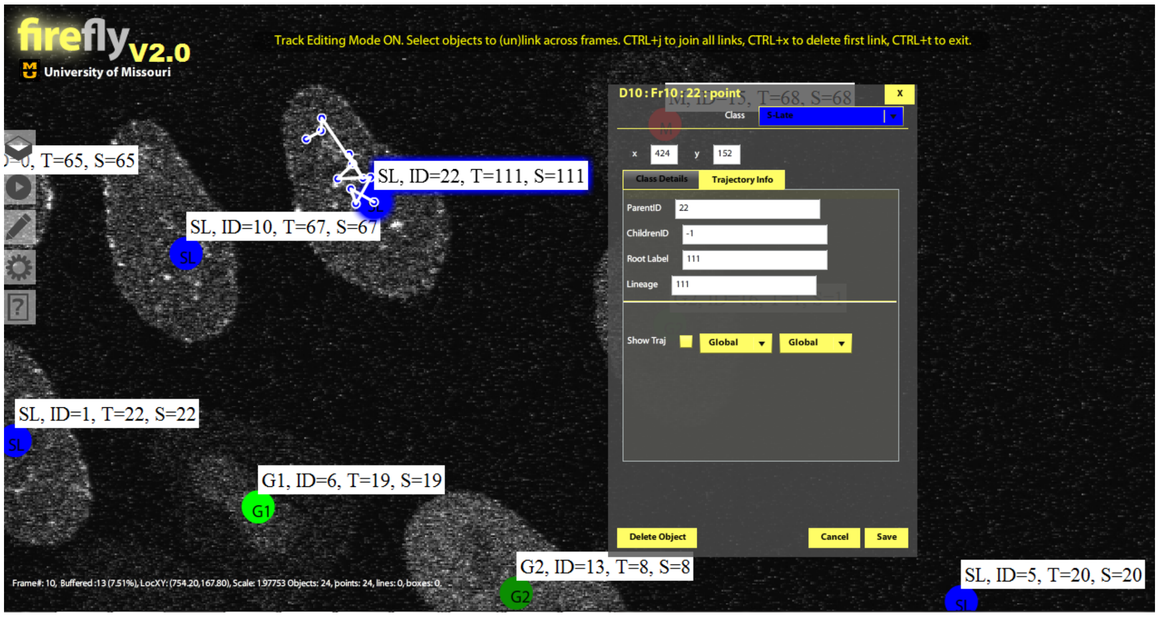


Figure 4.19 Track information **after** deleting spurious objects in a frameset

Operation on FireFly:

Figure 4.20 shows the attribute window of Object 1 with a label on the top representing dataset number which the user is working on, frame number of the object, ObjectID and the type of graphical shape. Figure 4.20 also, visualizes the trajectory of track label 1 from frame 0 to 8 using Auto Trajectory or Draw Trajectory selection.


Track Editing Mode ON. Select objects to (un)link across frames. CTRL+j to join all links, CTRL+x to delete first link, CTRL+t to exit.

D10 : Fr10 : 22 : point

Class: S-Late

x: 424 y: 152

ParentID: 22

ChildrenID: -1

Root Label: 111

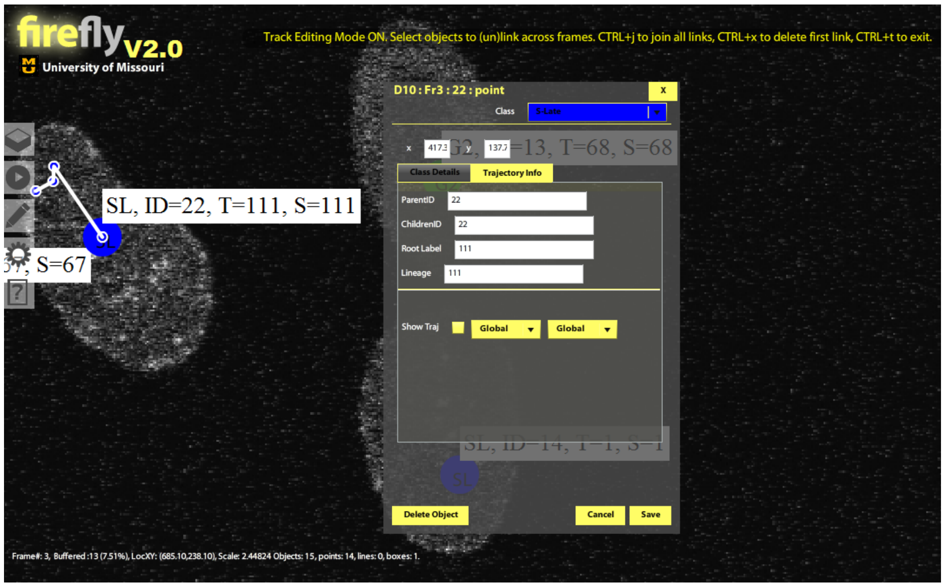
Lineage: 111

Show Traj: Global, Global

Frame#: 10, Buffered :13 (7.51%), LocXY: (754.20,167.80), Scale: 1.97753 Objects: 24, points: 24, lines: 0, boxes: 0.

Figure 4.20 Trajectory of Object 22 from frame 0 to frame 10

During track correction, object at Frame 3 is identified as a spurious object. User can manually delete this object using the 'delete' keyboard key or using the delete button available on the attribute window. When a manual delete operation is performed, application makes a PHP server call to update parent and ChildrenID objects and to propagate the Root Label. Since Object 1 in Frame 2 has it ChildrenID as 1, which does not exist anymore, the algorithm will update its ChildrenID to -1. Similarly ParentID of Object 1 from Frame 4 is updated to -1 and a new track/Root Label is propagated as shown in figures 4.21, 4.22 and 4.23.


Track Editing Mode ON. Select objects to (un)link across frames. CTRL+j to join all links, CTRL+x to delete first link, CTRL+t to exit.

D10 : Fr3 : 22 : point

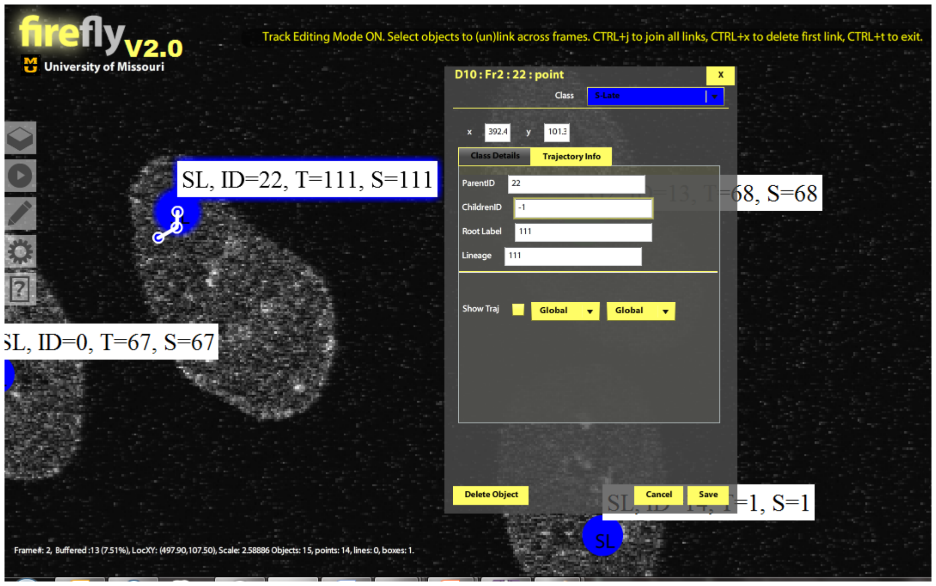
Class: S-Late

x: 417.3 y: 137.7

ParentID: 22
ChildrenID: 22
Root Label: 111
Lineage: 111

Frame#: 3, Buffered :13 (7.51%), LocXY: (685.10,238.10), Scale: 2.44824 Objects: 15, points: 14, lines: 0, boxes: 1.

Figure 4.21 Object 22 deletions in the frame 3


Track Editing Mode ON. Select objects to (un)link across frames. CTRL+j to join all links, CTRL+x to delete first link, CTRL+t to exit.

D10 : Fr2 : 22 : point

Class: S-Late

x: 392.4 y: 101.3

ParentID: 22
ChildrenID: -1
Root Label: 111
Lineage: 111

Frame#: 2, Buffered :13 (7.51%), LocXY: (497.90,107.50), Scale: 2.58886 Objects: 15, points: 14, lines: 0, boxes: 1.

Figure 4.22 ChildrenID of Object 22 update in the frame 2

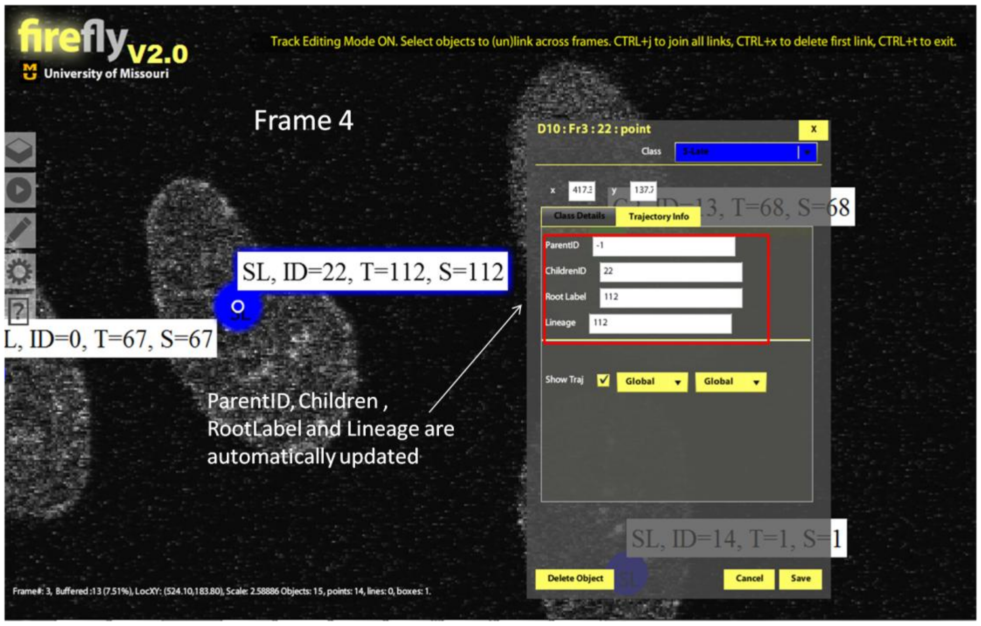
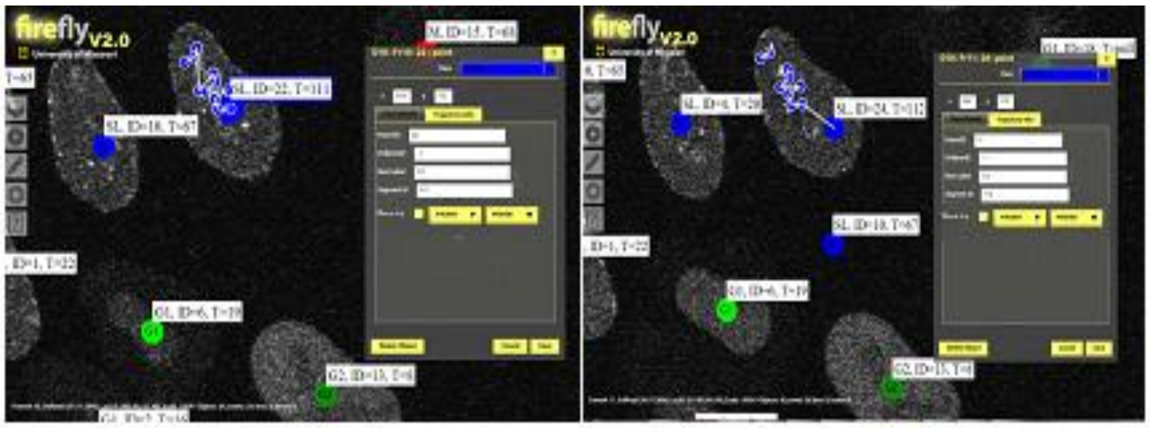

fireflyV2.0
University of Missouri
Track Editing Mode ON. Select objects to (un)link across frames. CTRL+j to join all links, CTRL+x to delete first link, CTRL+t to exit.
Frame 4
D10 : Fr3 : 22 : point
Class
x 417.3 y 137.7
Class Details
Trajectory Info
ParentID -1
ChildrenID 22
Root Label 112
Lineage 112
Show Traj
Global
Global
SL, ID=22, T=112, S=112
L, ID=0, T=67, S=67
SL, ID=14, T=1, S=1
ParentID, Children , RootLabel and Lineage are automatically updated
Delete Object
Cancel
Save
Frame#: 3, Buffered :13 (7.51%), LocXY: (524.10,183.80), Scale: 2.58886 Objects: 15, points: 14, lines: 0, boxes: 1.

Figure 4.23 Attributes of Object 1 updated in the frame 4

The new track label will be the maximum value retrieved from the database added one. This new label is propagated to all the descendant objects of Object 1 (at Frame 3) in the following frames (frame 4 to frame 8) as shown in the Figure 4.24.



(b) Before Delete Operation **(b) After Delete Operation**

Figure 4.24 Trajectories before and after delete operation

4.6 False Splits

In this case, the track information consists of a false split of an object. This will lead to wrong perception of two different tracks as a single track or creating a new branch to track which does not exist originally. Figure 4.25 shows false split of Object 3 at Frame T+1 into two objects 2 and 5; where originally there exists no split. User must manually edit the ChildrenID list of an Object 3 at Frame T+1 and delete Object 5 at Frame T+2 and Object 6 at Frame T+3.

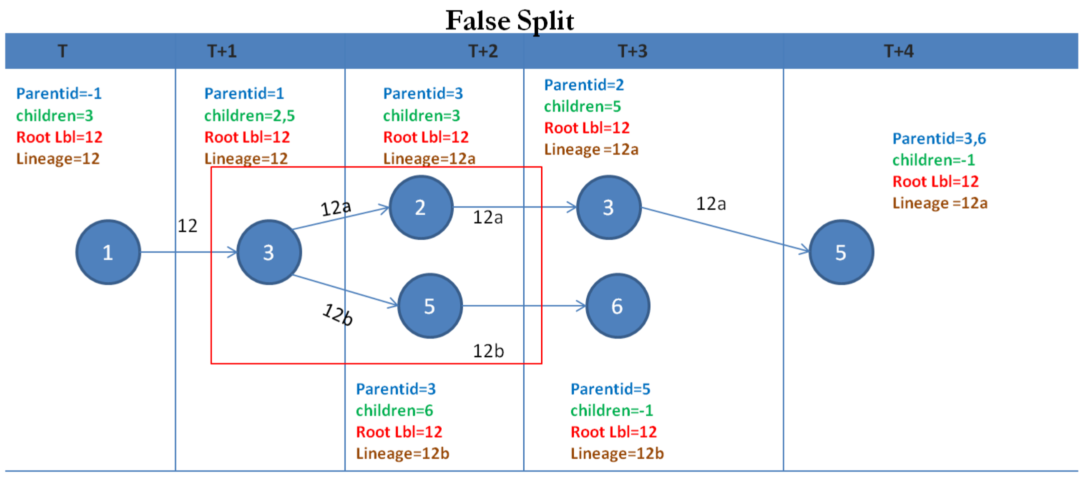


Figure 4.25 Track information of false object split in a frameset

Figure 4.26 shows the track after false split correction, which involves user to change the ChildrenID of Object 3 at frame T+1 and algorithm will update the Root Labels and Lineage of its descendants and updates Object 5 at T+2 and its descendants with new Root Label and Lineage.

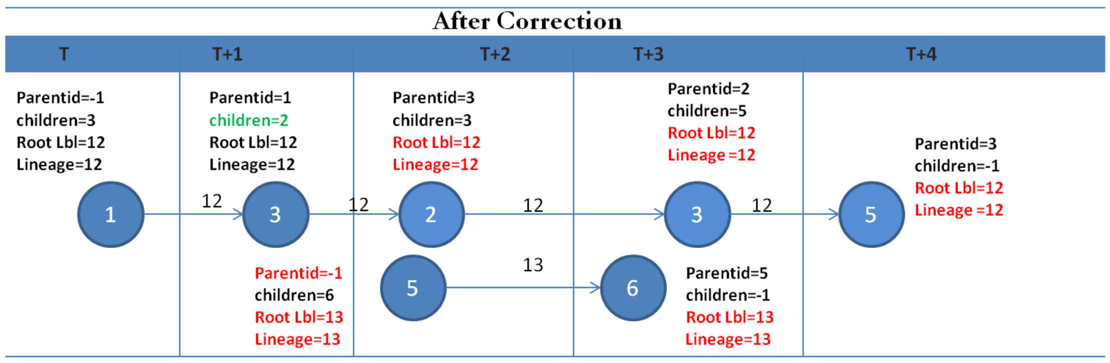


Figure 4.26 Track information **after** correcting false object split

4.7 False Merges

In this case, there is a false merge of objects. User must manually edit the parent list of the merged object. The ChildrenID lists of parent objects are updated automatically. New Root Label and new Lineage are propagated to all the descendants. In Figure 4.27, Object 2 at Frame T+2 is false merge with false parent Object 7. User must manually update the ChildrenID list of Object 7 at Frame T+1 or the parent list of Object 2 at Frame T+2.

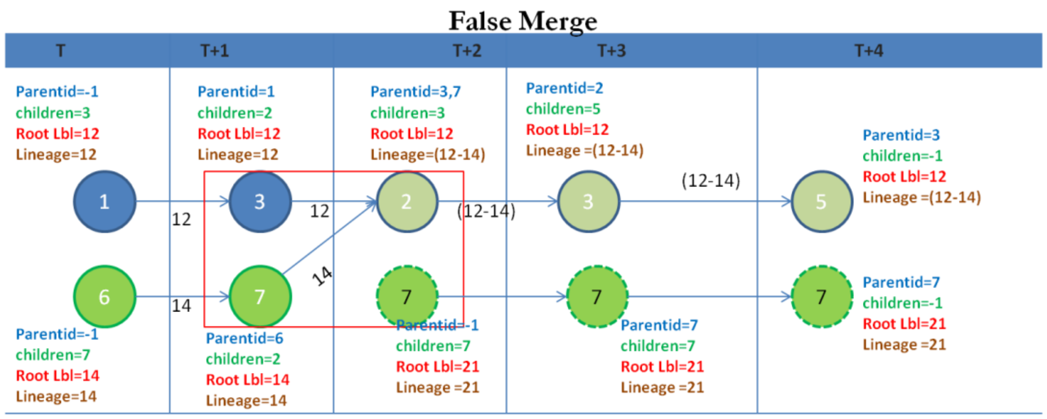


Figure 4.27 Track information of false merge of objects

Figure 4.28 shows the automatic propagation of Root Label and Lineage after correcting the ChildrenID of Object 7 at T+1.

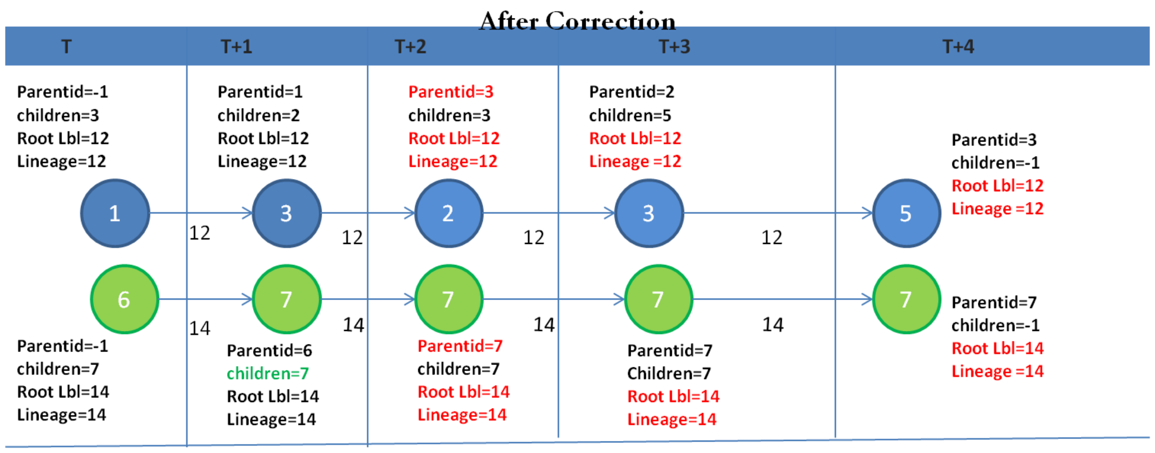


Figure 4.28 Track information **after** correcting false merge of objects

4.8 ID Switching

In this case ID is either ParentID or ChildrenID. Figure 4.19 depicts the example of ID switching. The ChildrenID of Object 2 and Object 7 at Frame T+2 are switched due to an algorithm error. To correct the track information and user must manually switch the ParentID of Object 3 and Object 7 at Frame T+3 or manually switch the ChildrenID of Object 2 and Object 7 at Frame T+2.

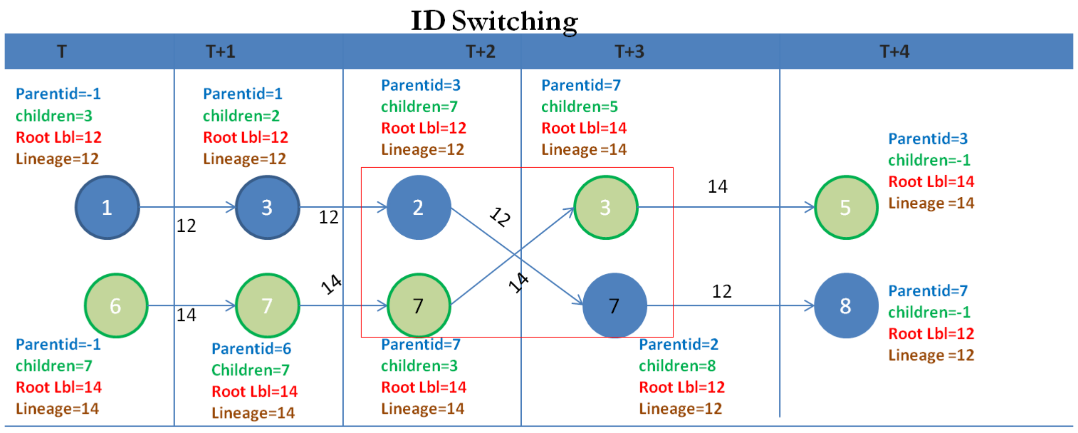


Figure 4.29 Track information of objects whose ParentID and ChildrenID attributes are switched

Figure 4.30 shows the automatic Root Label and Lineage propagation of correcting the ParentID or ChildrenID values.

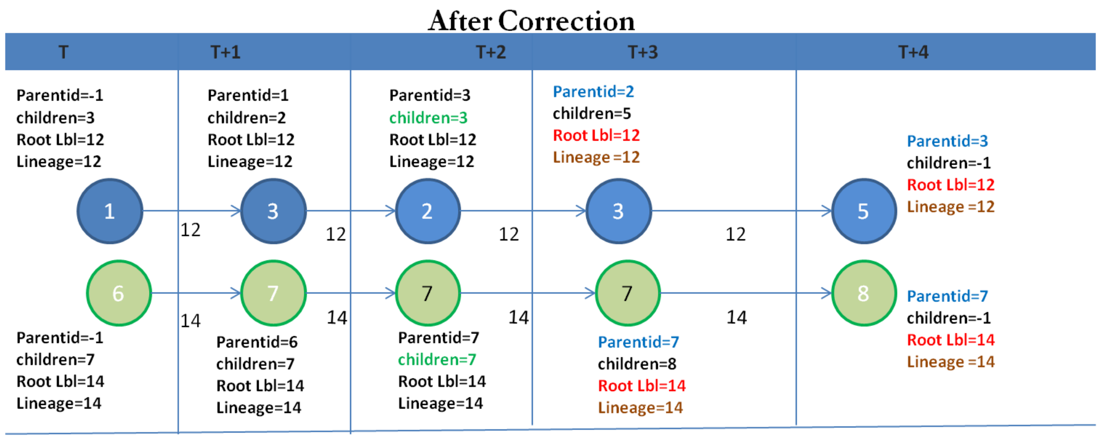


Figure 4.30 Track information of objects after correcting ID Switching

4.9 Efficient FireFly Update Operations after a User Edit

All these cases are examined and solved by using a reliable algorithm which automatically propagates the Root Labels and Lineages in lineage data representation. These algorithms are implemented in PHP and are invoked by the FireFly when needed. Figure 4.31 shows the consequent FireFly system functions based on user attribute edit operation. FireFly uses event handling functions for every user interactive operation. When user performs delete or edit operations on the object attributes, the corresponding event handler function is invoked and passes a request to the server for necessary updates. PHP on the server handles this request and execute the update functions to propagate the changes.

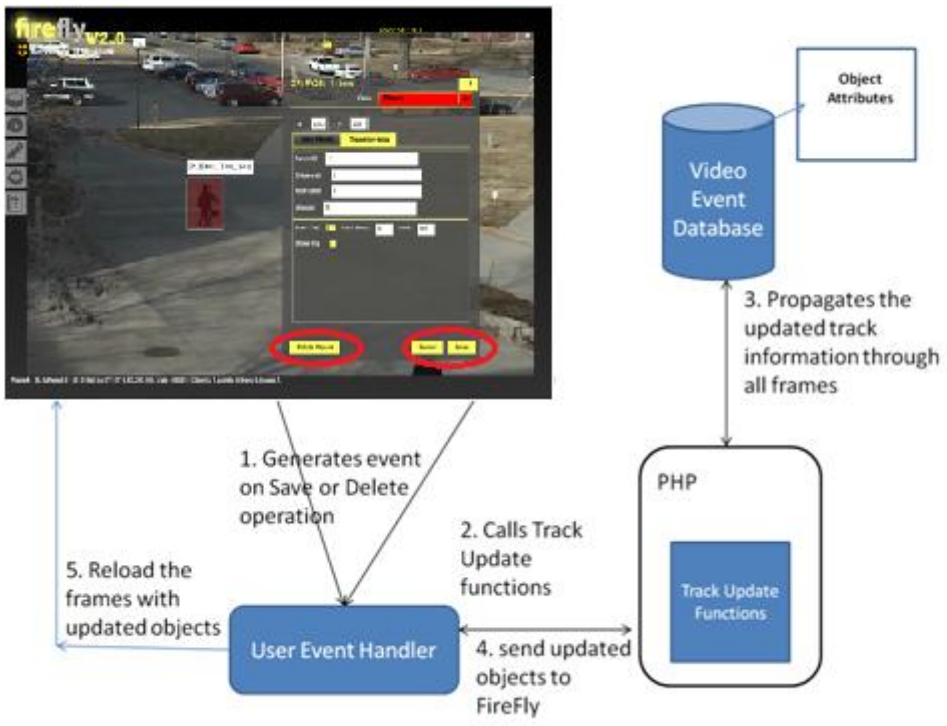

fireFly v2.0
Object Attributes
Video Event Database
3. Propagates the updated track information through all frames
1. Generates event on Save or Delete operation
2. Calls Track Update functions
PHP
Track Update Functions
5. Reload the frames with updated objects
User Event Handler
4. send updated objects to FireFly

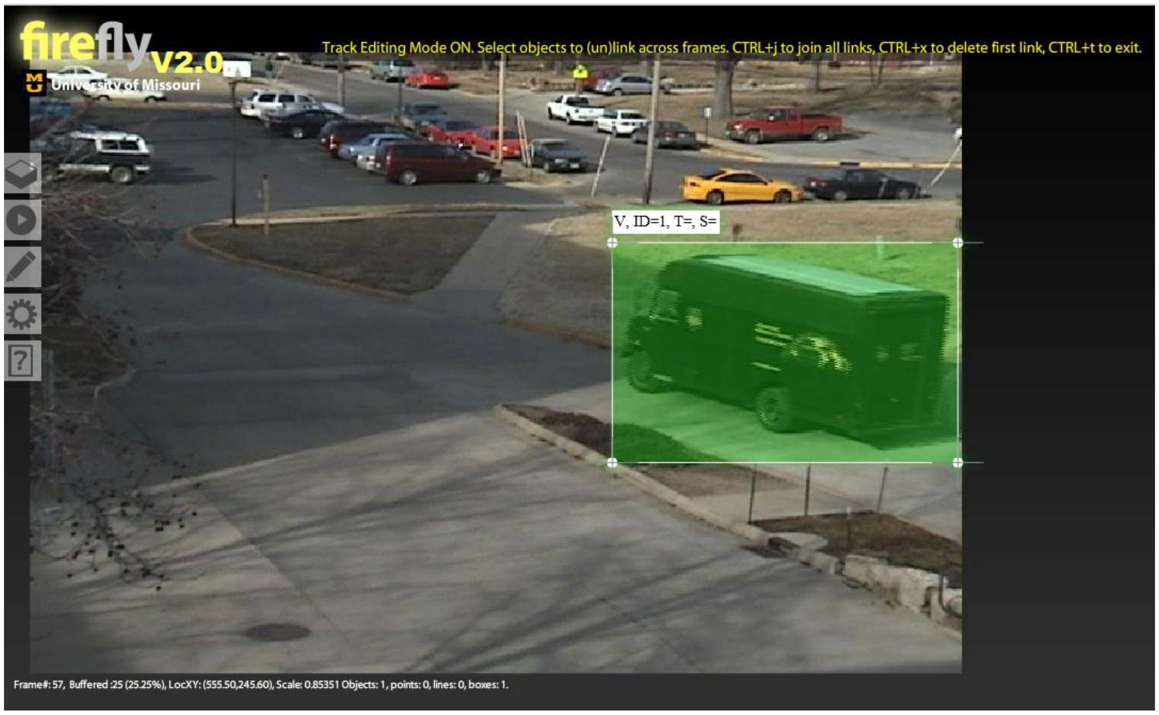
Figure 4.31 Sequence of update operation in FireFly

‘Attributes’ is an important table in the database which contains attribute information of all the objects. This table data is fetched by the update functions to update the changes in the track information. PHP functions make a ‘return call’ to the event handler on updating the table data and event handler will reload all the frames in a dataset to reflect the new attribute values. The implementation of PHP functions is discussed in detail in the chapter 5.

4.10 Interactive Track Editing Methods

Previously discussed track editing operations are manual operations which requires user to manually edit the ParentID and Children attributes for every object. In creating manual ground truth for large datasets, this procedure will be a time consuming and complex process for the user to make frequent frame change operations for knowing object's parents and children. A fast track editing approach is recommended with more interactivity and less manual edit operations. This interactive track editing method involves selecting objects in different frames and applying two shortcut keys: Ctrl+j to create a track and Ctrl+x to delete a track. The steps involve in **interactive track creating/connecting** are

Step 1: Ctrl+t to enable Track Mode ("Track Mode: Enabled" appears on screen as text feedback to user) as shown in the Figure 4.32.



Track Editing Mode ON. Select objects to (un)link across frames. CTRL+j to join all links, CTRL+x to delete first link, CTRL+t to exit.

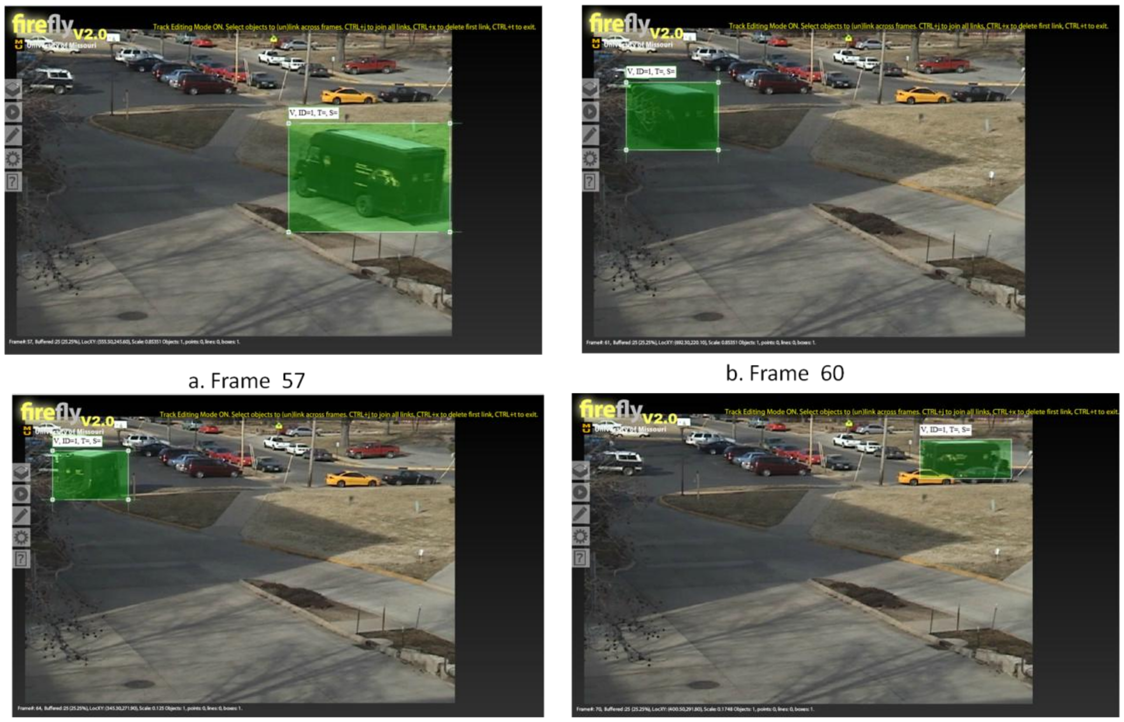
Frame#: 57, Buffered :25 (25.25%), LocXY: (555.50,245.60), Scale: 0.85351 Objects: 1, points: 0, lines: 0, boxes: 1.

Figure 4.32 Track mode is enabled in the frame 57

Step 2: Select object in each frame to connect/ create a track (only one object can be selected in each frame).

Step 2a: If the wrong object is selected then deselect the object (by clicking the selected object) and select the correct object.

Step 2b: Repeat Step 2 for each new frame as shown in Figure 4.33. In this scenario UPS vehicle objects were selected from frame 57 to frame 70 to create a track and Figure shows some of sample frames of object selected.


firefly V2.0
University of Missouri
Track Editing Mode ON. Select objects to (un)link across frames. CTRL+j to join all links, CTRL+x to delete first link, CTRL+t to exit.
V, ID=1, T=, S=

a. Frame 57

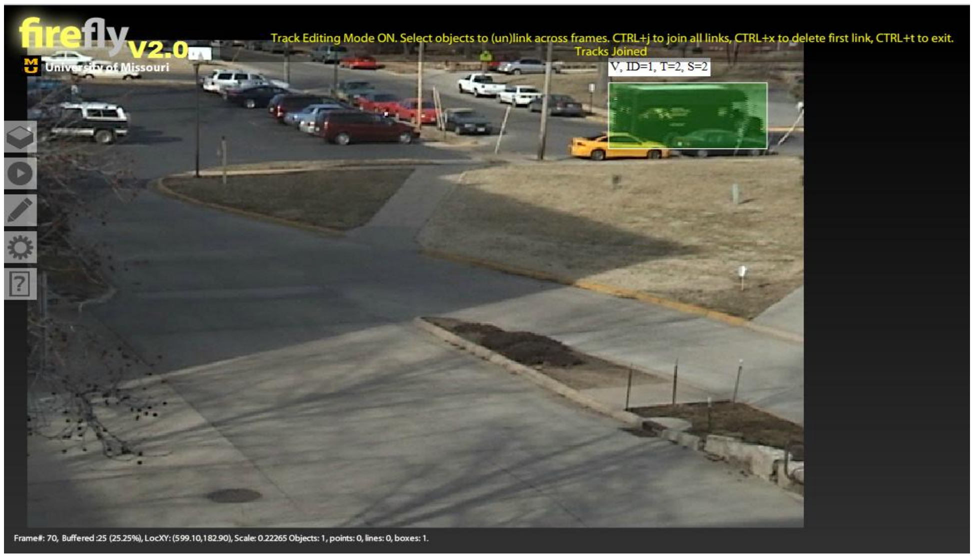
b. Frame 60

c. Frame 65

d. Frame 69

Figure 4.33 Sample frames objects selected to create track

Step 3: After selecting object in last frame then Ctrl+j to join all selected objects in to a single track automatically. Parent and child relationships are added to the Attribute panel automatically and new Root Label is generated if there is no track exists or Root Label is propagated ("Track joined" appears on screen as text feedback to user") as shown in the Figure 4.34.



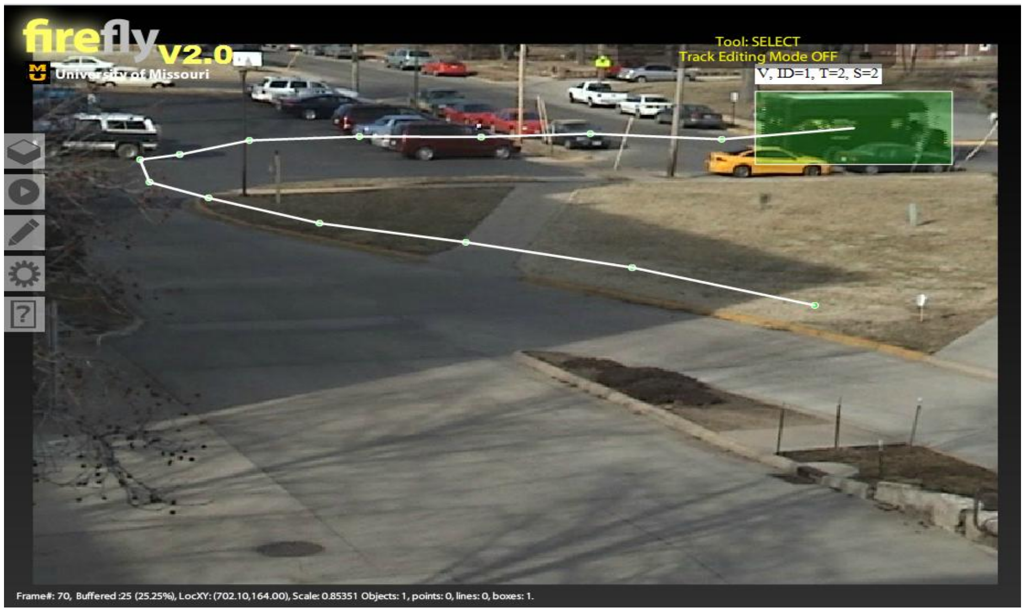
Track Editing Mode ON. Select objects to (un)link across frames. CTRL+j to join all links, CTRL+x to delete first link, CTRL+t to exit.
Tracks Joined

V, ID=1, T=2, S=2

Frame#: 70, Buffered :25 (25.25%), LocXY: (599.10,182.90), Scale: 0.22265 Objects: 1, points: 0, lines: 0, boxes: 1.

Figure 4.34 Track is created for selected objects

Step 4: If track creating operations are finished then Ctrl+t to disable track mode. (“Track Mode: Disabled” appears on screen as text feedback to user) as shown in the Figure 4.35. Figure shows the trajectory of UPS vehicle with small circles representing vehicle position in previous frames.



Tool: SELECT
Track Editing Mode OFF

V, ID=1, T=2, S=2

Frame#: 70, Buffered :25 (25.25%), LocXY: (702.10,164.00), Scale: 0.85351 Objects: 1, points: 0, lines: 0, boxes: 1.

Figure 4.35 Track mode is disabled and trajectory of UPS vehicle

The steps involve in **interactive link/edge deleting** are

Step 1: Ctrl+t to enable Track Mode (“Track Mode: Enabled” appears on screen as text feedback to user).

Step 2: Select objects in any two neighboring frames to disconnect/delete a link/edge (only one object can be selected in each frame).

Step 2a: If the wrong object is selected then deselect the object (by clicking the selected object) and select the correct object.

Step 3: Ctrl+x to delete an edge/disconnect the track between the selected objects. New parent and child are added to the Attribute panel automatically. New Root Label is generated and propagated for the new track. (“Track disconnected” appears on screen as text feedback to user”). Figure 4.36 shows UPS vehicle track before and after Ctrl+x operation.

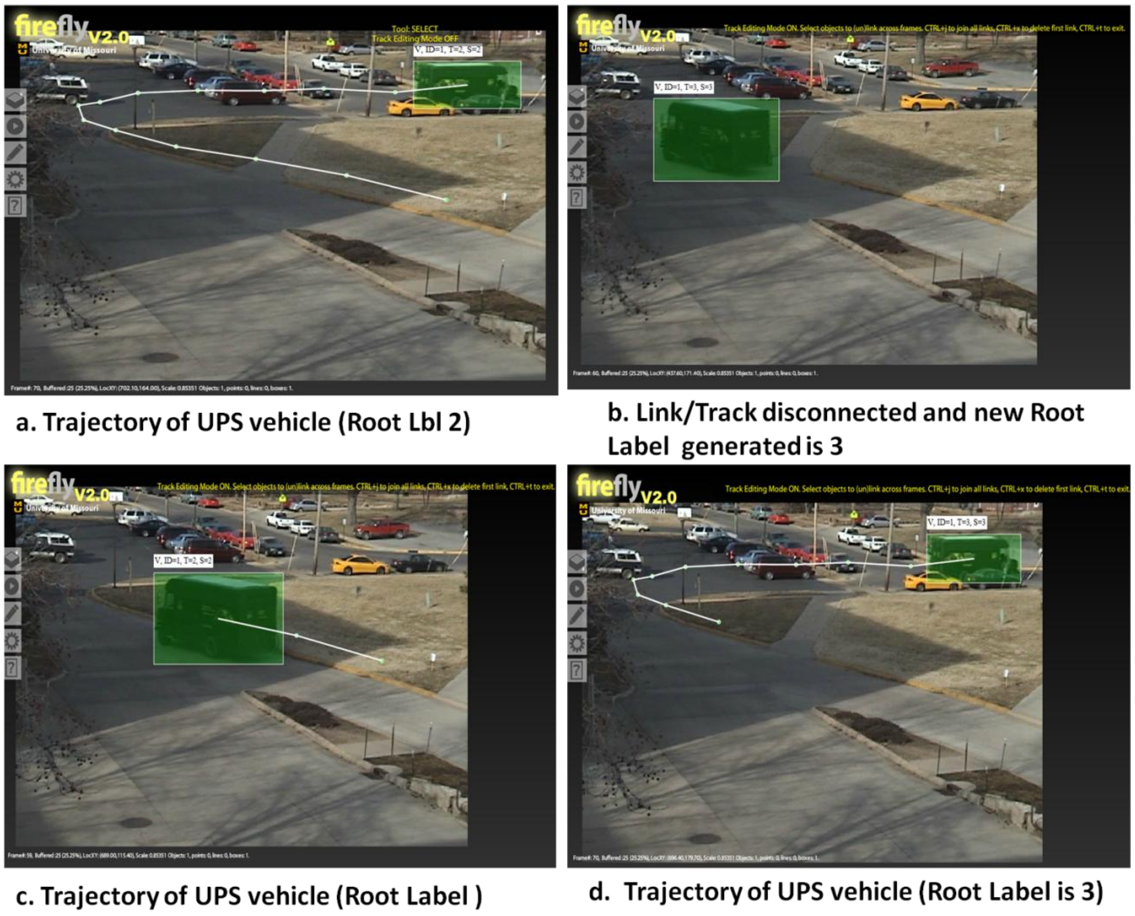

a. Trajectory of UPS vehicle (Root Lbl 2)
b. Link/Track disconnected and new Root Label generated is 3
c. Trajectory of UPS vehicle (Root Label)
d. Trajectory of UPS vehicle (Root Label is 3)

Figure 4.36 Track of UPS vehicle before (a) and after (b, c, d) Ctrl+x operation

Step 4: If link deleting operations are finished then Ctrl+t to disable track mode. ("Track Mode: Disabled" appears on screen as text feedback to user).

Interactive Track Editing helps the users to interactively and quickly perform track corrections. Interactive track editing operations of manual track corrections (section 4.1) are discussed below.

- **Creating track information**: Creating track information using Interactive Track Editing approach involves steps of **interactive track creating/connecting** discussed before.

- **Adding or Extending track information:** Adding/Extending track of an object using Interactive Track Editing approach involves
  - Step 1: Ctrl+t to enable interactive track editing mode
  - Step 2: Select objects which are supposed to join to the existing track and select first/last object (based on direction of extending track) of existing track
  - Step 3: Ctrl+j to propagate the existing track information to the selected objects
  - Step 4 : Ctrl+t to disable Interactive Track Editing mode
- **Deleting spurious points / Splitting tracks:** Deleting spurious point involves normal delete operation and track information will automatically updated. Splitting track using Interactive Track Editing approach involves steps of **interactive link/edge deleting** discussed before**.**
- **Joining tracks**: joining tracks using Interactive Track Editing approach involves following steps
  - Step 1: Ctrl+t to enable interactive track editing mode
  - Step 2: To join two tracks, select the leaf/last object of one track and root/starting object of another track.
  - Step 2a: To join multiple tracks, select the leaf/last object of the first track whose information should be propagated to other track objects and select all objects of other tracks.
  - Step 3: Ctrl+j to join tracks or propagate the first track information to other tracks

- Step 4 : Ctrl+t to disable Interactive Track Editing mode

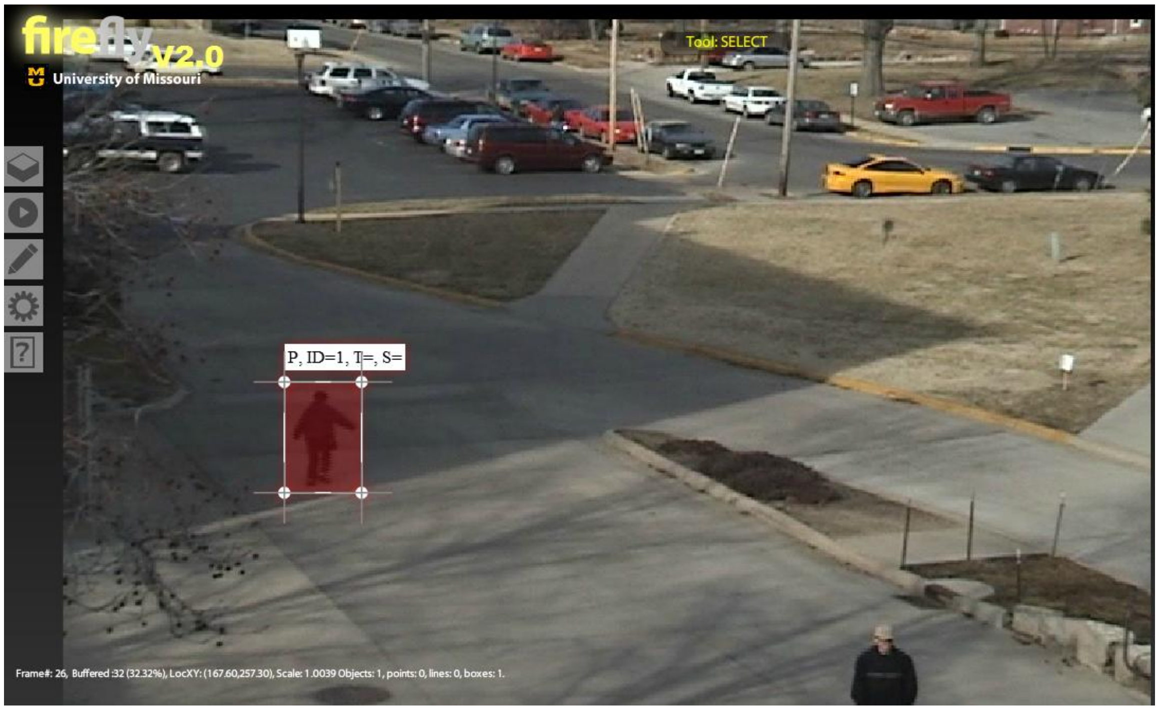
- **False merge:** False merge correction using Interactive Track Editing approach involves following steps
  - Step 1: Ctrl+t to enable interactive track editing mode
  - Step 2: Select false parent object and object of false merge
  - Step 3: Ctrl+x to split the link/track between false parent object and object
  - Step 4 : Ctrl+t to disable Interactive Track Editing mode
- **False split:** False split correction using Interactive Track Editing approach involves following steps
  - Step 1: Ctrl+t to enable interactive track editing mode
  - Step 2: Select false child object and object of false split
  - Step 3: Ctrl+x to split the link/track between false child object and object
  - Step 4 : Ctrl+t to disable Interactive Track Editing mode
- **ID switches:** ID switch correction using Interactive Track Editing approach involves following steps
  - Step 1: Ctrl+t to enable interactive track editing mode
  - Step 2: Select false child object and object of ID switch
  - Step 3: Ctrl+x to split the link/track between false child object and object
  - Step 4: Repeat step 2 and step 3 for all ID switches

- Step 5: Select objects and Ctrl+j to join the split objects to correct ID switches
- Step 6 : Ctrl+t to disable Interactive Track Editing mode

4.11 Single Object Interactive Tracking

During manual ground truth generation in a large datasets using the above interactive track editing operation involves two steps, in first step, user must manually create an object in every frame and in second step, user must select object in each frame and Ctrl+j to join all selected objects in to a single track. This procedure is time consuming in some large datasets with thousands of frames and each frame may consist hundreds of objects. A faster and interactive approach is recommended to generate manual ground truth with track creating. In this method, user can create object and also propagate its track information in each frame with a single click operation at each frame. Steps involved in single object interactive tracking method are

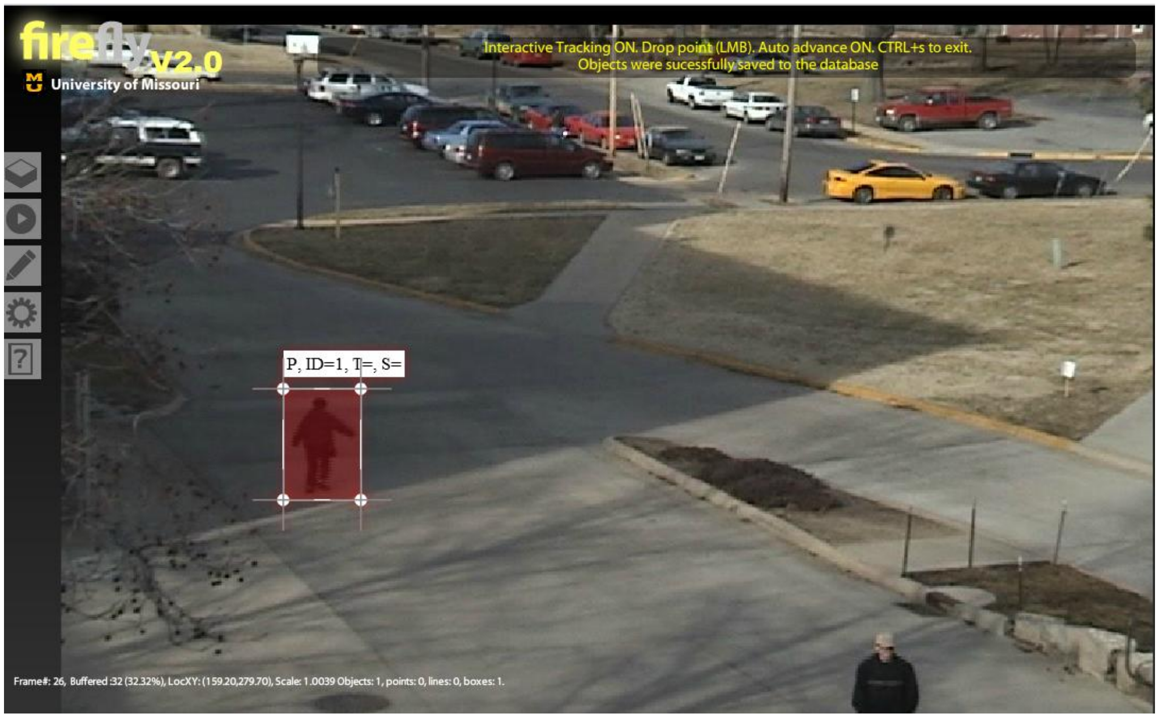
Step 1: Select an object to propagate and create a track as shown in the Figure 4.37. Red colored object selected in the figure is person, its object ID (ID) is 1 and Root Label/Track ID (T) is null (no track).



```
Frame#: 26,  Buffered :32 (32.32%), LocXY: (167.60,257.30), Scale: 1.0039 Objects: 1, points: 0, lines: 0, boxes: 1.
```

Figure 4.37 Object (ID=1) is selected propagate and create track

Step 2: Ctrl+s to enable single object interactive tracking mode ("Single object interactive tracking ON. ctrl-s to exit" appears on screen as text feedback to user). Figure 4.33 represents the screen when Single Object Interactive Tracking Mode is enabled.



Interactive Tracking ON. Drop point (LMB). Auto advance ON. CTRL+s to exit.
Objects were sucessfully saved to the database

Frame#: 26, Buffered :32 (32.32%), LocXY: (159.20,279.70), Scale: 1.0039 Objects: 1, points: 0, lines: 0, boxes: 1.

Figure 4.38 Single Object Interactive Tracking Mode is enabled in frame 26

Step 3: Go to next frame and click on the frame to create an object (previously selected object at step 1 is created at mouse clicked location)

Step 3a: After mouse click operation object is created and trajectory of an object is drawn automatically. If Auto Advance option is selected in the Save Display settings widget, frame will be automatically advanced to next frame with each mouse click operation. If Auto Advance option is disabled then user must manually advance to next frame.

Step 3b: Repeat Step 3 for each new frame.

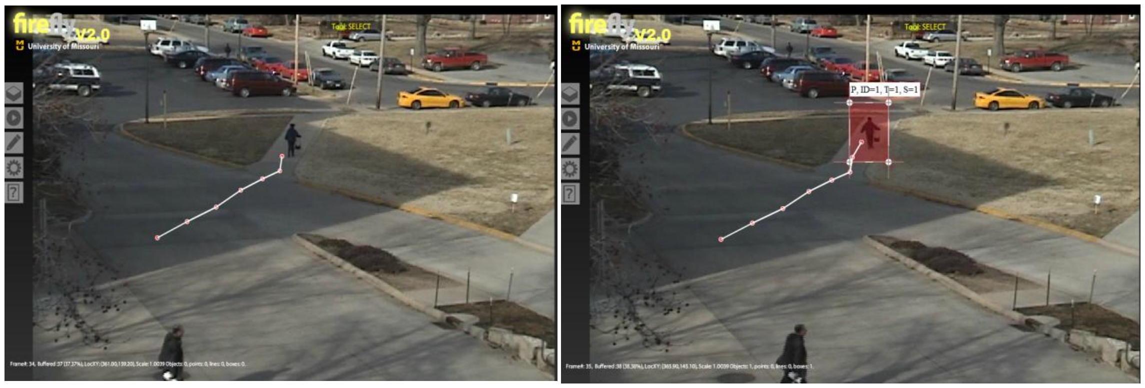
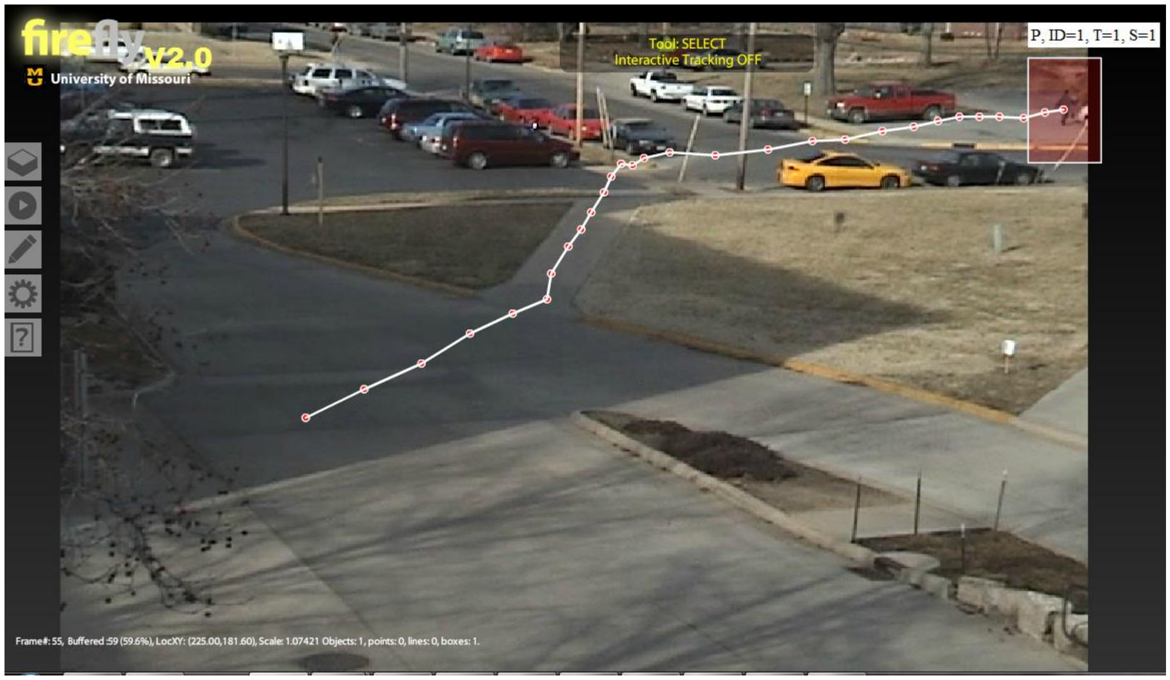


Figure 4.39 Single Object Interactive Tracking with Auto Advance feature (left) and without Auto Advance feature (right)

Step 4: After selecting object in last frame then Ctrl+s to disable single object interactive tracking mode (as shown in Figure 4.34) and automatically create a track for the objects created in single object tracking mode as show in Figure 4.35 with new Root Label 1 (T= 1).



Frame#: 55, Buffered :59 (59.6%), LocXY: (225.00,181.60), Scale: 1.07421 Objects: 1, points: 0, lines: 0, boxes: 1.

Figure 4.40 Single Object Interactive Tracking Mode is disabled in frame 55

Chapter 5

Track Updating and Management

5.1 Track Management Implementation

FireFly helps the user to change the ParentID or ChildrenID anytime and the Root Label is propagated to its descendants automatically. Whenever there is an edit operation by the user, values are passed to the algorithm on the server for automatic propagation of Root Label and Lineage to the descendants. Algorithm supports all the cases discussed in the previous chapter. This algorithm is divided in to 3 stages of updates:

- Updating previous frame objects
- Updating current frame objects and their descendants
- Updating subsequent frame objects and their descendants

5.2 Updating Previous Frame Objects

This module consists of two steps. In the first step we retrieve objects from the previous frame which are either parents of current frame edited object or an old parent, which is not a parent of current object following editing. In the second step, the ChildrenID attribute of retrieved objects are modified according to the changes made by the user. For example, in the Figure 5.1, ParentID of Object 2 in Frame T+1 is changed manually from 3 to 1.

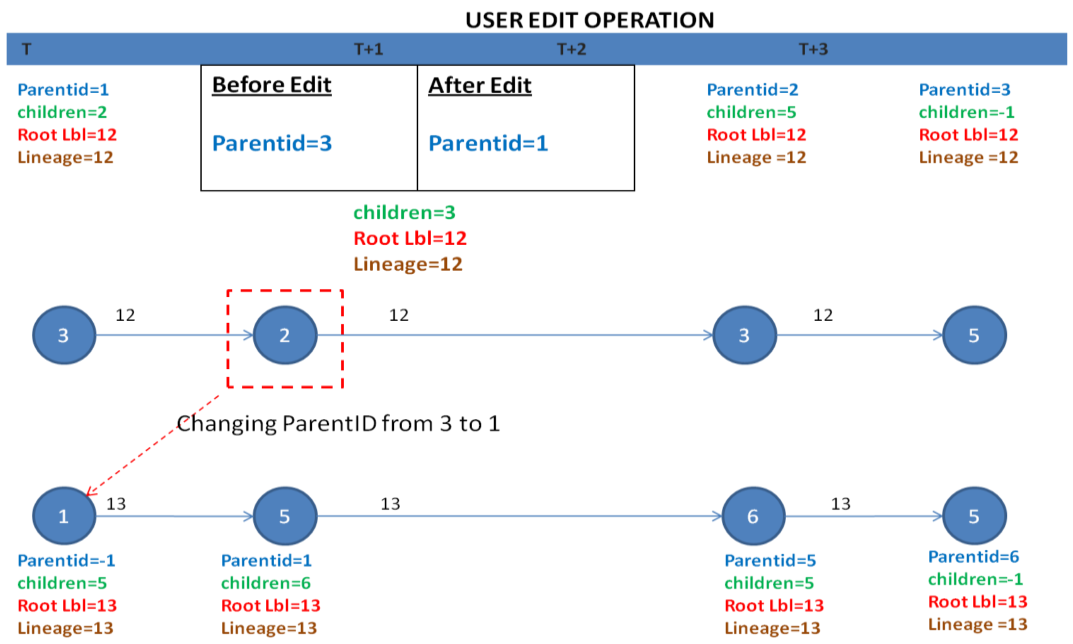


Figure 5.1 User edited ParentID of Object 2 from 3 to 1

When user does this change the first step in the algorithm will retrieve objects from the previous frame. These objects are either parents of Object 2 or previous frame objects which have Object 2 in their ChildrenID list. Therefore, from previous Frame T, objects 3 and 1 are retrieved for an update. In the next step ChildrenID attribute of these objects are compared with user edited object and modifies accordingly. ChildrenID attribute of Object 3 in Frame T is updated from ‘2’ to ‘-1’, since it does not have any ChildrenID in its list. Similarly, for Object 1 in Frame T, ChildrenID value is updated from ‘5’ to ‘5, 2’ as Object 2 is added to its ChildrenID list by the user.

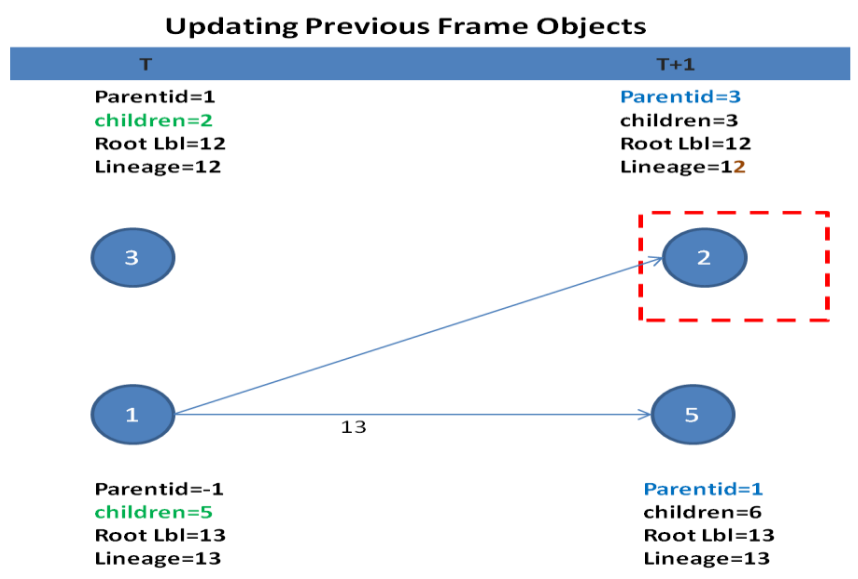


Figure 5.2 Algorithm modifying previous frame objects

Algorithm 1 Updating Previous Frame Objects

Input: ObjID

Output: None//Updates all previous frame objects

```
1.  prevframeobjs[] = getPreviousFrameObjects(ObjectID, ParentID, Frameno);
2.  foreach (object in previousframeobjs) do
3.  if ( object is a new parent) then
4.  add ObjID to object->ChildrenID
5.  end if
6.  if ( object->ChildrenID contains objID but not a parent) then
7.  remove ObjectID from object->ChildrenID
8.  if object->Children== null then object->Children= -1
9.  end if
10. end for
```

Algorithm 1: Updating previous frame objects

5.3 Updating Current Frame Objects and Their Descendants

This module updates the Root Label and Lineage of current frame objects according to the new siblings in the current frame. The current frame objects are retrieved which are in the ChildrenID list of ParentID object. In the Figure 5.3, from the ParentID of Object 2, Object 5 is identified as its sibling. Object 2, Object 5 and their descendants are update with new Lineage.

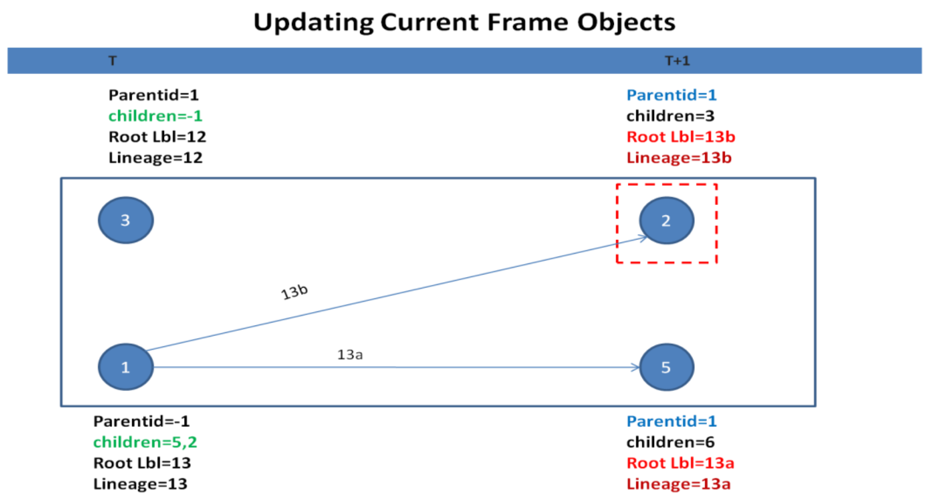


Figure 5.3 Algorithm modifying the current frame objects and their descendants

Algorithm 2: Updating current frame objects

Input : ObjID

Output: None

Functionality: Update Root Label and Lineage of sibilings

```
1.  siblingameobjs[] = getPreviousFrameObjects(ObjectID, ParentID, Frameno);
2.   Foreach siblingameobjs
3.  If parent -> chidrenlist is greater than 1 then
4.     Update Lineage of child with    appended alphabet
5.  End if
6.  End for
```

Algorithm 2: Updating current frame objects

5.4 Updating Subsequent Frame Objects and Their Descendants

This module is the combination of the above two modules. It retrieves the objects from subsequent frame and updates their ParentID, Root Label and Lineage. Similar to the ‘updating previous frame module’, all the objects which are either ChildrenID of edited object or have edited object’s ObjectID in their parents list are retrieved from subsequent frame. For example, in the figure, user changes the ChildrenID attribute from 3 to 6 for Object 2. This module retrieves Object 3 from Frame T+2 and updates its ParentID with -1 in the first step. Similarly, Object 6 is retrieved and adds ObjectID 2 to its ParentID. Now, similar to updating current frame objects module; Lineage and ParentID of these retrieved objects are modified. Object 3 and Object 6 in the Frame T+2 are updated to 'minimum' of their parents Root Label or Maximum Root Label if they don't contain the parent. Lineage is updated to their parent's Lineage in lineage data format or updated to their Root Label if no parent exists.

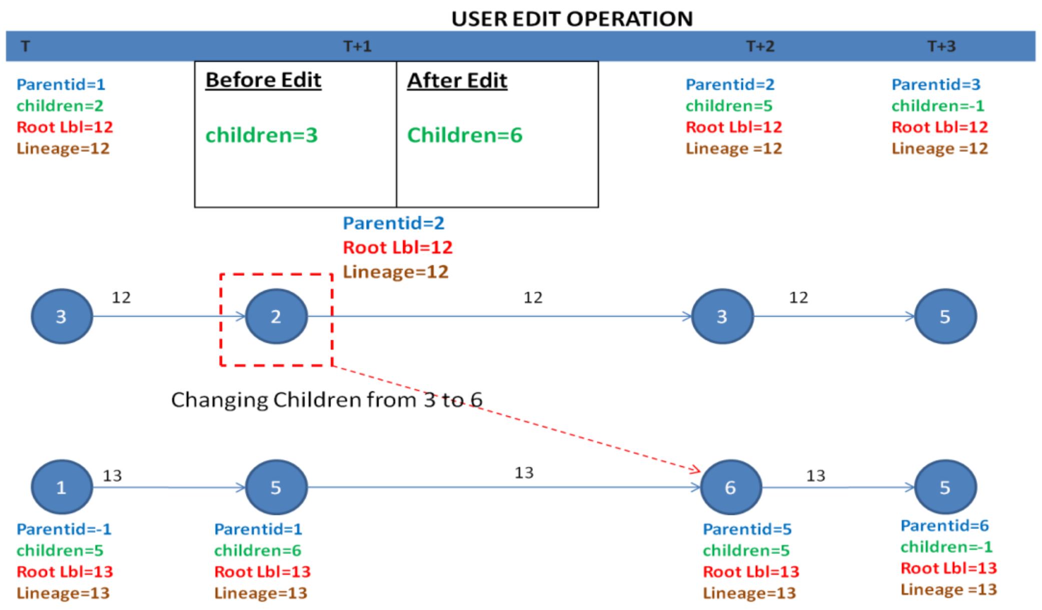


Figure 5.4 User edited ChildrenID of Object 2 from 3 to 6

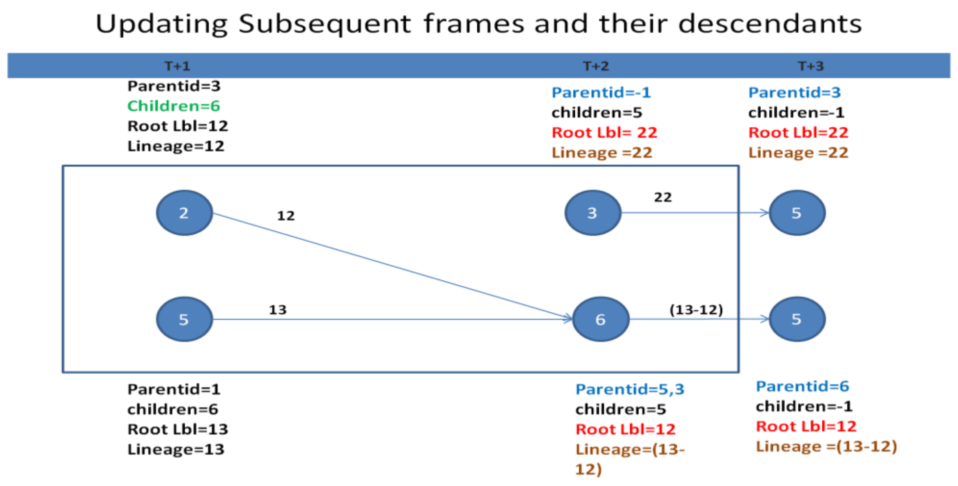


Figure 5.5 Algorithm modifying the subsequent frame objects and their descendants

Algorithm 3: Updating subsequent frame objects and their descendants

Input : ObjID

Output: None

Functionality: Update ParentID, Root Label and Lineage of sibilings

```
1.  nextframeobjects[] = nextFrameObjects(ObjectID, ParentID, Frameno);
2.  Foreach nextframeobjects
3.   if (nextframeobject is new child)then
4.   add ObjID to object->ParentID
5.   update RootLabel and Lineage
6.  end if
7.   if (nextframeobject isnot      child)then
8.  Update object->ParentID
9.  Update RootLabel and Lineage
10. end if
11. end for
```

Algorithm 3: Updating subsequent frame objects and their descendants

Chapter 6

Conclusions & Future Work

Image and video analysis algorithms are validated through ground truth generation and this process is accomplished by using FireFly. FireFly also supports visualization and editing/correcting the ground truth using interactive interface. Being a web application it provides a collaborative environment for researchers to work on a dataset. It is developed as a flexible and extensible tool to add new features on demand. This project extends the features of FireFly with track generation, visualization and editing capabilities. All possible user editing scenarios have been studied and necessary functions are implemented in PHP at the server to propagate the updates. It uses rich internet application Flex to provide interface for editing parent, child and track label properties. These edit operations are handled by the server, which updates the database with consistent track data and reflects the updated track at user interface. The automatic track edit and management reduces the time for the user to propagate changes and maintain consistent track data for accurate analysis.

FireFly provides rich interface for visualization, ground truth generation and editing. Here are some enhancements as an example for future enhancements to FireFly:

- **Segmentation Editing**
  - **Single Object Attributes Updates:** Editing automated segmentation by changing the parameters such as width, height and length of the graphical shapes are required in segmentation editing. This feature is supported for box and line objects. This feature should be extended to other graphical objects like polygon and polyline.
  - **Merge/Split Operations:** In some cases, two close objects are represented as a single object. A split operation of the contour should be provided to the user to correct the segments rather than deleting and redrawing two contours. Similarly, a merge operation should be provided in the case of an incorrect split of an object.
- **Coloring Strategies to Visualize Densely Clustered Large Populations of Objects**
  - For Bacteria dataset, each frame may consist more than 500 objects and representing each individual object may require coloring them uniquely. Currently, all the objects are represented in a single color. Providing different colors is not recommended as some of the colors are difficult to differentiate. Therefore, a four coloring method must be added to identify the objects from its neighbors.
- **Flex 3 to Flex 4.5 Migration**
  - The user interface of FireFly is been developed using Flex 3 SDK which only supports web and desktop applications. But in future FireFly

must be extended to support on mobile and tablet devices. There should be a migration in FireFly from Flex 3 to Flex 4.5 which supports building Flex applications for Google Android OS, BlackBerry Tablet OS and Apple iOS.

- **Create new table for trajectories**
  - In the current version of FireFly, Attribute table is used to store object classification and track attributes, this involve calling multiple database queries on multiple tables to pull the necessary trajectory information. Since, FireFly allows many options to the user to customize the trajectory drawing, a separate table is required to store trajectory related information thereby reducing the database queries and efficiently handling the trajectory information.

Appendix A

General FireFly Features and Enhancements

The first screen in the FireFly system is the login screen which checks user authentication. Once authenticated, the available labs, projects and respective datasets, that belongs to the user are displayed.

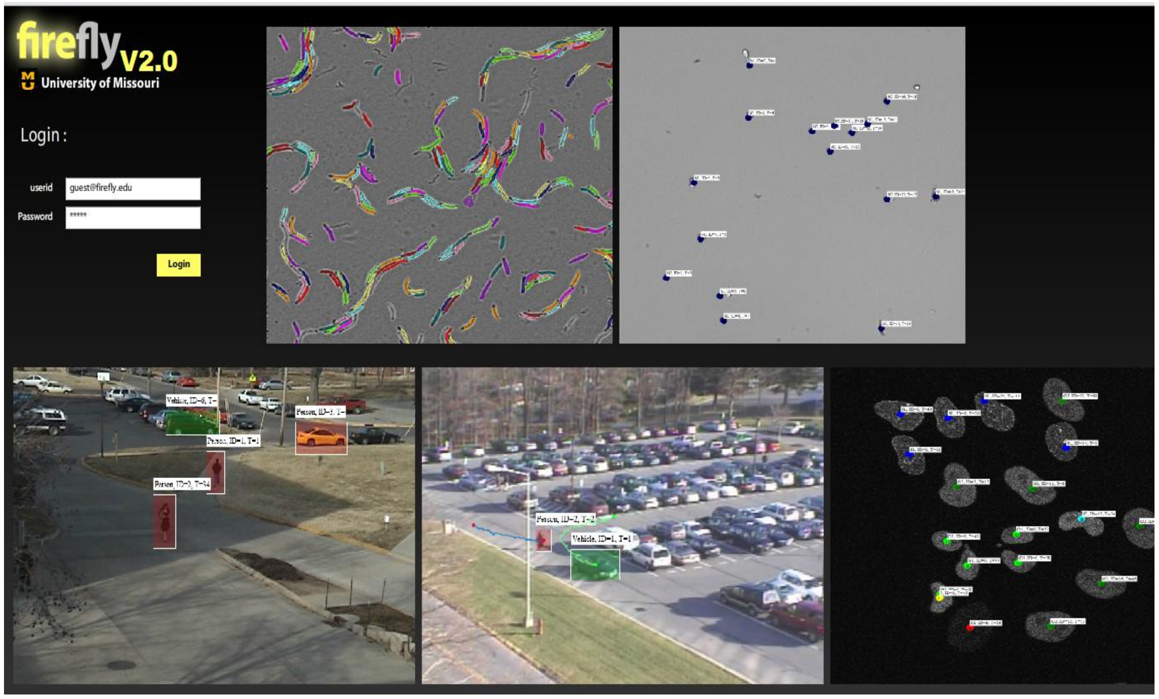
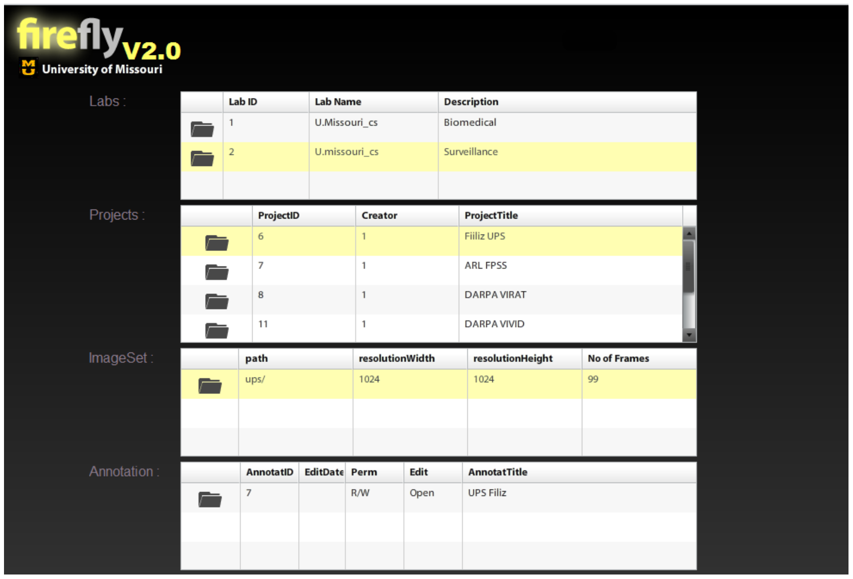


Figure A.1 Login Screen

Since FireFly works on a single database, a read-write conflict is expected when two individual users are working on the same dataset. Therefore, FireFly uses binary semaphore mechanism to lock the dataset from concurrent Read-Write operations. In the Figure A.2, a ‘0’ value for ‘lock’ represents the dataset that is available for Read-Write access and a value of ‘1’ represents datasets that are available for Read-Only access. FireFly also stores the authorizations details for each user. Permission ‘1’ in the Figure A.2 represents Read-Write access for the user and ‘0’ represents Read-Only access.



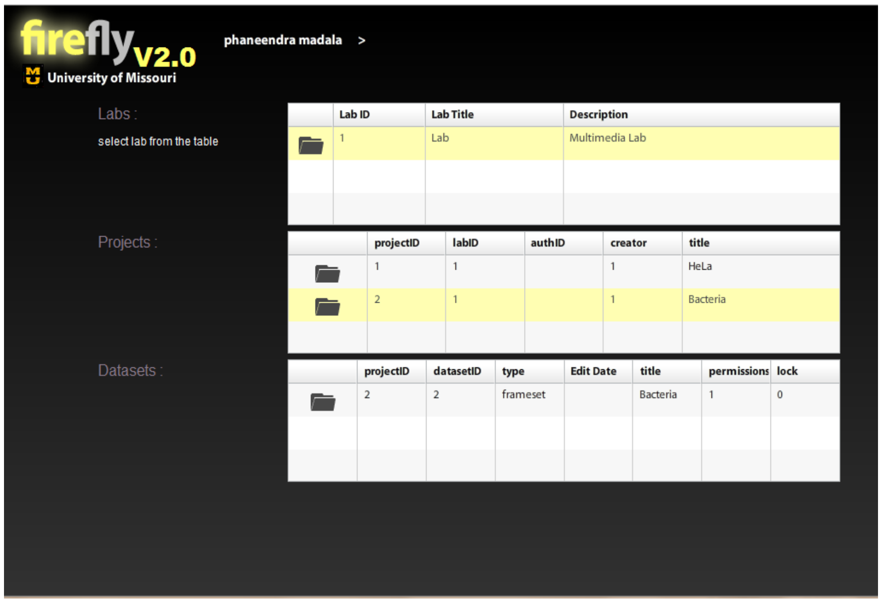
Lab ID	Lab Name	Description
1	U.Missouri_cs	Biomedical
2	U.missouri_cs	Surveillance

ProjectID	Creator	ProjectTitle
6	1	Filiz UPS
7	1	ARL FPSS
8	1	DARPA VIRAT
11	1	DARPA VIVID

path	resolutionWidth	resolutionHeight	No of Frames
ups/	1024	1024	99

AnnotatID	EditDate	Perm	Edit	AnnotatTitle
7		R/W	Open	UPS Filiz

(a)



Lab ID	Lab Title	Description
1	Lab	Multimedia Lab

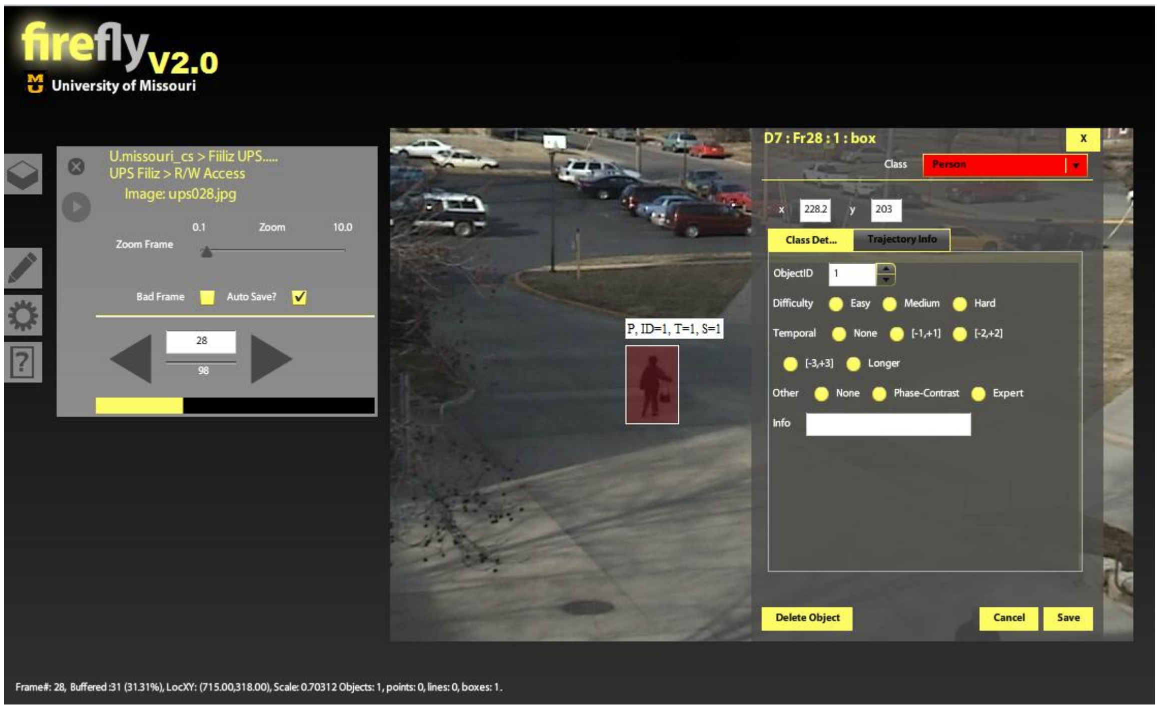
projectID	labID	authID	creator	title
1	1		1	HeLa
2	1		1	Bacteria

projectID	datasetID	type	Edit Date	title	permissions	lock
2	2	frameset		Bacteria	1	0

(b)

Figure A.2 User lab menu (a) new user interface (b) old user interface

The workspace view with HeLa datasets is represented in Figure A.3.



U.missouri_cs > Fiiliz UPS.....
UPS Filiz > R/W Access
Image: ups028.jpg

Zoom Frame 0.1 – 10.0

Auto Save? ✓

28 / 98

D7 : Fr28 : 1 : box

Class: Person

x 228.2 y 203

ObjectID: 1

Difficulty: Easy / Medium / Hard

Temporal: None / [-1,+1] / [-2,+2] / [-3,+3] / Longer

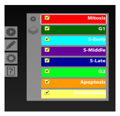
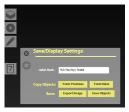
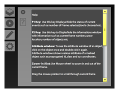
Other: None / Phase-Contrast / Expert

Info:

Frame#: 28, Buffered :31 (31.31%), LocXY: (715.00,318.00), Scale: 0.70312 Objects: 1, points: 0, lines: 0, boxes: 1.

Figure A.3 FireFly workspace view for Read-Write access

FireFly provides five widgets to the left side of the screen. These widgets are listed in the table A.1.

Widget Name	**Reference Figure**	**Controls**	**Purpose**
Class chooser	A.4  Mitosis G1 S-Early S-Middle S-Late G2 Apoptosis Undecidable	• List of all classes with respect to dataset	• Check box is used to show objects of that class. • Uncheck will hide those class objects.
Frame advance widget	A.5	• Zoom control • Frame navigation control • Bad Frame and Auto save controls • Frame Buffer control	• Displays image file name and user access type on top the widget. • Displays current frame number and total number of frames. • User can jump to the specific frame directly. User can set the frame as bad frame. • Auto save will saves all the objects in a frame automatically when frame is changed. • It shows the number of frames buffered and what frames buffered in the frame buffer control.
Drawing tool chooser	A.6	• List of tools	• Currently FireFly consists of point, line, box, polygon and polyline tools.
Settings widget	A.7 	• Save and display controls	• Copy objects will copy the objects from previous frame or previous and display them in the current frame • Save objects will save the current frame objects to the database. Used when Auto Save option is disabled • Export image will make a image of current frame with objects and export the image to the specified location
Help Widget	A.8 	• Tutorial	• Explains how to use each control in each widget and attribute details.

Attribute window	3.4	• Double click on object	• Gives classification and track attributes of an object

Table A.1 FireFly widgets and their controls

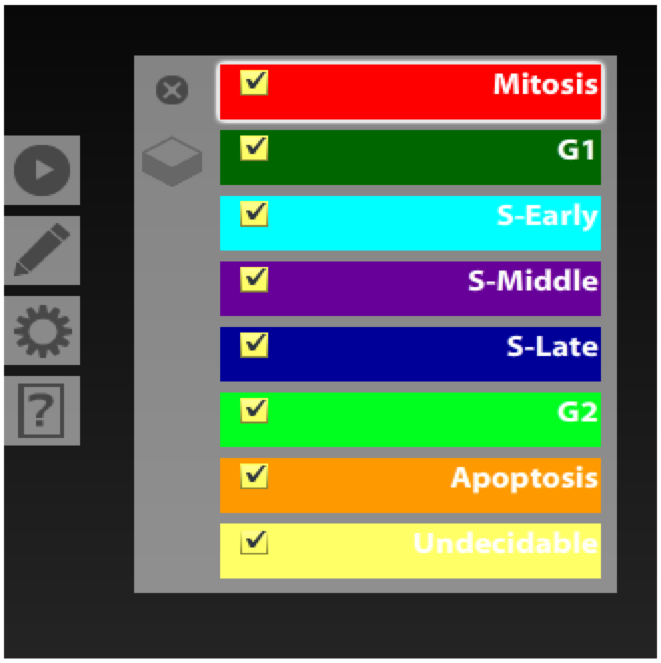


Figure A.4 Class layer chooser widget

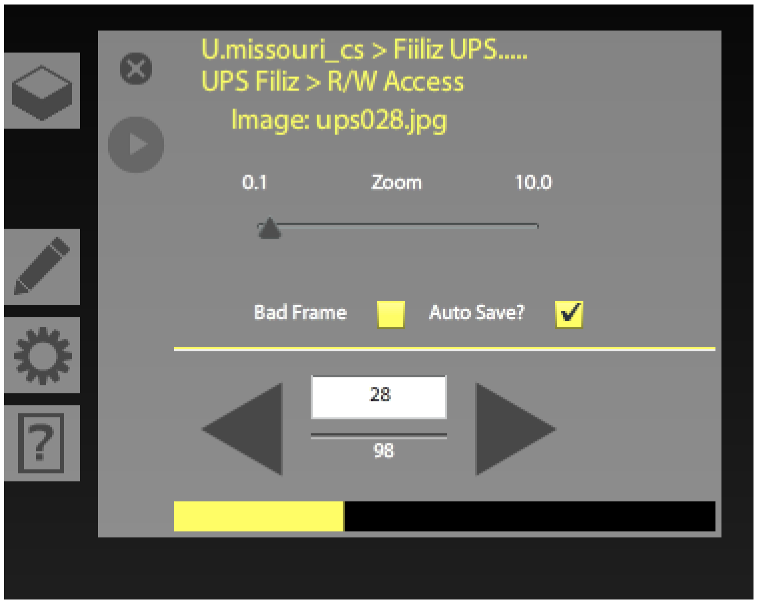


Figure A.5 Frame advance widget

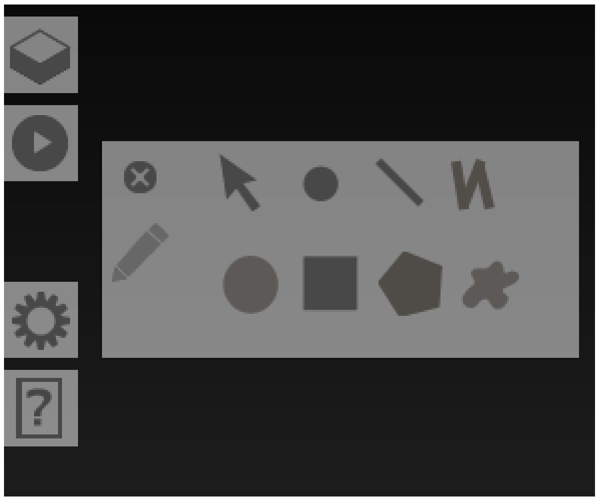
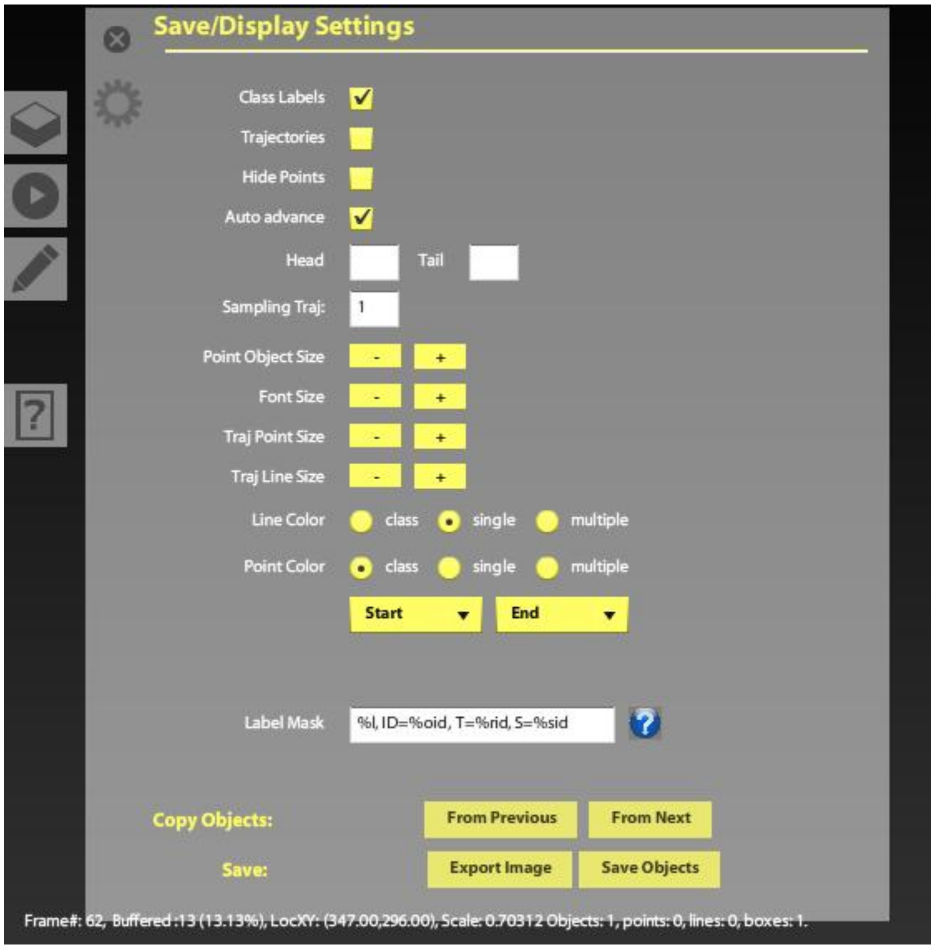


Figure A.6 Drawing Tool Chooser



Save/Display Settings

Class Labels ✓

Trajectories

Hide Points

Auto advance ✓

Head Tail

Sampling Traj: 1

Point Object Size - +

Font Size - +

Traj Point Size - +

Traj Line Size - +

Line Color: class [✓ single] multiple

Point Color: [✓ class] single multiple

Start End

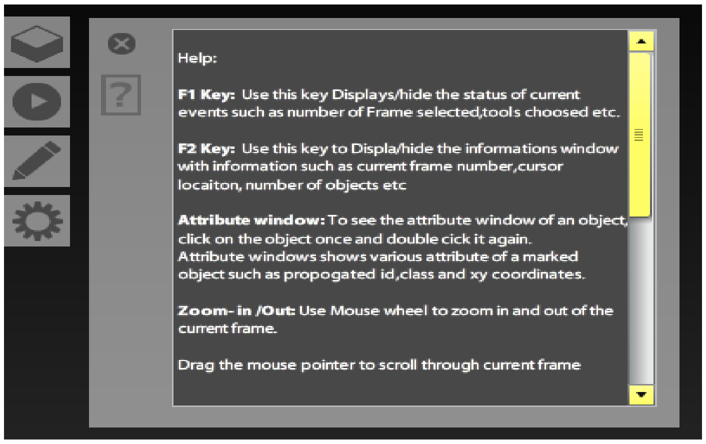
Label Mask: %l, ID=%oid, T=%rid, S=%sid

Copy Objects: From Previous From Next

Save: Export Image Save Objects

Frame#: 62, Buffered :13 (13.13%), LocXY: (347.00,296.00), Scale: 0.70312 Objects: 1, points: 0, lines: 0, boxes: 1.

Figure A.7 Settings control widget



Help:

F1 Key: Use this key Displays/hide the status of current events such as number of Frame selected,tools choosed etc.

F2 Key: Use this key to Displa/hide the informations window with information such as current frame number,cursor locaiton, number of objects etc

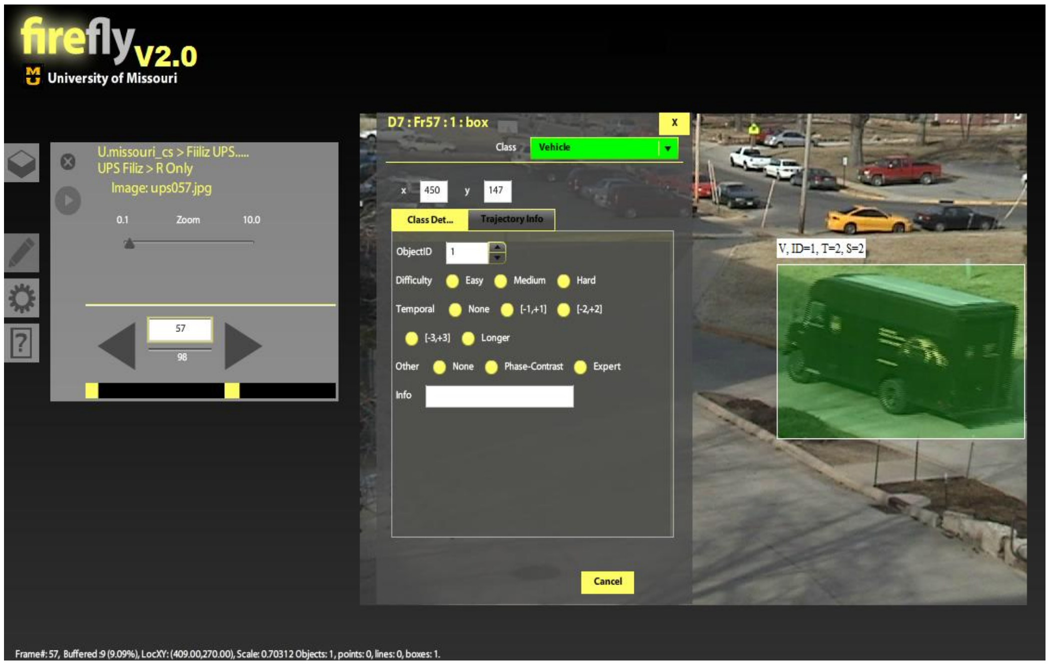
Attribute window: To see the attribute window of an object, click on the object once and double cick it again. Attribute windows shows various attribute of a marked object such as propogated id,class and xy coordinates.

Zoom- in /Out: Use Mouse wheel to zoom in and out of the current frame.

Drag the mouse pointer to scroll through current frame

Figure A.8 Help widget

Figure A.9 shows the sample workspace displayed to the user who has Read-only access. In this scenario, the 'Tool choose' widget, the 'Settings control' widgets and all edit operations are disabled.



U.missouri_cs > Fiiliz UPS.....
UPS Filiz > R Only
Image: ups057.jpg

0.1 Zoom 10.0

57
98

D7 : Fr57 : 1 : box

Class: Vehicle

x 450 y 147

ObjectID 1

Difficulty ● Easy ● Medium ● Hard

Temporal ● None ● [-1,+1] ● [-2,+2] ● [-3,+3] ● Longer

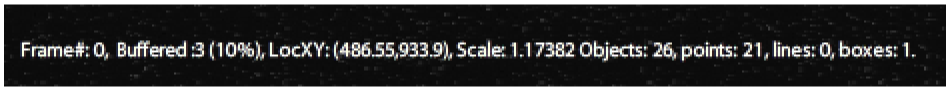
Other ● None ● Phase-Contrast ● Expert

Info

Frame#: 57, Buffered :9 (9.09%), LocXY: (409.00,270.00), Scale: 0.70312 Objects: 1, points: 0, lines: 0, boxes: 1.

Figure A.9 FireFly workspace view for Read-Only access

Figure A.10 shows the status bar which can toggle using F2 (or Fn+F2) key. This status bar is located on bottom of every frame and consists of information about the frame and objects in the frame.



```
Frame#: 0,  Buffered :3 (10%), LocXY: (486.55,933.9), Scale: 1.17382 Objects: 26, points: 21, lines: 0, boxes: 1.
```

Figure A.10 Status bar

Annotating is an easy process in the FireFly. User can select the tool from tool chooser widget and has to click and drag the mouse to draw a line or box object. For polygon and polyline object drawing user needs to click to drop points for the polyline or polygon and press ‘c’ (close) to close the polygon or to stop drawing polyline.

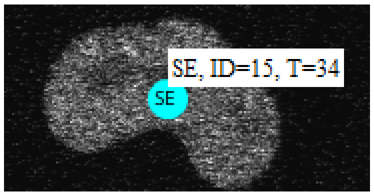


Figure A.11 Point Drawing

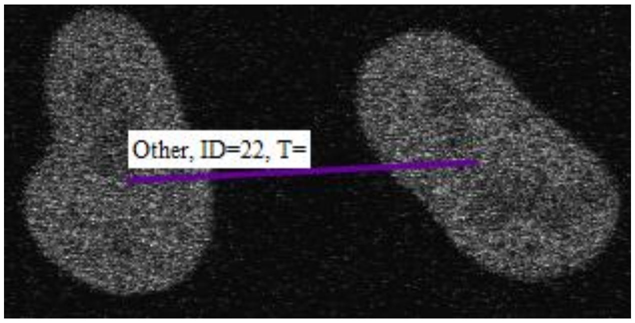


Figure A.12 Line Drawing

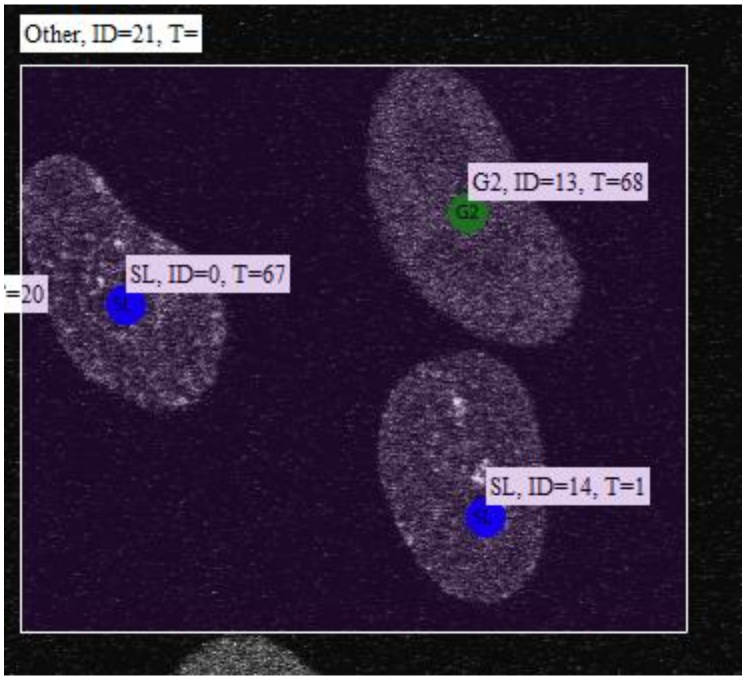


Figure A.13 Box Drawing

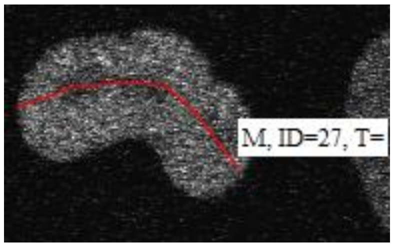


Figure A.14 Polyline Drawing

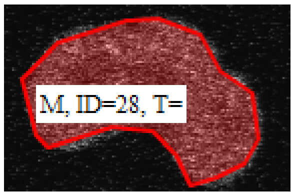
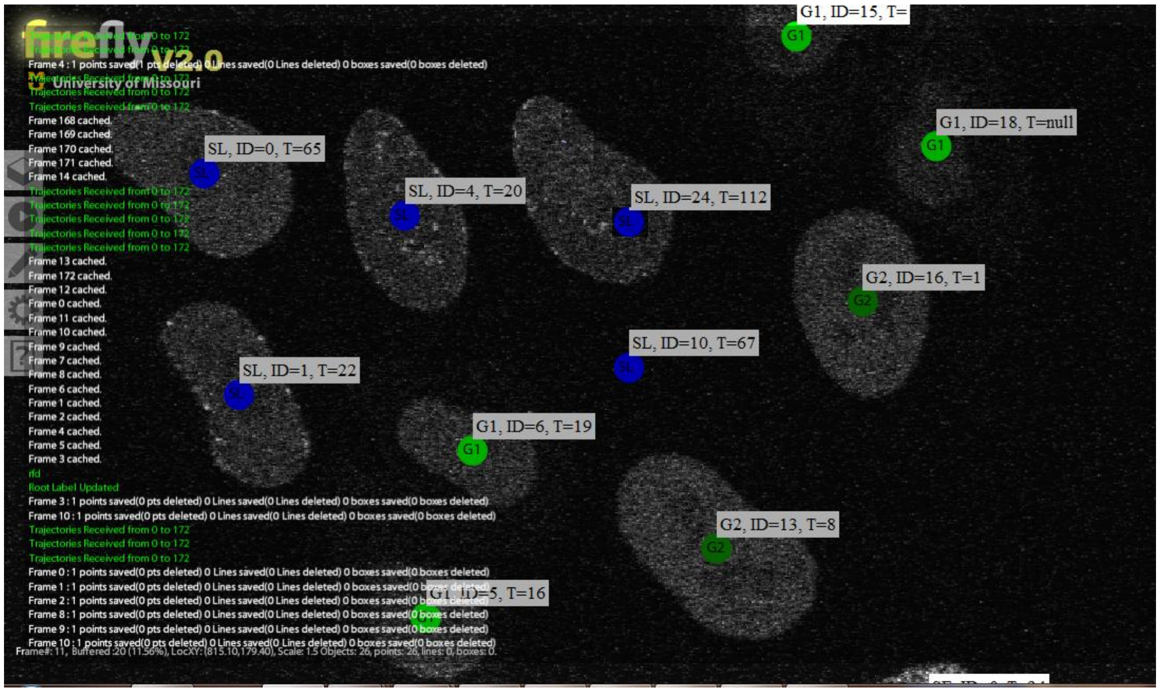


Figure A.15 Polygon Drawing

Figure A.16 represents debug window which can be toggled using F1 key (or Fn+F1 Key if F1 key is overloaded). Debug window gives the information used for the user to understand the operations and modifications made.



```
Trajectories Received from 0 to 172
Trajectories Received from 0 to 172
Frame 4 : 1 points saved(1 pts deleted) 0 Lines saved(0 Lines deleted) 0 boxes saved(0 boxes deleted)
Trajectories Received from 0 to 172
Trajectories Received from 0 to 172
Trajectories Received from 0 to 172
Frame 168 cached.
Frame 169 cached.
Frame 170 cached.
Frame 171 cached.
Frame 14 cached.
Trajectories Received from 0 to 172
Trajectories Received from 0 to 172
Trajectories Received from 0 to 172
Trajectories Received from 0 to 172
Trajectories Received from 0 to 172
Frame 13 cached.
Frame 172 cached.
Frame 12 cached.
Frame 0 cached.
Frame 11 cached.
Frame 10 cached.
Frame 9 cached.
Frame 7 cached.
Frame 8 cached.
Frame 6 cached.
Frame 1 cached.
Frame 2 cached.
Frame 4 cached.
Frame 5 cached.
Frame 3 cached.
rfd
Root Label Updated
Frame 3 : 1 points saved(0 pts deleted) 0 Lines saved(0 Lines deleted) 0 boxes saved(0 boxes deleted)
Frame 10 : 1 points saved(0 pts deleted) 0 Lines saved(0 Lines deleted) 0 boxes saved(0 boxes deleted)
Trajectories Received from 0 to 172
Trajectories Received from 0 to 172
Trajectories Received from 0 to 172
Frame 0 : 1 points saved(0 pts deleted) 0 Lines saved(0 Lines deleted) 0 boxes saved(0 boxes deleted)
Frame 1 : 1 points saved(0 pts deleted) 0 Lines saved(0 Lines deleted) 0 boxes saved(0 boxes deleted)
Frame 2 : 1 points saved(0 pts deleted) 0 Lines saved(0 Lines deleted) 0 boxes saved(0 boxes deleted)
Frame 8 : 1 points saved(0 pts deleted) 0 Lines saved(0 Lines deleted) 0 boxes saved(0 boxes deleted)
Frame 9 : 1 points saved(0 pts deleted) 0 Lines saved(0 Lines deleted) 0 boxes saved(0 boxes deleted)
Frame 10 : 1 points saved(0 pts deleted) 0 Lines saved(0 Lines deleted) 0 boxes saved(0 boxes deleted)
Frame#: 11, Buffered :20 (11.56%), LocXY: (815.10,179.40), Scale: 1.5 Objects: 26, points: 26, lines: 0, boxes: 0.
```

Figure A.16 Debug Window

Help:

F1 (or Fn+F1) Key: Use this key Displays/hide the status of current events such as number of Frame selected, tools choose etc.

F2 (or Fn+F2) Key : Use this key to Display/hide the information window with information such as current frame number, cursor location, number of objects etc

Zoom- in /Out: Use Mouse wheel to zoom in and out of the current frame.

Drag the mouse pointer to scroll through current frame

Accessing a particular frame: Use forward or previous arrow from FrameAdvance tool or enter a particular frame number.

Class Chooser Tool: Class chooser tool shows the various classes based on the dataset selected.

Save/Display Panel: This panel can be used to for various save or display options as below:

Save Now- Saves all the objects of the current frame to database

AutoSaveOff - Deactivates the AutoSave functionality while traversing through frames

Not Editable? - Deactivates all the save and display functionality. The changes to objects are not saved after

Attribute window: To see the attribute window of an object, click on the object once and double click it again. Attribute windows shows various attribute of a marked object such as propagated id, class, xy coordinates, classification and tracking attributes as below.

ObjectID: Unique identification number for object in a frame

ParentID: ObjectID of its parent object (same object in previous frame). -1 if object doesn't have any parent.

ChildrenID: ObjectID of its child object (same object in next frame). -1 if object doesn't have any child.

Root Label: Unique number assigned for each track

Lineage: Uses root label to draw trajectory for object track. Follows lineage representation in case object split

Auto Trajectory: Draws trajectory of an object on frame progress

Draw Trajectory: Draws trajectory of an object in specified start and end frames

Key 'c': Stop polyline drawing or join last point and starting point in case of polygon drawing

Figure A.17 FireFly help text

Appendix B

Limitations in Using the IE browser

FireFly uses various shortcut keys to increase the efficiency and reduce number of widgets to make user interface simple. Some of the shortcut keys in FireFly include F1, F2, Ctrl-t, Ctrl-x, Ctrl-j, Ctrl-s, c, etc. These shortcut keys are selected to be implemented since they convey the meaning of feature and therefore it would be easy for the user to remember. Running FireFly application in Internet Explorer browser has some limitations in using these short cut keys as some of them conflict with Internet Explorer's shortcut keys. Table shows the shortcut keys which conflict with the IE browser keys. However FireFly runs as expected in other browsers such as Firefox, Chrome, Safari, etc.

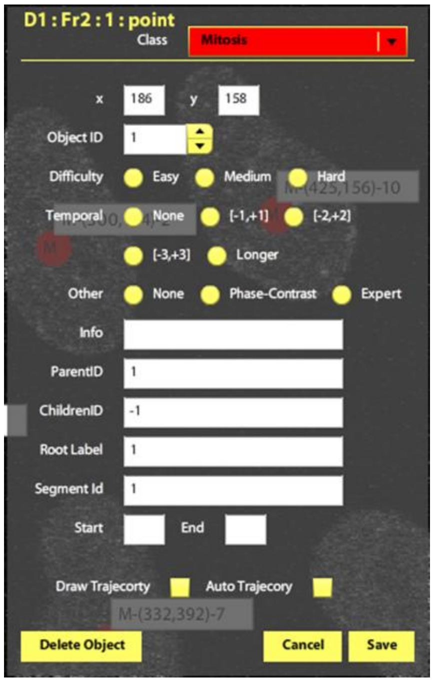
Key	FireFly	Internet Explorer
F1	Debug Window	Help and Support
Ctrl-t	Enable track mode	Opens new tab
Ctrl-j	Joins the object track	View downloads history

Table B.1 Key conflicts in IE

Appendix C

Evolution of the FireFly User Interface

They have been many changes made to the FireFly user interface to make the usage of FireFly simpler to the user. The two main widgets which has been completely changed from FireFly initial development are Attribute Window and Save Display Settings widgets. Figure shows the initial version of FireFly Attribute window where all classification and trajectory attributes are displayed in a single window.



D1 : Fr2 : 1 : point

Class: Mitosis

x: 186 y: 158

Object ID: 1

Difficulty: Easy Medium Hard

Temporal: None [-1,+1] [-2,+2] [-3,+3] Longer

Other: None Phase-Contrast Expert

Info:

ParentID: 1

ChildrenID: -1

Root Label: 1

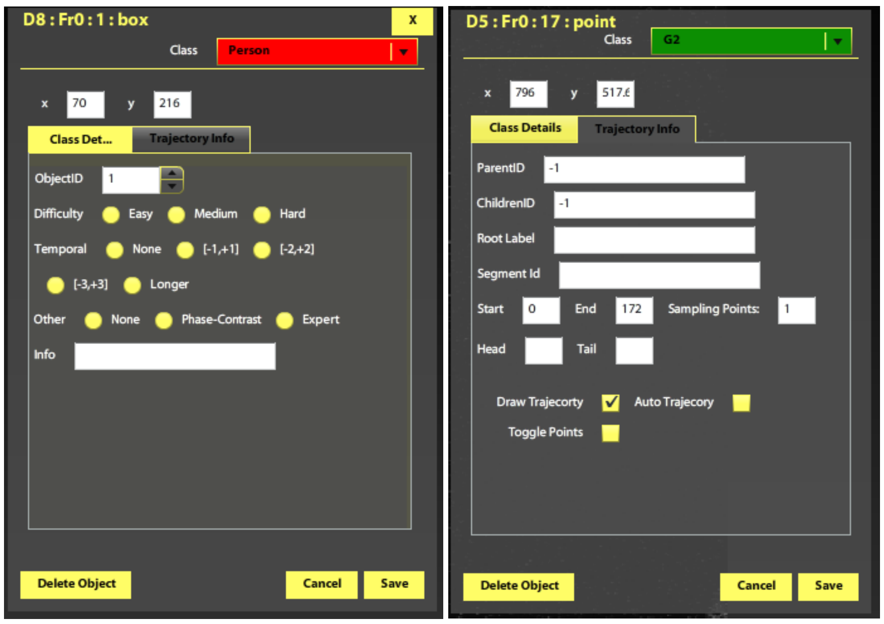
Segment Id: 1

Start: End:

Figure C.1 Attribute Window version 1

Figure shows the modified attribute window with tabbed window interface. Attribute window differentiate the classification and trajectory attributes in two tabs. Trajectory details tab consists of trajectory/ track attributes and other trajectory display settings such as

- Draw Trajectory to display trajectory between mentioned start and end frames.
- Auto trajectory displays the trajectory of an object from its start frame to the current frame.
- Toggle points will hide/displays the frame points within the trajectory
- Samplings points sample the frame points within the trajectory
- Head and Tail displays the trajectory number of frames mentioned in Head to the current frame and Tail specifies the number of frames after the current frame.



D8 : Fr0 : 1 : box

Class: Person

x: 70 y: 216

ObjectID: 1

Difficulty: Easy Medium Hard

Temporal: None [-1,+1] [-2,+2] [-3,+3] Longer

Other: None Phase-Contrast Expert

Info:

D5 : Fr0 : 17 : point

Class: G2

x: 796 y: 517.£

ParentID: -1

ChildrenID: -1

Root Label:

Segment Id:

Start: 0 End: 172 Sampling Points: 1

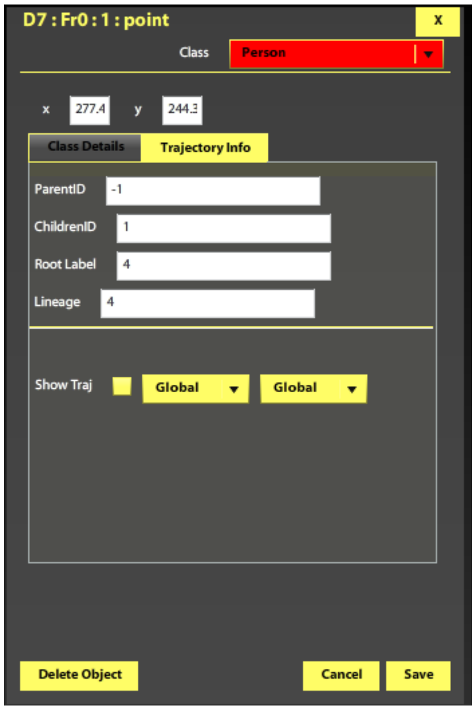
Head: Tail:

Draw Trajecorty [✓] Auto Trajecory [ ]

Toggle Points [ ]

Figure C.2 Attribute Window version 2 with tabbed window

Figure shows the current Attribute Window where trajectory display settings such as Head, Tail, Sampling points, Toggle points settings have been moved to the Save/Display setting widget and these settings are globally effected on all the trajectories. Auto trajectory is changed to Show trajectory setting and by default the start and end frame of trajectory is set to the global settings (in Save/Display settings) and any changes to the start/end frame will affect the trajectory display locally to that object trajectory.


D7 : Fr0 : 1 : point

Class: Person

x 277.4 y 244.3

ParentID -1

ChildrenID 1

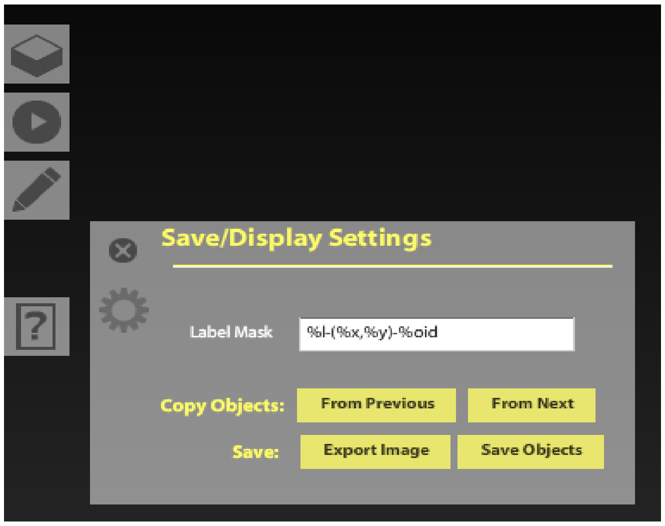
Root Label 4

Lineage 4

Show Traj Global Global

Figure C.3 Attribute Window version 3

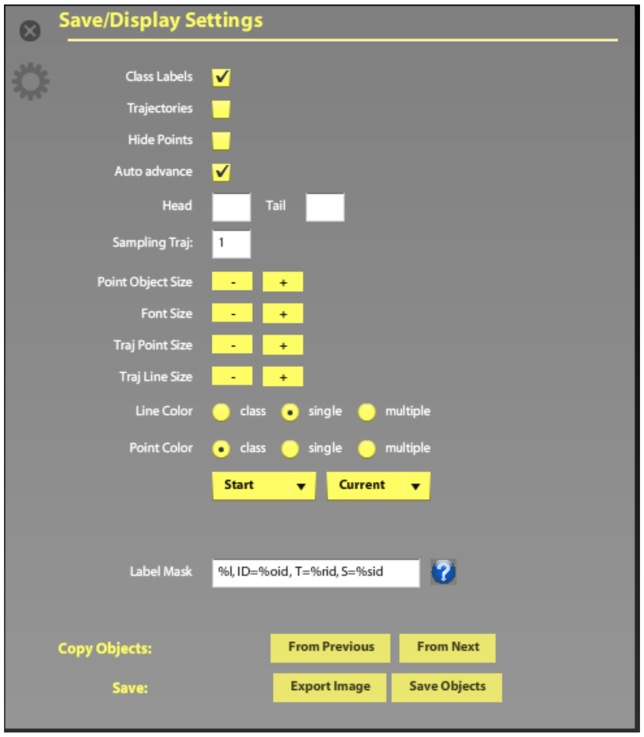
Figure shows the initial version of Save/Display settings, which consists of Label mask, Save objects, Copy objects settings.


Save/Display Settings

Label Mask: %l-(%x,%y)-%oid

Figure C.4 Save/Display settings version 1

Figure shows the current version of Save/Display widget with the additional following settings as shown in the table


Save/Display Settings

Class Labels ✓
Trajectories
Hide Points
Auto advance ✓
Head Tail
Sampling Traj: 1
Line Color: class [✓ single] multiple
Point Color: [✓ class] single multiple
Label Mask: %l, ID=%oid, T=%rid, S=%sid

Figure C.5 Save/Display settings version 2

Settings	Description
Class Labels	Hide the class labels for objects
Trajectories	Draw trajectories of all objects
Hide Points	Hide frames points in the trajectory
Auto advance	Auto advance settings for single object tracking
Head and Tail	Head and Tail settings for trajectory drawing
Sampling Points	Sample the frame points in trajectory drawing
Point Object Size	Increase/decrease the point object size
Font Size	Increase/decrease font size of text
Traj Point Size	Frame point increase/decrease in trajectory drawing
Traj Line Size	Increase/decrease the trajectory line size
Line Color	Trajectory line color to -class : same as class color -single: white color -multiple: each object is assigned with random color
Point Color	Trajectory points color to -class : same as class color -single: white color -multiple: each object is assigned with random color
Start and End Frames	Global start and end frames for trajectory drawing

Table C.1 Settings in Save/Display widget

Appendix D

FireFly Development Environment

FireFly uses Flex, Flash, MySQL and PHP technologies. Adobe Flex is a software development kit for the development of Rich Internet Applications. Flex applications can be built on Adobe Flash Builder software and it is available at https://www.adobe.com/cfusion/tdrc/index.cfm?product=flash_builder. Adobe Flash Builder is also available as a plug-in for Eclipse and Eclipse is available at http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/heliossr2. Flash Builder installation procedure on windows and as a plug-in on Eclipse is explained at http://kb2.adobe.com/cps/921/cpsid_92180.html. FireFly application is developed using Flex 3.5 SDK; therefore Flex 3.5 SDK must be downloaded (http://opensource.adobe.com/wiki/display/flexsdk/Downloads) and imported to the Flash Builder to compile FireFly application. Flex applications are compiled to .swf files which need Flash Player software installed on browser to run these files. Flash Player plug-in for browser can be downloaded at http://get.adobe.com/flashplayer/.

Development of Flex code

FireFly uses subversion to store the back up of code, this helps the new users to get required credentials and simply check out the latest code from the repository. Also, whenever the new feature is updated, user can check in the updated code to the subversion, which allows other development users to consistently update their localhost code base with the modified feature. Subversion installation procedure is explained at http://www.rose-hulman.edu/class/csse/resources/Subclipse/installation.htm. FireFly application is compiled and deployed on the server after testing the application on localhost. Some of the resources for quick learning Flex applications development are:

- http://livedocs.adobe.com/flex/3/
- http://blog.flexexamples.com
- http://cookbooks.adobe.com/flex

Debugging Tools

Debugging tools used for debugging Firefly application are

- Flash Debugger plug-in for web browsers will help the developers in prompting error/exception messages with the line numbers and also with trace of function calls.
- Flash Builder provides debug feature which helps the developer to run the applications with control break points. Debugging Flex applications is explained at http://help.adobe.com/en_US/flashbuilder/using/index.html.
- Charles Web Debugging is also an important tool which provides HTTP monitor/proxy that helps the developers to view all HTTP request and response made from their browser to the internet. Since the communication between FireFly application is in Action Message Format,

Charles helps in viewing the contents of Flash Remoting / Flex Remoting messages as a tree. (Available at http://www.charlesproxy.com).

PHP Development

PHP code is available on the meru server and can be accessible to one with required credentials. In order to connect to php you can connect through securefx or any ftp client, connect to host: meru.cs.missouri.edu with pawprint and password. To setup FireFly application on the localhost, developers must have to install a WampServer which is available at http://www.wampserver.com/en/. Latest version of PHP can be downloaded from server and copied to the WampServer software location on the localhost (C:/wamp/www). Service-config.xml file in the Flex code consists of the server path has to be modified to the localhost path.

Database

FireFly uses MySQL database for storing all data associated with different tables which is located on meru server and users must have required credentials to access the database. To setup the database on localhost, download and install MySQL software from http://dev.mysql.com/downloads/. FireFly database can be build from the .sql file located in the subversion. Necessary database authentication details are supposed to be changed in the application.ini file of PHP code.

Bibliography

1. D. Doermann and D. Mihalcik. Tools and techniques for video performance evaluation. In 15th int. conf. Pattern Recongnition, volume 4, pages 167-170. http://viper-toolkit.sourceforge.net, 2000.

2. Bryan C. Russell, Antonio Torralba. LabelMe: a database and web-based tool for image annotation. International Journal of Computer Vission, volume 77, pages 157-173. http://people.csail.mit.edu/brussell/research/AIM-2005-025-new.pdf, May 2008.

3. Beard, D. FireFly- web based interactive tool for the visualization and validation of Image Processing algorithms. Master's thesis, University of Missouri, 2009.

4. I. Ersoy, F. Bunyak, V. Chagin, C.M. Cardoso, and K. Palaniappan. Segmentation and classification of cell cycle phases in fluorescence imaging. Lecture Notes in Computer Science (MICCAI), 5762:617-624, Sep. 2009.

5. F. Bunyak, K. Palaniappan, V. Chagin, and C.M. Cardoso. Cell segmentation in time-lapse fluorescence microscopy with temporally varying sub-cellular fusion protein patterns. In Proc. IEEE Engineering in Medicine and Biology Society Conference (EMBC), pages 1-5, Minneapolis, MN, Sep. 2009.

6. Jenny Yuen, Bryan Russell, Ce Liu, Antonio. LabelMe video: Building a Video Database with Human Annotations. 2009 IEEE 12th International Conference On. 2009. 1451-1458.

7. A. Haridas, R. Pelapur, J. Fraser, F. Bunyak, K. Palaniappan, "Visualization of automated and manual trajectories in wide-area motion imagery", 15th Int. Conf. Information Visualization, London, 2011.

8. Allan R. Jones, Caroline C. Overly, Susan M. Sunkin. "The Allen Brain Atlas: 5 years and beyond". Nature Reviews Neurosciences, Vol. 10, No. 11. 2009

9. Christopher Lau, Lydia Ng, Carol Thompson, Sayan Pathak, Leonard Kuan, Allan Jones, Mike Hawrylycz. "Exploration and visualization of gene expression with neuroanatomy in the adult mouse brain". BMC Bioinformatics, Vol. 9. 2008.

10. Joshua Duhl (2003). IDC white paper on Rich Internet Applications [White paper]. http://lib.trinity.edu/research/citing/APAelectronicsources.pdf

11. Jeremy Allaire. Macromedia flash mx—a next-generation rich client. Technical report, March 2002.

12. Farrell, J.; Nezlek, G.S.; , "Rich Internet Applications The Next Stage of Application Development," Information Technology Interfaces, 2007. ITI 2007. 29th International Conference on , vol., no., pp.413-418, 25-28 June 2007.

13. Amitava Kundu, Charu Agarwal, Anushka Chandrababu, Mukul Kumar, Karthik Ramanarayanan, Raul F. Chong. Getting started with Adobe Flex. May 2010. IBM Corporation.

14. J. C. Preciado, M. Linaje, S. Comai, and F. Sanchez-Figueroa. Designing rich internet applications with web engineering methodologies. In WSE '07: Proceedings of the 2007 9th IEEE International Workshop on Web Site Evolution, pages 23–30, Washington, DC, USA, 2007. IEEE Computer Society

15. Grossman, Gary; Huang, Emmy. (2006). "ActionScript 3.0 overview". Adobe Systems Incorporated.

16. Lu Gao; Li-Yong Zhou; , "The improvement of RIA's software framework in Flex," Electronic and Mechanical Engineering and Information Technology (EMEIT), 2011 International Conference on , vol.8, no., pp.4351-4354, 12-14 Aug. 2011

17. Swiz framework. http://swizframework.jira.com. Retrieved 11/5/2011.

18. Wade Arnold. Zend Framework: Zend_Amf Component Proposal. January 2008.

19. Erik Meijering, Oleh Dzyubachyk, Ihor Smal, Wiggert A. van Cappellen. Tracking in cell and developmental biology. Seminars in Cell & Developmental Biology, Volume 20, Issue 8, October 2009

20. "Definition". Biology Online Dictionary. http://www.biology-online.org. Retrieved 11/10/2011.

21. Adobe Consultant team. Developing Flex RIAs with Cairngorm Micro architecture. [OL] http://www.adobe.com

22. Palaniappan, K.; Bunyak, F.; Kumar, P.; Ersoy, I.; Jaeger, S.; Ganguli, K.; Haridas, A.; Fraser, J.; Rao, R.M.; Seetharaman, G.; , "Efficient feature extraction and likelihood fusion for vehicle tracking in low frame rate airborne video," Information Fusion (FUSION), 2010 13th Conference on , vol., no., pp.1-8, 26-29 July 2010.

23. Jaeger, S.; Palaniappan, K.; Casas-Delucchi, C.S.; Cardoso, M.C.; , "Dual Channel Colocalization for Cell Cycle Analysis Using 3D Confocal Microscopy," Pattern Recognition (ICPR), 2010 20th International Conference on , vol., no., pp.2580-2583, 23-26 Aug. 2010.

24. F. Bunyak, A. Hafiane, K. Palaniappan, Histopathology tissue segmentation by combining fuzzy clustering with multiphase vector level sets. Software Tools and Algorithms for Biological Systems, Ed. H. R. Arabnia and Q.N. Tran, Advances in Experimental Medicine and Biology Series, Springer, Part V1, Chapter 41, pp. 413 – 424, 2011

25. Johnston, J.; Nagaraja, A.; Hochheiser, H.; Goldberg, U.; , "A flexible framework for Web interfaces to image databases: supporting user-defined ontologies and links to external databases," Biomedical Imaging: Nano to Macro, 2006. 3rd IEEE International Symposium on , vol., no., pp.1380-1383, 6-9 April 2006

26. Kristian Kvilekval, Dmitry Fedorov, Boguslaw Obara, Ambuj Singh and B.S. Manjunath, “Bisque: A Platform for Bioimage Analysis and Management” Bioinformatics, vol. 26, no. 4, pp. 544-552, Feb. 2010.