KOLAM: HUMAN COMPUTER INTERFACES FOR VISUAL ANALYTICS IN BIG DATA IMAGERY

A Thesis Dissertation presented to
the Faculty of the Graduate School
at the University of Missouri

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

---

by

ANOOP HARIDAS

Dr. Kannappan Palaniappan, Thesis Supervisor

MAY 2018

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

KOLAM: HUMAN COMPUTER INTERFACES FOR
VISUAL ANALYTICS IN BIG DATA IMAGERY

presented by Anoop Haridas,
a candidate for the degree of Doctor of Philosophy and hereby certify that, in their opinion, it is worthy of acceptance.

Dr. Kannappan Palaniappan

Dr. Jeffrey Uhlmann

Dr. Youssef Saab

Dr. David Larsen

ACKNOWLEDGMENTS

To my Mother and Father, for their love, wisdom and support;
To my wife Vidhu, for her selfless love and understanding;
To Aaroop, for being a tough, wise and loving brother;
To my Guru, for his priceless blessings, love and encouragement;
To Dr. Palaniappan, for his mentorship, advice and support;
To Drs. Uhlmann, Saab and Larsen; for their advice and support;
To Rengarajan, Surya and all my other friends and colleagues;
To the University of Missouri, for supporting me through the years;

Thank you all, for helping in making this thesis a reality.

TABLE OF CONTENTS

CHAPTER

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

List of Algorithms

ABSTRACT

In the present day, we are faced with a deluge of disparate and dynamic information from multiple heterogeneous sources. Among these are the big data imagery datasets that are rapidly being generated via mature acquisition methods in the geospatial, surveillance (specifically, Wide Area Motion Imagery or WAMI) and biomedical domains. The need to interactively visualize these imagery datasets by using multiple types of views (as needed) into the data is common to these domains. Furthermore, researchers in each domain have additional needs: users of WAMI datasets also need to interactively track objects of interest using algorithms of their choice, visualize the resulting object trajectories and interactively edit these results as needed. While software tools that fulfill each of these requirements individually are available and well-used at present, there is still a need for tools that can combine the desired aspects of visualization, human computer interaction (HCI), data analysis, data management, and (geo-)spatial and temporal data processing into a single flexible and extensible system.

KOLAM is an open, cross-platform, interoperable, scalable and extensible framework for visualization and analysis that we have developed to fulfil the above needs. The **novel contributions in this thesis** are the following: **1) Spatio-temporal caching** for animating both giga-pixel and Full Motion Video (FMV) imagery, **2) Human computer interfaces purposefully designed to accommodate big data visualization**, **3) Human-in-the-loop interactive video object tracking** - ground-truthing of moving objects in wide area imagery using algorithm assisted human-in-the-loop coupled tracking, **4) Coordinated visualization** using stacked layers, side-by-side layers/video sub-windows and embedded imagery, **5) Efficient one-click manual tracking, editing and data management** of trajecto-

ries, **6) Efficient labeling of image segmentation regions** and passing these results to desired modules, **7) Visualization of image processing results** generated by non-interactive operators using layers, **8)** Extension of interactive imagery and trajectory visualization to **multi-monitor wall display** environments, **9) Geospatial applications:** Providing rapid roam, zoom and hyper-jump spatial operations, interactive blending, colormap and histogram enhancement, spherical projection and terrain maps, **10) Biomedical applications:** Visualization and target tracking of cell motility in time-lapse cell imagery, collecting ground-truth from experts on whole-slide imagery (WSI) for developing histopathology analytic algorithms and computer-aided diagnosis for cancer grading, and easy-to-use tissue annotation features.

Chapter 1

Introduction

1.1 Summary of Novel Contributions

Before beginning, the author would like to enumerate the **novel contributions in this thesis**, which are the following:

1) Spatio-temporal caching for animating both giga-pixel and Full Motion Video (FMV) imagery,

2) Human computer interfaces purposefully designed to accommodate big data visualization,

3) Human-in-the-loop interactive video object tracking - ground-truthing of moving objects in wide area imagery using algorithm assisted human-in-the-loop coupled tracking,

4) Coordinated visualization using stacked layers, side-by-side layers/video sub-windows and embedded imagery,

5) Efficient one-click manual tracking, editing and data management of trajectories,

6) Efficient labeling of image segmentation regions and passing these results to desired modules,

7) Visualization of image processing results generated by non-interactive operators using layers,

8) Extension of interactive imagery and trajectory visualization to **multi-monitor wall display** environments,

9) Geospatial applications: Providing rapid roam, zoom and hyper-jump spatial operations, interactive blending, colormap and histogram enhancement, spherical projection and terrain maps,

10) Biomedical applications: Visualization and target tracking of cell motility in time-lapse cell imagery, collecting ground-truth from experts on whole-slide imagery (WSI) for developing histopathology analytic algorithms and computer-aided diagnosis for cancer grading, and easy-to-use tissue annotation features.

These novel contributions are described in a little more detail in Section 1.10, and are the topics of the remainder of this thesis.

1.2 Need for Visual Analytics

Data acquisition methods are rapidly maturing in the current time. The consequence of this is the deluge of disparate, dynamic and at times conflicting information from multiple sources that are ubiquitously heterogeneous. At the present time, the data collection rate far outstrips the decision-making rate based on the data thus collected. The danger of getting lost in data, which may be task-irrelevant, inappropriately processed or presented is ever increasing. Why? Primarily because while information extracted from the data deluge has enormous value, the raw data possesses no value on its own. A related important factor involves the continuing crucial role played by the human component - the currently irreplaceable cognitive capacity (comprising both the domain knowledge and instincts borne of creativity and expertise) of human experts is growing very slowly, and is being overwhelmed by the data deluge. Thus, the information overload problem that is currently pervasive across industry and academia continues to result in wastage of time and money, as well as lost scientific and industrial opportunities.

In a nutshell, the mission of visual analytics is to transparently (a) identify the agent (who or what) which defines the relevance of information for a given task, (b) demarcate what procedures are appropriate in a given complex decision-making process, (c) define means to present the extracted information in a decision/task-oriented manner; which in turn facilitates better understanding of the information by (d) reducing the cognitive burden placed on human experts to manageable levels. Visual analytics has been defined as 'the science of analytical reasoning facilitated by interactive user interfaces' [1]. Its genesis is linked to the increased integration of the human in the knowledge discovery and data mining components of information visualization through effective and efficient visualization techniques, interaction capabilities and knowledge transfer. It currently combines aspects of visualization, human computer interaction, data analysis, data management, (geo-)spatial and temporal data processing and statistics [2]. The latter point is particularly crucial. Researchers from all across the spectrum more often than not forget that: (a) Over time, based on prevailing needs, processes will be put to the test under circumstances which are entirely different from those they were originally intended for, (b) Despite the continuing creation of increasingly powerful automated data analysis tools, completely and efficiently analyzing and understanding the generated results at future times remains an unsolved problem (due to said automated tools needing well-defined and understood problems to be maximally effective), and (c) The fully automated tools, while being representative of their creators' knowledge, lack the ability to effectively communicate said knowledge, especially beyond the boundaries of domain expertise.

1.3 Visual Analytics

Continuing our description, visual analytics allows for the visualization of the tasks of transparently processing data and information, thereby providing the means to examine the actual processes as well as the results. By enabling the rapid constructive evaluation, correction and improvement of expert-designed processes and models, visual analytics can accelerate the improvement of knowledge and decisions derived from the data deluge / information overload. This is achieved in the most productive manner by having humans and machines work cooperatively using their respective, distinct capabilities and strengths in tandem within a semi-automated analytical process, for which visualization serves as a medium. In this manner, visual analytics enables its users to tackle massive, dynamic and at times ambiguous data; and derive information and insight from it. It also enables the detection of the expected and the discovery of the unexpected: by allowing decision makers to focus their full cognitive and perceptual capabilities on the analytical process, while allowing them to apply advanced computational capabilities to augment the discovery process [3]. Most importantly, it communicates the defensible and understandable assessments it generates in an actionable manner [2].

Visual analytics can also be explained in terms of the overlapping problem domains tackled by automatic analysis methods and visual analysis methods [4]. *Automatic analysis methods* are the best candidates for the problem when (i) means for measurement and comparison of candidate solution quality are readily available, and (ii) when the analysis solution does not need to be dynamically adaptive. However, they fall short when they get trapped in local optima, which might be completely unrelated to a globally optimum solution. On the other hand, *visual analysis methods* leverage human domain knowledge, creativity, intuition and decision making to solve problems efficiently by steering the analysis process, in a manner that either cannot be automated or will serve as input for future development of automatic methods [3]. They also work well for problems that are not well defined and for which the relationship between algorithm input and output is unclear. However, they fall short when the data for solving the problem is too large in scope to be fully captured by a human analyst. Additionally, involving human users is cost-intensive and introduces unpredictability. Visual analytics aims to combine the best aspects of both analysis types. In doing so, visual analytics attains several unique reasoning techniques that enable rapid and insightful knowledge discovery; these include relationship discovery, whole-part relationship, combination of confirmatory and exploratory interaction, multiple data type support, temporal, spatial and linked views, outlier identification and groupings, labeling and analytical reporting [2].

When utilized and prioritized correctly, visual analytics methodologies can serve as a highly interdisciplinary collaboratory for visualization, data mining, data fusion and data management, machine learning and statistics research. Domain experts are becoming increasingly aware of the worth and usefulness of using visualization to interpret and communicate their information extraction activities and results. Unfortunately however, ad hoc solutions which do not utilize state-of-the-art visualization capabilities, which do not completely address the complexity of the problem at hand while also being incapable of handling usage scenarios beyond the rigid confines of the original problem description; continue to be used. Given the disparate disciplines they connect together, visual analytics methodologies become even more essential in that they aid experts from different domains communicate their findings amongst themselves effectively and productively. This is made possible through the creation of highly effective and novel visual analytics tools.

1.4 Visual Analytics: Building blocks and Challenges

The building blocks of a visual analytics system come from multiple disciplines, as illustrated in Figure 1.1.

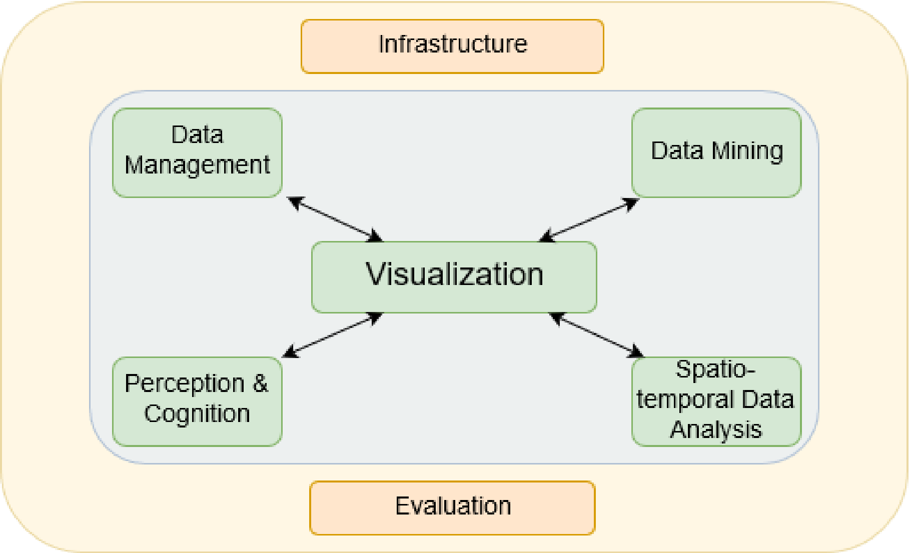


Figure 1.1: Building blocks of a visual analytics system.

The heart of the system is the visualization component, which can *communicate* the data values and results of some analyses, as well as *visually monitor* processes in the other constituent disciplines. The building blocks are briefly summarized below, and will be explored in detail in the context of KOLAM throughout the rest of this thesis.

Visualization: Visualization research is classified into scientific visualization and information visualization. *Scientific visualization* involves visualizing 2D/3D data and representing the same as surfaces, volumes, flows etc. Research has been focused on improving visualization efficiency to better support interactive exploration, a continuing problem due to rapidly increasing data sizes. Focus has also been placed on techniques to automatically derive the relevant visualization parameters both to aid the prior task as well as for user usability effectiveness and efficiency. In the same vein, focus and context based interaction techniques are also gaining importance. *Information visualization* is currently focused on the development of visualization techniques for huge volumes of high-dimensional abstract data without explicit spatial references. Besides the high dimensionality, several of the individual dimensions may be complex data types which do not trivially lend to traditional 2D/3D, plot and chart representations. Novel visualization techniques such as treemaps, parallel coordinates and glyph/pixel based visual representations have been developed to handle such data while also facilitating user interactivity and reducing display clutter. Preprocessing techniques such as clustering methods or dimensional reduction may also be applied to such data prior to visualization. Human Computer Interaction (HCI) forms a vital component of the visualization system. The foundations of HCI principles are outlined in Ben Shneiderman's seminal work [5], and have a huge body of other associated literature.

Data management: There is a need for database systems capable of handling heterogeneous data, operations such as data cleansing, disparate data type and format integration, intelligent data fusion, etc. The following challenges to the integrated exploration and analysis of data need to considered at the outset: (a) Data source heterogeneity, (b) Multiple data types, (c) Data streams, (d) Constraint-driven operation, and (e) Time consuming minutiae. Classical data management effectively deals with large data by utilizing (a) the relational model for data integration and exchange, and (b) the Structured Query Language (SQL) which has been highly optimized and standardized for data access. Despite the extensive research advancements in both fields, the integration between visual analytics and data management faces the following challenges: (a) Dynamic Responsiveness, (b) Relative lack of data handling standards in visual analytics, and (c) Longer data interaction life-cycle for visual analytics. The challenges faced by visual analytics and data management when the two are used together are: (a) Data Uncertainty, (b) Data Integration, (c) Semantics Management, (d) Data and Result Integrity, (e) Time-Consuming Low level Activities, (f) Interactive Large Database Visualization, and (g) Distribution and Collaboration in Visual Analytics. These challenges are explored in detail at relevant points later on in this thesis.

Data mining: This area encompasses the techniques utilizing automatic analysis algorithms in order to extract the information of interest from (pre-processed) data. Said techniques include: (a) *Supervised learning* - based on training data samples, deterministic or probabilistic algorithms are employed to learn models for the classification/prediction of unseen/real-world data, (b) *Unsupervised learning* - techniques including cluster analysis, which extract structure from the data (using mutual similarity and outlier identification measures) in the absence of a-priori knowledge, (c) *Association rule mining* - analysis of the co-occurrence of data items, and (d) *Dimensionality reduction*. Common to all approaches is the non-trivial task of specifying a minimum number of guiding parameters: a non-trivial task necessitating the inclusion of a human expert operator. Thus, it is important to develop UIs needed for this task.

Spatio-temporal analysis: The analysis of data with references in both space and time, this encompasses the characteristics of both spatial analysis (efficient representation, analysis and management of data via data structures, distance and similarity measure design) and temporal analysis (identifying time-varying data patterns and correlations) while introducing unique challenges. These are: (a) Scale: patterns and features of interest need to be searched for across multiple spatial and temporal scales, (b) Uncertainty: Spatio-temporal data is often incomplete, interpolated, collected at different points in time and with different underlying assumptions, and (c) Complex topological relationships between different objects in space.

Perception and Cognition: This component comprises the decades of research done in the fields of psychology, cognitive science and neuro-science, regarding the workings of the human visual system in concert with the brain and how humans consequently visually perceive and interpret information, in particular on the computer screen. Knowledge of how humans think visually is crucial in UI design; taken together with HCI practices, it can guide the design and implementation of perception-driven multi-modal interaction mechanisms for the exploratory visualization of huge data spaces. It can also be used for the design of usability evaluation methodologies of such systems.

Infrastructure: The concern here is the inter-linkage of all components of the visual analytics system in a harmonious manner such that users can efficiently and effectively operate the system. The challenges facing this objective are - (a) low level incompatibilities of the different software infrastructures, and (b) the high interactivity requirement. HCI system design principles [5] need to be taken into account with regards to the user interaction components of the infrastructure. Proper software design and software re-usability are key factors in application adaptability and short build times.

Evaluation: Evaluative methodologies are essential so that the efficiency, effectiveness and user acceptance of new visual analytics techniques and models with customized human computer interfaces may be assessed in a standardized manner for comparison and identification of potential problems. The evaluative process is a challenge however, due to - (a) The exploratory nature of visual analytics, (b) The wide range of user experience, (c) Data source diversity, and (d) The variance in the actual tasks. It is critical to create effective and reproducible evaluation techniques.

Per Cook and Thomas [6] there are 5 categories of *Grand Challenges* faced by visual analytics, which are as follows: (1) Analytical reasoning - This comprises the reasoning frameworks by which users derive insights or perform knowledge discovery, in order to support decision making. Said frameworks are fundamental to the application of specific transformations, visual techniques or other operations on the data. (2) Visual representations and Interaction techniques - Which comprise all human computer interactive (HCI) techniques and methods that facilitate visual data representation. (3) Data representations and transformations - These define specific means of data representation, as well as operations upon possibly noisy, incomplete or uncertain data. ’Representation’ refers to the fundamental (at times non-intuitive) structure of data which facilitates transformations. (4) Production, presentation and dissemination - this refers to User activity and interaction, ie. their relationship to data and the roles they play. (5) Moving research into practice - this refers to the practical applications of visual analytics methods and techniques.

1.5 The Visual Analytics Process

The visual analytics process (see Figure 1.2) tackles the problem of extracting knowledge from data by leveraging human interaction to tightly couple relevant automatic and visual analysis methods together.

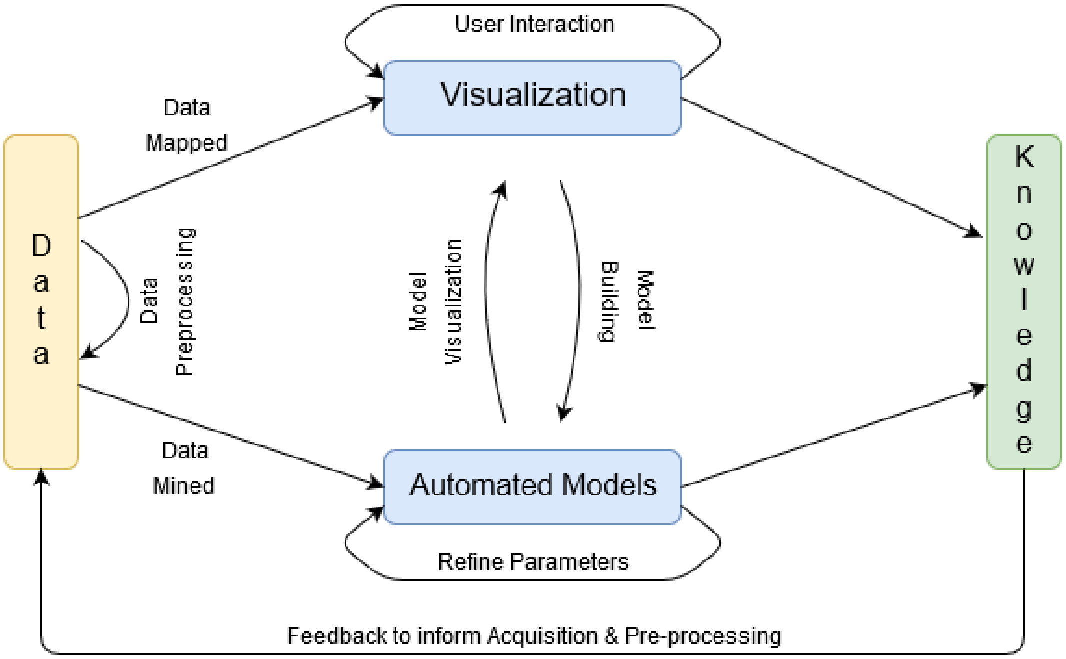


Figure 1.2: Operational stages of a visual analytics system.

The ***first step*** of the process involves data transformation and pre-processing in order to derive the different representations required for meaningful and efficient exploration. These tasks include data cleaning, normalization, grouping and integration techniques. For the ***second step***, there is a choice between applying either automatic or visual analysis methods first. In the *former* case, the data is appropriately mined in order to generate data models. Subsequent interaction (the ***third step*** in this case, with feedback) with the data via visualization (algorithm selection and/or parameter modification) is needed in order to evaluate and refine these models. In the *latter* case, visual analysis methods are used to generate the initial hypotheses, which automated analysis techniques are then used to confirm. Findings in the visualizations (either through zooming or by considering different data views) can yield insightful information capable of guiding model building (the ***third step*** in this case) in the automatic analysis phase. In *both cases*, iteratively alternating between both analysis methods leads to continuous result refinement and verification (the ***final step*** with feedback). Thus, the visual analytics process has ***three*** avenues for knowledge generation: *visualization*, *automated analysis* and the *interaction between visualizations, models and the human analysts*.

A formal guide to visually exploring data was first given by Shneiderman [7] - **"Overview first, zoom/filter, details on demand"**, which describes how data is to be presented on screen, and this represents one of the foundations of HCI techniques as they apply to visual data exploration. However, given the greater dimensional extents of current big data (ultra-high resolution, or gigapixel) datasets, more zooming out must be done in order to create an overview visualization of the whole data. This in turn results in loss of visually descriptive details, which in turn hinder the user from visually identifying regions of interest for further zooming in for analysis. In conclusion, merely retrieving and displaying the data becomes increasingly insufficient with increasing data size. Furthermore, increasing data dimensionality and complexity necessitate that automatic analysis methods be applied both before and after the use of the interactive visual representation of the data. The formal guide must thus be extended as follows: **"Analyze, highlight important regions, zoom/filter, analyze further, details on demand"** [3].

1.6 Motivation For Interactive Visual Analytics

It has been found that given a particular task, human operators and computer programs each have their unique strengths and weaknesses; this has been formalized by Shneiderman [5]. A summary of this is provided below. These opposing strengths and weaknesses strongly suggest that a system capable of combining the strengths and ameliorating the weaknesses, such as a carefully designed interactive visual analytics system, is highly desirable.

Human operators are generally better for the following situations: (i) Sensing low level stimuli, (ii) Detecting stimuli in noisy background, (iii) Recognizing constant patterns in varying situations, (iv) Sensing unusual and unexpected events, (v) Remembering principles and strategies, (vi) Retrieving pertinent details without an a-priori connection, (vii) Experience; adapting decisions to situation, (viii) Selecting alternatives if original approach fails, (ix) Reasoning inductively: generalize from observations, (x) Acting in unanticipated/novel situations, (xi) Applying principles to solve varied problems, (xii) Making subjective evaluations, (xiii) Developing new solutions, (xiv) Focusing on important tasks when overload occurs, and (xv) Adapting physical response to changes in situation.

On the other hand, machines/computer programs are generally better, when: (i) Sensing stimuli outside human range, (ii) Counting/measuring physical quantities, (iii) Storing quantities of coded information accurately, (iv) Monitoring (esp. infrequent) pre-specified events, (v) Rapidly, consistently responding to input signals, (vi) Recalling quantities of detailed information accurately, (vii) Processing quantitative data in pre-specified ways, (viii) Reasoning deductively: inferring from a general principle, (ix) Performing repetitive pre-programmed actions reliably, (x) Performing several activities simultaneously, (xi) Maintaining operations under heavy information load, and (xii) Sustained maintenance of performance.

1.7 Extreme-Scale Visual Analytics

Extreme-scale applications invariably combine high-performance computational hardware, high-performance database systems and/or cloud services (for data management and storage) and ubiquitous desktop hardware for human-computer interactivity. In addressing the top challenges of the day, these applications both fulfil critical scientific and technical requirements, but are also responsible for the dissemination of the solutions they create to the wider community. The authorities of VA research have broadly identified ten challenges which most prominently face the discipline and need solutions in the near term. They are listed below.

(1) In situ Analysis: This type of analysis implies that as much VA as possible be performed on the data while it still resides in memory. Performing analytics tasks in situ gains importance due to the fact that performing data analytics beyond the petascale boundary is expected to rapidly diminish the feasibility of the traditional approach of storing data on disk for later analysis. It should be noted that prioritizing in situ analytics also prioritizes the optimal design and implementation of interactive analytics, algorithms, memory, I/O, workflow and threading tasks. These tasks continue to be tackled by the HPC community at the current time.

(2) Interaction and UIs: The role of these factors continue to gain importance with the growing problem of extreme-scale data, due to the fact that human cognitive capabilities are remaining unchanged. The top challenges involving interaction and UIs are categorized as follows: (a) In situ interactive analysis - Efficiently and effectively sharing cores in the hardware execution units while ameliorating HCI resultant workflow disruptions; (b) User-driven data reduction - having user data collection practices and analytical needs drive the design of flexible data reduction mechanisms which will replace current compression

driven data reduction methods that are being rendered outdated at present; (c) Multilevel hierarchical solutions to scalability - while multilevel solutions address scalability issues at present, care needs to be given to the increasingly complex navigation of deepening hierarchies corresponding to increasing data size; (d) Evidence and uncertainty representation - Visualization is the crossroads where evidence synthesis and uncertainty quantification meet, and humans then interpret these in a VA environment. How to continue doing so in the face of still-growing extreme-scale data? (e) Heterogeneous-data fusion - Extracting the optimum amount of semantics from extreme-scale data and interactively fusing it for VA requires proper analysis of the interrelationships amonst heterogeneous data entities; (f) Data summarization for interactive query - The increasing size of extreme-scale data is rendering analysis of the entire data volume increasingly impractical and/or unnecessary. Data summarization allows for data with specific features to be requested, and I/O components must work well with the results of the same in order to enable interactive extreme-scale data queries; (g) Analytics of temporally evolved features - Developing effective, computationally practical VA techniques that exploit human cognitive abilities to track data dynamics is the key challenge when dealing with extreme-scale time-varying data; (h) The human bottleneck - Ways need to be found to circumvent the weakness of unchanging human cognitive capacity in the face of improving performance on all other fronts; (i) Design and Engineering - Dissemination of standardized API and framework support for UI and interaction development is needed; (j) Renaissance of wisdom in handling extreme-scale VA problems.

(3) Visualization of Big-Data: This challenge comprises visualization techniques and information display aspects of data presentation in VA. Abstract visualization, scalable data projection, data dimensionality reduction, increasing display resolution and multi-monitor

wall displays are different approaches that have been developed to address the aforementioned challenge. However, the shortcomings of each of these approaches is becoming increasingly apparent, implying that this challenge will remain unsolved for the foreseeable future.

(4) Databases and Storage: Cloud services are currently capable of serving the exabyte data demands of extreme-scale visual analytics database and storage implementations. However, there exist a number of hardware and software concerns regarding cloud services, which call into question their ability to scale with even greater data size demands. These concerns are: (a) The unfavorable cost of cloud unit storage versus traditional secondary storage unit cost, (b) That cloud network bandwidth still affect cloud database latency and throughput unfavorably, and (c) failure of all cloud systems to fully support the ACID (Atomicity, Consistency, Isolation, Durability) requirements of distributed databases.

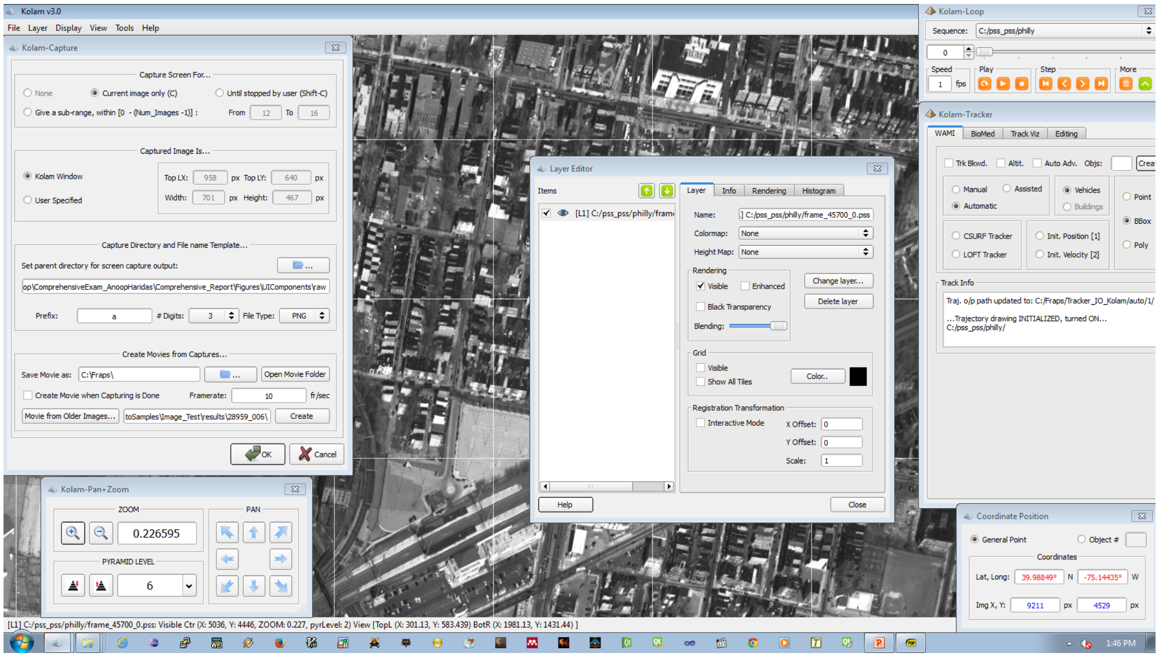
(5) Algorithms: Traditional VA algorithms haven't been designed with scalability as a design consideration. Therefore, such algorithms are either too computationally expensive, produce sufficiently human friendly output or assume that all required data will be readily available in memory or local secondary storage. The best algorithms to be developed moving forward must address data-size and visual efficiency concerns, and must incorporate novel visual representations and user interactability. Most importantly, highly adaptable visualization output must be realized by creating novel HCI interfaces which integrate user preferences with automatic learning in a versatile manner.

(6) Data Transport and Network infrastructure: Data movement is quickly becoming the most expensive component in the VA pipeline, given the still falling costs of computing power. Adding to the problem is the ever increasing demand for demand for data movement driven by increasing geographic dispersion of ever-expanding data stores.

(7) Quantification of Uncertainty: Awareness of the source of uncertainty in the data is crucial for decision making and risk analysis tasks. This fact is underscored in the present day, when the ever increasing size of data implies that the uncertainty that needs to be dealt with is also increasing; due to the fact that data subsampling must be increased in order to satisfy the real time constraint when dealing with growing data sizes. Uncertainty visualization and analysis will both have ever increasingly important roles to play in the future; therefore, the development of algorithms to tackle incomplete data as well as treat it as distributions must become a priority. Visualization has a key role to play by providing the most intuitive possible views of the ever increasing uncertainty, thereby enabling risk comprehension and minimizing misleading result generation via proper parameter selection.

(8) Parallelism: Parallel processing is effective in enabling interactive analytics by virtue of reducing the turnaround visual computing time. In order to fully exploit pervasive parallelism, VA algorithms need to be designed with data models that take the shrinking memory footprint per computing core into account. They also need to incorporate hybrid parallelism models and perform a majority of their data-intensive operations out-of-core.

(9) Tools and User Involvement: Affordable libraries and tool frameworks are key to rapid development of HPC VA applications. Finally, extreme-scale VA research and development can only be fostered by constructive involvement from civilian government and commercial vendor entities worldwide. These entities need to proactively influence hardware manufacturers, academia for technology research, development and dissemination of these achievements in usable forms to society at large.



Kolam-Capture:

○ None ● Current image only (C) ○ Until stopped by user (Shift-C)

○ Give a sub-range, within [0 - (Num_Images -1)] : From 12 To 16

● Kolam Window ○ User Specified

Top LX: 958 px Top LY: 640 px

Width: 701 px Height: 467 px

Set parent directory for screen capture output: op\ComprehensiveExam_AnoopHaridas\Comprehensive_Report\Figures\UIComponents\raw

Prefix: a # Digits: 3 File Type: PNG

Save Movie as: C:\Fraps\

□ Create Movie when Capturing is Done Framerate: 10 fr/sec

toSamples\Image_Test\results\28959_006\

Kolam-Pan+Zoom: ZOOM 0.226595; PYRAMID LEVEL 6

Layer Editor:

☑ [L1] C:/pss_pss/philly/fram

Name:] C:/pss_pss/philly/frame_45700_0.pss

Colormap: None

Height Map: None

Rendering: ☑ Visible □ Enhanced □ Black Transparency Blending:

Grid: □ Visible □ Show All Tiles

Registration Transformation: □ Interactive Mode X Offset: 0 Y Offset: 0 Scale: 1

Kolam-Loop:

Sequence: C:/pss_pss/philly

0

Speed 1 fps

Kolam-Tracker:

□ Trk Bkwd. □ Altit. □ Auto Adv. Objs:

○ Manual ○ Assisted ● Automatic

● Vehicles ○ Buildings

○ Point ● BBox ○ Poly

○ CSURF Tracker ○ LOFT Tracker

○ Init. Position [1] ○ Init. Velocity [2]

Track Info:

Traj. o/p path updated to: C:/Fraps/Tracker_IO_Kolam/auto/1/

...Trajectory drawing INITIALIZED, turned ON...
C:/pss_pss/philly/

Coordinate Position:

● General Point ○ Object #

Lat, Long: 39.98849° N -75.14435° W

Img X, Y: 9211 px 4529 px

[L1] C:/pss_pss/philly/frame_45700_0.pss: Visible Ctr (X: 5036, Y: 4446, ZOOM: 0.227, pyrLevel: 2) View [TopL (X: 301.13, Y: 583.439) BotR (X: 1981.13, Y: 1431.44)]

Figure 1.3: An interactive visual analytics example in KOLAM, with a time-lapse image dataset loaded and several pertinent tool interfaces for manipulating the view of the data or obtaining certain types of results from the data being displayed.

1.8 Introduction to KOLAM

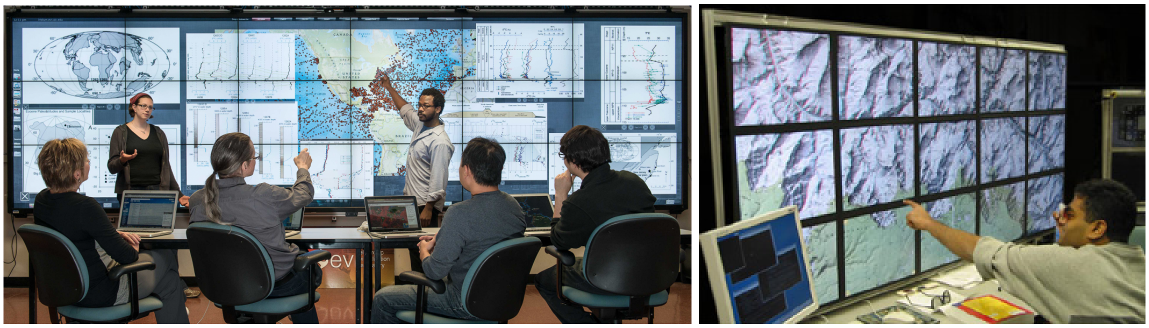
Exploratory data analysis involves the search and analysis of databases to find implicit yet potentially useful information [3]. Unlike *confirmatory analysis*, exploratory analysis provides no hypotheses about the data as initial starting points to work from to the analyst, thereby making it a difficult task at the onset [3]. The implication is that versatile tools and understanding of domain knowledge are required in order to perform interactive and (usually) undirected searches for structures and trends in the data [8]. In this context, ***exploratory visualization and analytics tools*** are powerful methods to support multiple data-streams and navigation through very large datasets. Such tools offer ways to mitigate the data deluge being faced by users and analysts and are useful for providing insight into complex patterns in scientific, geospatial, satellite, and surveillance applications. ***KOLAM*** (K-tiles for Optimized muLtiresolution Access with coMpression, Figure 1.3) [9] is a scalable and extensible framework for high-resolution, high throughput image data visualization with applications in a variety of image analysis domains including WAMI [10]. It is platform and operating system independent and supports embedded datasets scalable from hundreds of gigabytes to petabytes in size on architectures ranging from clusters to netbooks. KOLAM uses a scalable data structure and cache management strategies to support large datasets along with an extensive set of image processing and analysis features. Novel Human-Computer Interfaces are realized in the form of robust animation and novel tracking interfaces, which enable smooth animation of WAMI sequences and support *one-click* per frame tracking of objects. KOLAM is capable of interfacing with different tracking algorithms and visualizing the resulting trajectories that may be composed of multiple segments. A feature-rich tracking interface enables simultaneous visualization of multiple tracks, context-sensitive track operations, track archival and retrieval, moving object ground truth generation, track editing and annotation. The need for user-driven models in the analysis of WAMI data to address challenging or unsolved problems in WAMI exploitation include: automated search tools, a complementary visual analytics tool for analysis, human-computer interaction that is capable of capturing user domain knowledge to improve productivity and search tools applied to multitarget tracking results.

Furthermore, KOLAM has been extended to support rapid interactive labeling and correction of automatic image classifier-based region labels of tissue micro-environment Whole-Slide Imagery (WSIs) by pathologists. KOLAM's extensible nature made visualization of big-data histopathology imagery a straightforward task that involved few changes to the existing architecture besides adding support for the requisite image file formats. Besides annotating regions-of-interest (ROIs), KOLAM enables extraction of the corresponding large polygonal image sub-regions for input into automatic segmentation algorithms, single-click region label re-assignment and maintaining hierarchical image sub-regions. Experience indicates that clinicians prefer simple-to-use interfaces that support rapid labeling of large image regions with minimal effort. The incorporation of easy-to-use tissue annotation features in KOLAM makes it an attractive candidate for integration within a **multi-stage histopathology image analysis pipeline** supporting assisted segmentation and labeling to improve whole-slide imagery (WSI) analytics.

1.9 Related Work and Success Stories

1.9.1 Visualization

Gigapixel-sized imagery have become much more prevalent over the past decade in a variety of domains including medicine, space, satellite and defense. Consequently, the development of tools for gigapixel-sized image visualization has been an active area of research. A review of the most prominent literature allows for the rough classification of gigapixel visualization into single display-oriented and multiple display-oriented systems. As an important example of a single display-based system, Kopf et al. created a system for creating, processing and displaying ultra-high resolution images with high dynamic range and wide angle fields of view [11]. Means for systematically annotating and rendering both auditory and visual annotations in gigapixel imagery have also been devised [12]. Additionally, work has been done towards rapid post-processing of gigapixel imagery utilizing an efficient parallel programming methodology [13].



(a) (b)

Figure 1.4: Multi-display visualization system examples: (a) SAGE and (b) JuxtaView.

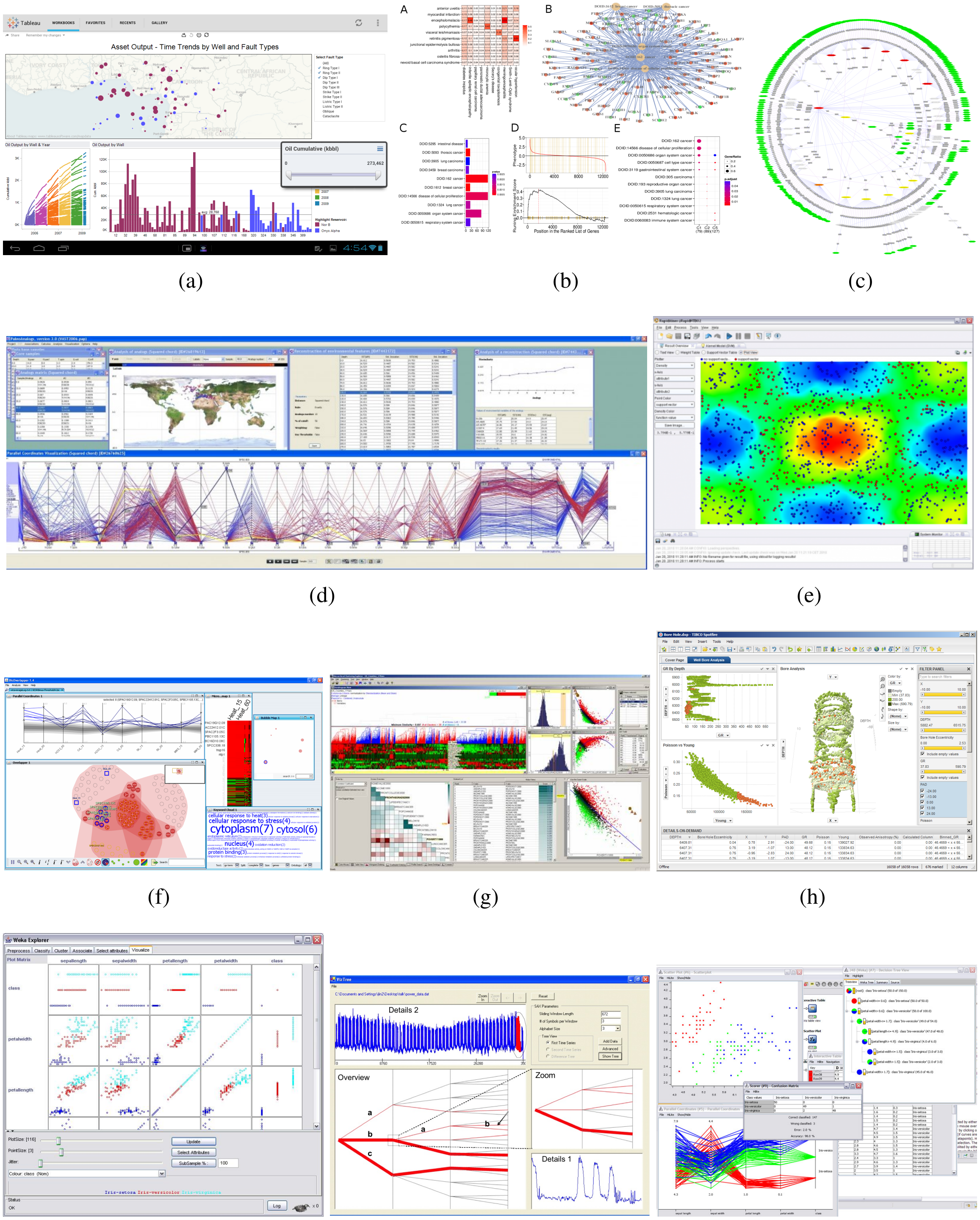
Based on the ability of the human visual system to rapidly identify patterns and discern differences, especially when the targeted data is extremely large, extensive research has been performed regarding the visualization of high-resolution image planes on tiled displays [14–24]. Architectures such as SAGE [16, 20, 25] (See Figure 1.4) and DIGI-Vis [24]; and applications such as JuxtaView [17] (See Figure 1.4) and Giga-Stack [23] seamlessly display a number of visualization applications over the entire tiled display. The primary goal of such systems is to support application heterogeneity and scalability by decoupling the graphics rendering and display processes from each other, and by utilizing high-bandwidth lambda networks to bridge them. Such ultra large-scale visualization initiatives are currently being used with infrastructures such as the OptiPlanet Collaboratory [22]. This persistent, distributed visualization cyberinfrastructure connects a series of OptIPortals [21,22], which are tiled displays that comprise the visual interfaces to OptIPuters [14,21]. These global-scale computing systems seek to create a synergistic combination between high-speed lambda networking, remote storage containing extremely large scientific datasets, distributed processing, data mining and visualization.

1.9.2 Visualization and KDD Methods

There are currently a good number of large and mature software systems that represent a successful marriage of visualization and Knowledge Discovery and Data-Mining (KDD) methodologies into highly effective visual analytics systems. A selection of more prominent examples is now provided.

In the field of *BioInformatics*, a prominent example is BioConductor [26], which provides tools for the analysis and comprehension of high-throughput genomic data. A successful integration of visualization and KDD methods in the field of *Climate Change* is the Paleoanalogs system [27], which can illustrate distribution of micro-fossil species across the earth for the past several million years, as well as reconstruct the environmental features of this entire time period. In the *Manufacturing and Process* industry, systems such as Spotfire [28] help make sense of quality parameters, process trends and maintenance events; and also perform anomaly detection and cause analysis. In the context of visualization and KDD, popular *statistical and mathematical* tools include R, Matlab, Mathematica [29] and SAS [30]. In the same context, GraphViz [31]; and KNIME [32], Weka [33] and RapidMiner [34] are examples of *algorithmic* tools and *visual data mining* tools. These tools do not explicitly combine the visualization and KDD aspects; however, certain tools do perform this combination. Examples include VizTree [35], Hierarchical Clustering Explorer [36] and BicOverlapper [37]. Figure 1.5 is a collective illustration of several of these tools in action.

To summarize the achievements in this thesis at a glance, a matrix comparing the problems, challenges and solutions implemented within KOLAM; to the capabilities of popular visual analytics systems, is illustrated in Figure 1.6.



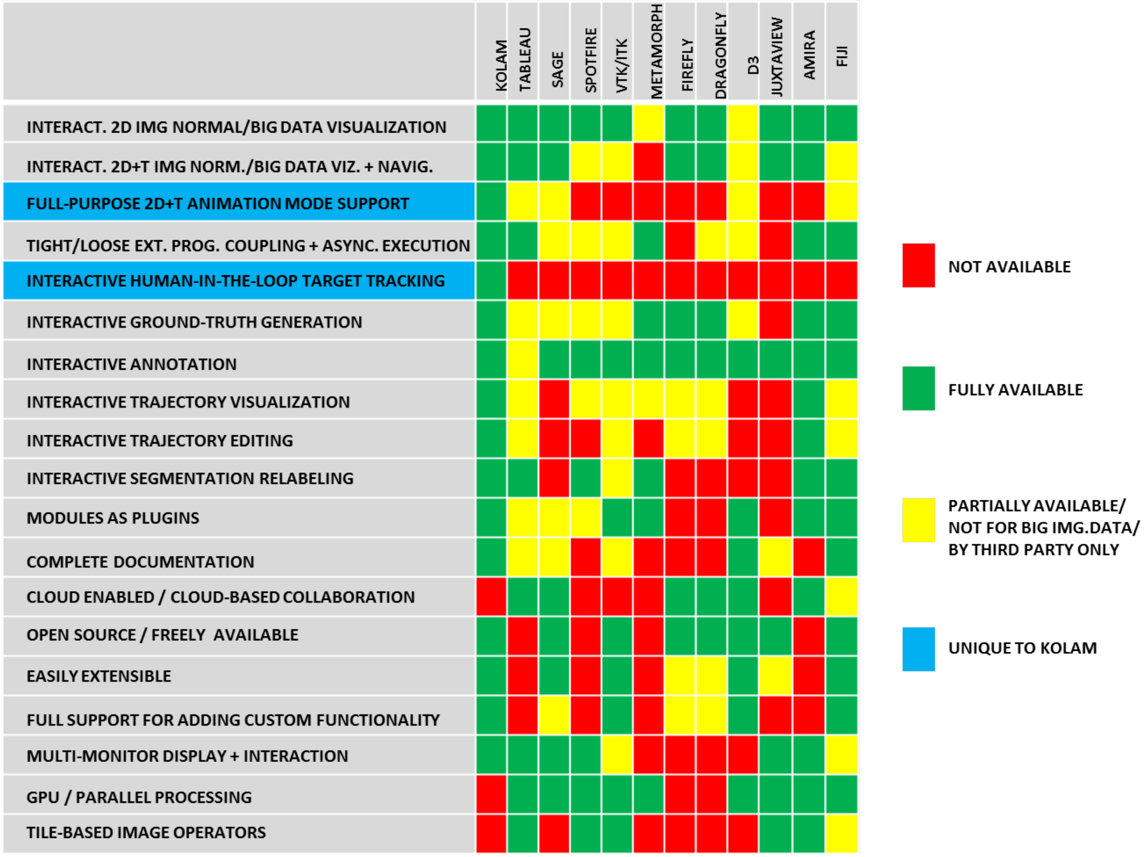
(a) (b) (c)

(d) (e)

(f) (g) (h)

(i) (j) (k)

Figure 1.5: Visual Analytics/KDD Systems: (a) Tableau, (b) Bioconductor, (c) Graphviz, (d) PaleoAnalogs, (e) RapidMiner, (f) Bicoverlapper, (g) HCE, (h) TIBCO Spotfire, (i) Weka, (j) Viztree and (k) KNIME.



	KOLAM	TABLEAU	SAGE	SPOTFIRE	VTK/ITK	METAMORPH	FIREFLY	DRAGONFLY	D3	JUXTAVIEW	AMIRA	FIJI
INTERACT. 2D IMG NORMAL/BIG DATA VISUALIZATION	🟩	🟩	🟩	🟩	🟩	🟨	🟩	🟩	🟨	🟩	🟩	🟩
INTERACT. 2D+T IMG NORM./BIG DATA VIZ. + NAVIG.	🟩	🟩	🟩	🟨	🟨	🟥	🟩	🟩	🟨	🟩	🟩	🟨
🟦 FULL-PURPOSE 2D+T ANIMATION MODE SUPPORT	🟩	🟨	🟨	🟥	🟥	🟥	🟥	🟥	🟨	🟥	🟥	🟨
TIGHT/LOOSE EXT. PROG. COUPLING + ASYNC. EXECUTION	🟩	🟩	🟨	🟨	🟨	🟩	🟥	🟨	🟨	🟥	🟩	🟩
🟦 INTERACTIVE HUMAN-IN-THE-LOOP TARGET TRACKING	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
INTERACTIVE GROUND-TRUTH GENERATION	🟩	🟨	🟨	🟨	🟨	🟩	🟩	🟩	🟨	🟥	🟩	🟩
INTERACTIVE ANNOTATION	🟩	🟨	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
INTERACTIVE TRAJECTORY VISUALIZATION	🟩	🟨	🟥	🟨	🟨	🟨	🟨	🟨	🟥	🟥	🟩	🟨
INTERACTIVE TRAJECTORY EDITING	🟩	🟨	🟥	🟥	🟨	🟥	🟨	🟨	🟥	🟥	🟩	🟨
INTERACTIVE SEGMENTATION RELABELING	🟩	🟩	🟥	🟩	🟨	🟩	🟥	🟥	🟥	🟥	🟩	🟩
MODULES AS PLUGINS	🟩	🟨	🟨	🟨	🟩	🟩	🟥	🟥	🟩	🟥	🟩	🟩
COMPLETE DOCUMENTATION	🟩	🟨	🟨	🟥	🟨	🟥	🟥	🟥	🟩	🟨	🟥	🟩
CLOUD ENABLED / CLOUD-BASED COLLABORATION	🟥	🟩	🟩	🟥	🟥	🟥	🟩	🟩	🟩	🟥	🟩	🟨
OPEN SOURCE / FREELY AVAILABLE	🟩	🟥	🟩	🟥	🟩	🟥	🟩	🟩	🟩	🟩	🟥	🟩
EASILY EXTENSIBLE	🟩	🟥	🟩	🟥	🟩	🟥	🟨	🟨	🟩	🟨	🟥	🟩
FULL SUPPORT FOR ADDING CUSTOM FUNCTIONALITY	🟩	🟥	🟨	🟥	🟩	🟥	🟨	🟨	🟩	🟥	🟥	🟩
MULTI-MONITOR DISPLAY + INTERACTION	🟩	🟩	🟩	🟩	🟨	🟥	🟥	🟥	🟥	🟩	🟩	🟨
GPU / PARALLEL PROCESSING	🟥	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟩	🟩	🟩	🟩
TILE-BASED IMAGE OPERATORS	🟥	🟩	🟥	🟩	🟩	🟥	🟥	🟥	🟥	🟩	🟩	🟨

🟥 NOT AVAILABLE

🟩 FULLY AVAILABLE

🟨 PARTIALLY AVAILABLE/
NOT FOR BIG IMG.DATA/
BY THIRD PARTY ONLY

🟦 UNIQUE TO KOLAM

Figure 1.6: A Functionality Comparison Matrix of KOLAM versus popular Visual Analytics Systems.

1.10 Novel Contributions in KOLAM

Following is the list of novel contributions made by the author in KOLAM towards this thesis. The remainder of the thesis tackles these topics in detail.

1) **Animation interface**: Design and implementation of an (HCI customized for big data visualization) player interface for image sequences and video files: Unlike ordinary full-fledged video players which expose primitive functions, namely Play, Pause, Stop etc. directly and conceal all other features in menu hierarchies, KOLAM gives direct access to these ubiquitous sequence controls as well as data navigation controls (Step Forward,

Step Backward, Jump to Start, Jump to End), playlist handling, jump to arbitrary frame, playback speed, variable frame step and sequence size; in one compact tool. Tool size was minimized as well as made a discrete entity in order to a) minimize screen space taken from actual display and b) allow to be moved with other tools to another display, if multiple displays are available.

2) **Spatio-temporal dual caching mechanism**: For handling multiresolution image sequences; KOLAM first caches temporally using a small buffer and out-of-core memory management, then caches spatially using most-recently-viewed-by-user distance as Manhattan distance, in order to exploit user behavior to optimize data access. This mode of time-lapse huge imagery sequence handling fits perfectly with the spatio-temporal data handling techniques of visual analytics: when considering multi-resolution spatial data handling and temporal data handling individually, each has its own unique challenges; and spatio-temporal big data handling has both sets of challenges and brings unique challenges of its own (ref. section 2.3.3). KOLAMs caching mechanism aims to tackle these challenges in an elegant and transparent manner (ref. section 2.3.4).

3) **WAMI Applications**: Driven by WAMI data needs, customized HCI interfaces for interactive one-click multi-target tracking, versatile multi-object ground-truthing, one-click tracker start and stop , assisted tracking by tight integration of visualization, tracker library and expert analyst interaction, all together for a highly functional visual analytics system for object tracking; and presentation of pertinent metadata like latitude-longitude information.

4) Multiple **trajectory** dataset management, configurable trajectory visualization, trajectory storage formats i.e. unstructured and semi-structured file formats for efficient trajectory information storage and fast retrieval, animation of sequences in spherical projection

mode with simultaneous terrain display.

5) **Trajectory filtering for efficient trajectory editing**: Taking HCI design principles in the context of WAMI and Biomedical data domains into consideration: reduction of search space of user input interaction by assigning trajectories of interest to a selection subset, in a feedback loop mechanism driven by user selection in the service of user focus on one object or a small number of objects for accuracy. All required trajectory editing operations via minimal user interaction for ease of use and efficient operation trajectory create, add point(s), delete point(s), move point(s), split trajectories, join trajectories, saving of selected trajectory subset(s).

6) **Full range of data visualization manipulation**: Per-layer visibility toggling, interactively move individual layer(s) up or down the visibility stack, per-layer transformation or all-layer transformation; all from one interface. The interface fulfils the requirements of HCI principles and meets the (input imagery) domain-specific needs.

7) **Composite view creation**: Applying HCI design principles and using features mentioned in (5) in conjunction with alpha blending, intensity thresholding, histogram manipulation, mix of single image and image sequence, viewing multiple images/image sequences either overlaid on top of each other or spread apart by user-defined offsets.

8) **Loose and tight coupling of external programs**: Such as trackers, filters, and segmentation and classification modules: program input gathering within KOLAM, program initialization, program execution control, and immediate display of results and program feedback in KOLAM. All these operations again serve the visual analytics and HCI design paradigm of KOLAM.

9) **Region-Of-Interest information and extraction**: Simple and versatile to use single click-and-drag interface. Provides ROI box coordinates as well as box width and height

directly in overlay display. These values interactively update depending on zoom level. ROI coordinate information can be passed to the Screen Capture module for precise user-selected region screen capturing.

10) **Screen Capturing and Movie Creation**: Human-Computer Interfaces for full-fledged screen capturing capability dedicated tool which works together with user display manipulation to create customized captures. Capturing enabled for image sequences; capture from either whole sequence or user-specified subrange. Captures saved as individual images, with coupling with mature movie-creation facilities of FFMpeg capable of directly outputting a movie given a custom capture area and temporal range.

11) **Biomedical applications**: Human-Computer Interfaces for Single and Multi-stage object tracking; visualize big data histopathology imagery, support most popular image format libraries thus providing support for widest possible number of biomedical image formats. Interactive one-click segmentation relabeling for ground truth generation for classifier training. Support for multiple labels which are cycled through via single mouse clicks. Support for features to aid user operation such as segmentation class boundary toggle-able display, all class boundary toggle-able display. Saving of intermediate result files per user change. KOLAM can also maintain project file(s) for individual files, as well as higher-level project files each of which manage multiple such projects. Such files are shareable among users.

12) **Extension to multi-monitor Wall Displays**: KOLAM's single display / monitor paradigm needed to be extended to allow seamless image data display and interaction across monitor arrays of any available arrangements. The largest such environment comprised a system of 40 wall mounted monitors, in an array configuration 8 monitors along the length of the aggregate display area and 5 monitors along the height (ie. an 8 x 5 array).

The relevant Human-Computer Interface principles were taken into account when creating these extensions.

13) **Integration of mature analytics packages**: Highly mature analytics libraries have been implemented as packages under R and Python. Work is underway to provide seamless access to these 3rd party library and language implementations within KOLAM as well as support for their output formats. These analytics capabilities can then be used in conjunction with other functionalities within KOLAM.

14) **Persistent Settings Management and Migration**: All of KOLAMs widgets involving user choice or value setting are persistently maintained across multiple usage sessions by the same user. These settings can also be migrated to any number of users for any of the supported platforms. This is highly powerful in that once a single user configures KOLAM extensively for highly specialized usage; this whole customization can be fully propagated for any other user on any other machine via a single file. Time of multiple users is saved, and there is no danger of users creating a different group of settings thereby ensuring the exact same operating environment can be set up with a single file selection for users with different levels of usage expertise.

15) **Persistent Data Usage Management and Migration**: KOLAM persistently maintains the usage histories for multiple static and time-varying image datasets, as well as the paths for input datasets and results of all tightly coupled modules such as object trackers (trajectories), segmentation and classification (input file paths, project file paths, intermediate result paths) etc. All of this customized data usage history can be propagated to any number of users for any of the supported platforms via a single migration file. Whats more, any number of such full customizations can be saved and shared as needed.

1.11 Credits

The history of development on KOLAM 2.0 is documented in [38]. Anoop Haridas (the author) has worked as the primary developer for KOLAM 3.0 for the past several years. During this time, a highly extensive feature set comprising several tools and capabilities were added. The details of these additions are presented in the following chapters. Initial development guidance was provided by Joshua Fraser, who added support for a number of image file formats and lent his expertise toward creation of the networking aspect of KOLAM's multi-monitor display capability. He also performed initial deployment of KOLAM 3.0 for the Mac OS platform.

Addition of new features was only half the story: extensive testing of all possible execution paths were carried out, which was crucial to the success of the multiple rounds of refinement and re-testing. Dr. Palaniappan has been very gracious throughout the years with testing of all developed features on the Mac OSX platform and providing extensive feedback regarding corrections and improvements. The author would like to note the efforts of all his colleagues in general, and Rengarajan Pelapur and Ilker Ersoy in particular; who tested KOLAM 3.0 the most and provided the most constructive feedback with regards to missing features, current feature loopholes, unhandled special cases and hard-to-track error conditions.

Chapter 2

KOLAM 3.0

2.1 Synopsis of KOLAM 2.0

KOLAM 3.0 began its life as a ported version of the KOLAM 2.0 code base. The main features of KOLAM 2.0 may be summarized as follows: (a) A multi-layer, static, extremely large geo-spatial image data visualization tool, which handled image data of high spatial and spectral resolution, (b) Multiple viewing modes (orthographic and spherical projections), (c) Terrain maps, (d) Edit some properties of individual layers: Use of Layer Editor and Color map Editor; Influenced by visualization needs of geo-spatial imagery, (e) Rapid roam, zoom and hyper-jump spatial operations; (f) Interactive color and histogram enhancement, and (g) Overview and cache glyph display tools for data navigation and memory usage monitoring. Further details of KOLAM 2.0 are presented in [38] and [9].

2.2 Motivation for KOLAM 3.0

Beginning sometime in the 2007 - '08 time period, the **acquisition of huge volumes** of wide-area, high-throughput airborne video was initiated by multiple agencies. The immediate need to track objects of interest across time within such imagery necessitated the development of feature-based tracking algorithms applicable to this high-resolution wide format video also known as **wide-area motion imagery (WAMI)**. The development, testing and refinement of such algorithms **required a system** that could *easily visualize* the results overlaid on the WAMI imagery in question, *smoothly navigate* through the time-lapse imagery with the overlaid results, provide *interfaces to initialize and terminate* algorithm execution, provide *capabilities to inspect* the results at custom zoom levels in manner that allowed for *visual determination of causes* of algorithm abnormal behavior or failure when working on the data under certain conditions (eg. objects that can be tracked in a straightforward fashion under optimum lighting and non-occluded conditions, fail to be correctly tracked when the objects are warped, in shadow or are occluded to varying degrees by other objects in the image), to *edit the generated results* as needed and a mechanism to *generate ground-truth* on the data to evaluate tracker algorithm performance via various statistical measures. The preferred algorithm development and testing environments such as MATLAB and OCTAVE are general purpose development environments which are not uniquely specialized in this context. That is, they do not provide the aforementioned facilities in a single, easy-to-use package. The existing features in KOLAM 2.0 made it an attractive target for feature addition and enhancement to provide the needed functionality; the end result of this being the conception of KOLAM 3.0 in its initial state.

The dissemination of the capabilities of KOLAM 3.0 in both literature and presentation formats ([39], [40]), coupled with software delivery and extended support in the form of implementation of desired domain-specific features, made KOLAM 3.0 an attractive choice for visual analytics platform to users outside the surveillance domain. At the forefront were potential users from the biomedical research community, who have been generating huge volumes of small to ultra-high resolution imagery via several microscopy imaging modalities. These different types of imagery, both static and time-lapse, need to be interactively visualized and require a number of analytical operations performed upon them. Modifying KOLAM 3.0 to meet these new needs was a straightforward set of tasks due to its extensible architecture. Indeed, it is important to note that KOLAM 3.0 can be used without alterations by researchers and analysts from both the surveillance and biomedical domains. The new features of KOLAM 3.0 have found application in cell motility tracking, cell cycle analysis and cell lineage visualization in histology imagery, and tumor identification via the visualization and relabeling of multi-level segmentation results for big-data histopathology imagery.

2.3 KOLAM Multithreaded Synchronization Architecture

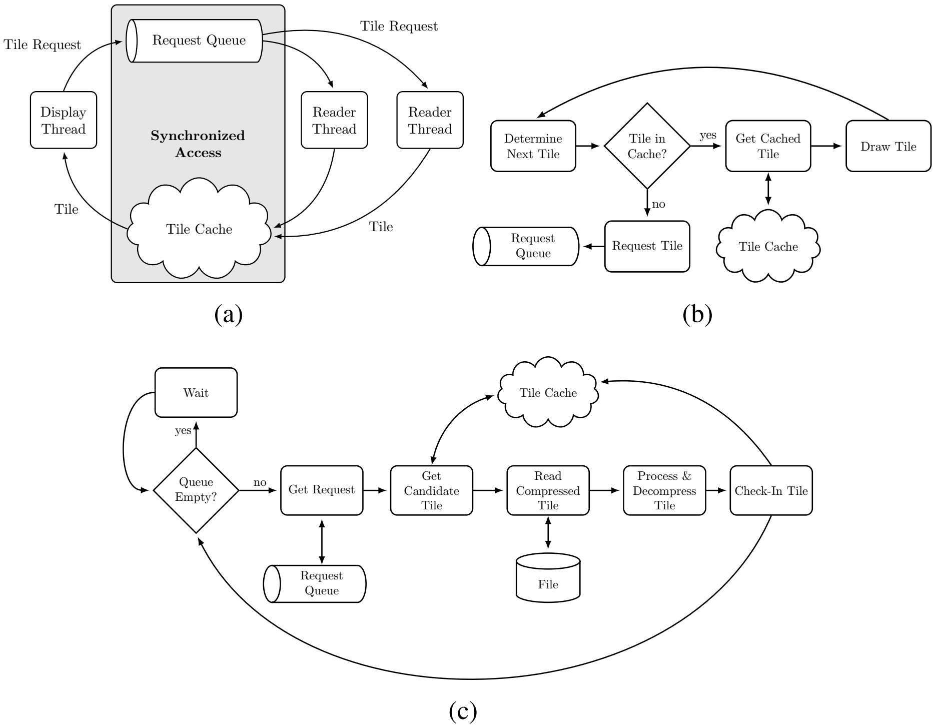
The reorganization of images into tiled multiresolution pyramids provides on-demand loading of partial regions and scaled views of the whole dataset. Loading the image piecewise as determined by the user's interaction, effectively amortizes the expense of transferring the entire dataset from secondary storage to primary and video memory. While the total amortized cost may exceed that of loading the original image, it is this on-demand view-dependent loading that provides an interactive experience for the user. Additionally, the total amortized cost for a given session will still provide savings as the user rarely views every tile contained in the dataset.

An efficient multiresolution temporally registered tile access mechanism is essential to provide smooth interactivity for any user-driven view into the WAMI data volume. Determining the visibility of *spatial-temporal tiles*, load balancing I/O throughput, decompression, and memory management are key parameters to optimize to provide a truly interactive experience for the user. A *spatial-temporal dual-caching* mechanism is proposed to efficiently handle a time sequence of large pyramid (wide-area) imagery. This mechanism was primarily designed and implemented by Joshua Fraser [39], with the author being the supporting developer making additions and modifications to the system from 2010 onward.

2.3.1 Multithreaded Cache Access and Management

A tile is the smallest data element in the cache management system. As the user interactively pans, zooms, and animates the view on to the data, tiles are loaded on-demand in order to provide that view. When a tile is determined to be needed in order to fill the user-requested view, a tile access cache-request is generated and queued. This request is handled asynchronously by a separate thread in order to keep the user interface interactive and prevent any I/O blocking that would occur as a result of fetching from secondary storage. To achieve this asynchronous behavior two types of threads are used: one display thread for user interaction and display, and multiple read threads to fulfill tile requests from disk. The number of threads is variable and can be tuned for different systems. Thread cooperation is shown in Figure 2.1. The work described in this sub-section was first done in [9] [41] [42], with additions by the author from 2010 onward.

The display thread, Figure 2.1(b), is responsible for user interaction, determining which tiles are necessary to provide the current view of the data, and drawing the view as tiles become available (with or without display buffering). The display thread transforms the



(a)

(b)

(c)

Figure 2.1: (a) Multithreaded synchronization of display and reader (worker) threads in KOLAM. (b) Display thread interactions with Request Queue and Tile Cache. (c) Reader thread interactions with Request Queue and Tile Cache.

current view requirement to a set of tile requests. Mapping the current view to the underlying data is dependent on position, scaling, and current time. The view is tracked in a global coordinate system with respect to the finest resolution of the pyramid and sequence order of the time-varying data. Uniquely identifying a tile in the system requires that each tile first be associated with its temporal index (mapped from time to a file), and then spatially within the pyramid structure.

When the current view is resident and the system load is light, KOLAM will predict what will be needed in the near future and generate requests for this data in advance. Preloading data is necessary to fulfill the user's expectation of interactivity; failure to fill the view in a timely fashion will cause portions of the view to remain blank or out-of-date. Prefetching can occur temporally with respect to the time sequence, but also spatially by pre-loading tiles surrounding the current view. Spatial prefetching attempts to predict tiles that will soon be required to fill the view for the current position in time. Scheduling tiles to be loaded immediately adjacent to the current view, but not yet resident, allows the system to anticipate the view's requirements and deliver needed data in time. Temporal prefetching determines which frames of the sequence will be needed (and when) in order to ensure that the associated pyramids are available in the frame buffer, and to issue requests for the tiles that will be needed in the next time step.

Under heavy system load, the demands of the view may exceed the ability of threads to deliver data. Compromises must be made to provide a seamless experience for the user. Spatially, rather than allow blank regions or out-of-date data, a consistent view can still be presented by progressively loading coarser tiles when the tiles at the current resolution are not yet available for display. Temporally, rather than playing back the animation too slowly, the system should attempt to fulfill the user's expectations of interactivity by dropping frames.

Reader threads (Figure 2.1(c)) provide asynchronous delivery of tiles to the tile cache. Initially a reader thread is in a wait state. The thread waits on a POSIX condition until there are tile requests in the queue. When the request queue is no longer empty, reader threads are awakened to retrieve requests from the queue and service those requests. When the thread has completed the task of delivering a tile to the tile cache, it either continues with the next request in the queue or returns to the wait state if the request queue is empty.

Reader threads are responsible for fulfilling the three phases of a tile request: reading, decompressing, and caching the tile in primary memory. The first phase of fulfilling a tile request is to find memory within the cache to hold the result of the request. A centralized cache contains the collection of all tiles currently resident and provides storage for new tiles if the cache has not reached its predetermined maximum size. If the cache is currently full, a candidate tile for replacement (removal) is chosen and checked out of the cache. Checking out a tile transfers ownership of the tile's memory to the thread and ensures that the tile scheduled for deletion is unavailable to other threads. The replacement tile can no longer be accessed by the display thread nor is it available to other reader threads until it is again checked back into the tile cache.

Once cache memory for a tile has been determined, the read phase begins. A requested tile is mapped from its spatial-temporal index first to a file in the sequence, and then to a byte offset and size within the file using the tile dictionary. Using this size and offset, a read system call is issued to the file to transfer the data from disk to memory. This system call is intentionally a synchronous call which will block the current thread of execution–but not other threads–until such time as the operating system can honor the request. The result of this read is a compressed tile which is saved in to a thread-local buffer.

Once the compressed tile is available, the thread is then responsible for decompressing the tile. The tile is decompressed from the thread read buffer in to cache memory. The result of this decompression is a display-ready tile located in the central tile cache. Following successful loading and decompression of a tile, the cache is notified of the presence of this new tile. The check-in of this tile serves two purposes: transferring ownership of the tile's memory back to the system for use and notifying the display thread that new data is ready for display.

2.3.2 Thread Synchronization and Load Balancing

Synchronization of threads occurs only when accessing two structures: the request queue and the spatial cache. This synchronization ensures that access to these protected structures is serialized amongst competing threads. Sychronization is provided by POSIX mutex and conditions. This synchronization is shown in Figure 2.1(a). The work described in this sub-section was first done in [9] [41] [42], with additions by the author from 2010 onward.

Coordination of tasks amongst the threads is provided by a workpile concurrent model. Tasks (in this case tile requests) are placed in the first-in-first-out request queue by the display thread. Threads retrieve requests from this queue in order to get tiles to deliver to the tile cache.

As described in the description of the read phase, KOLAM uses synchronous blocking reads when fetching compressed tile data. The purpose of blocking reads is towards load balancing of I/O and processing. Blocking reads provide a yield point for the executing read threads. When a read from disk is requested to the operating system, the thread will block yielding execution to other threads until that transfer from disk is available. While one thread is blocking on the read system call, other threads can utilize the CPU for processing and decompression. The nature of KOLAM's both I/O and CPU intesive demands is well suited to this workpile model.

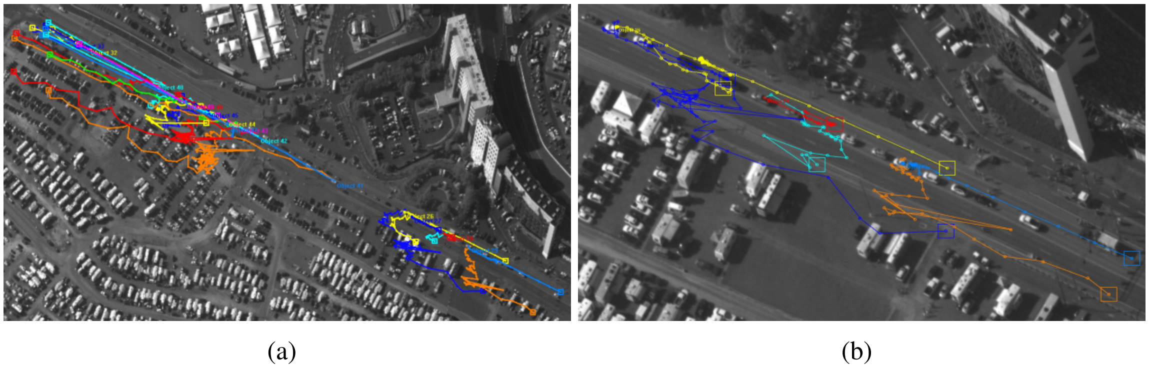
2.3.3 Spatio-Temporal Data Characteristics

As presented in section 1.4, the analysis of spatial data involves efficient data representation, analysis and management of data via data structures and distance and similarity measure design. Likewise, the analysis of temporal data involves identifying time-varying data patterns and correlations in the data. Now, analysis of data with both spatial and temporal components (ie. spatio-temporal data) includes all the above challenges, while adding unique challenges of its own. These were briefly summarized as: a) Scale: patterns and features of interest need to be searched for across multiple spatial and temporal scales, (b) Uncertainty: Spatio-temporal data is often incomplete, interpolated, collected at different points in time and with different underlying assumptions, and (c) Dependencies between observations, and complex topological relationships between different objects in space. These aspects are examined in further detail below.

(a) Scale: It is possible for spatial and temporal data to exist at different scales. Taking the dimension of time into account first: it is possible for the temporal dimension to include single or multiple levels of scale; where the scale used is the granularity of time. Temporal primitives can be aggregated into larger units, or split into smaller units. Most current visual analytics tools, KOLAM included, consider only a single level of temporal granularity, ie. data is treated as sequences of simple (time-point, value) pairs. KOLAM does possess the ability to set an arbitrary time step size within its animation system; nonetheless, this feature has not been used in a continuously extended manner by any of KOLAM's users for their visual analytics tasks. Let us now consider the spatial dimension(s) of the data.

The scale of spatial analysis is determined by two factors; the size of the units used to measure phenomena in the data, as well as the size of the units in which measurements are aggregated in the data. Patterns and relationships which are meaningfully discerned at one scale, may either be undetectable or exist as an opposite relationship at a different scale. In order to perform analysis on phenomena of interest in the data, it is important that the scale of analysis should match the scale of the phenomena in the data being analyzed. In the case of the various usage scenarios of KOLAM, the analytics components ie. object trackers, segmentation, classification analyze the data at its native resolution. However, other analytics operations within KOLAM itself, such as layer overlaying, visibility toggling, applying masks, setting transparencies, manipulating object trajectories, or editing segmentation/classification label values can happen at non-native resolution scales; indeed, at times analysts can exploit more than a single scale when performing these tasks. This is in keeping with the fact that analysis scales should be chosen according to analysis goals. Consider the example in Figure 2.2. Thus we conclude that, given the types of data being analyzed, visual analytics tools such as KOLAM must support visual analyses at multiple spatial scales at the least, if not providing full facilities for analyses across temporal scales. KOLAM's support for hierarchical multi-resolution (ie., multi-scale) data is explained in detail further in this chapter.

(b) Uncertainty: Uncertainty in data, defined as the 'degree to which the lack of knowledge about the amount of error is responsible for hesitancy in accepting results and observations without caution' [43], may be considered as a composition of the following data aspects: errors, imprecision, accuracy, noise, non-specificity, lineage and subjectivity. Reasons for data uncertainty include problems with the data acquisition methods, problems with data transmission, or the use of analytical processes which cause a loss of information.



(a) (b)

Figure 2.2: Visualization of multiple tracked objects and their trajectories in KOLAM using different vector primitives. The images show unstabilized (without local registration) and stabilized (locally registered) pairs of vehicle trajectories at different spatial scales. (a) represents an analyst's high-level view into the data and of all object trajectories for the given time span in the currently observed area. (b) represents the area corresponding to the bottom right of (a). At this spatial scale, the analyst may select and track individual objects of interest, or manipulate the visualization of the displayed trajectories. The images display results for a PSS WAMI dataset collected over Charlotte, NC.

A universal way to visually represent uncertain data does not exist. Nonetheless, visual analytics systems need to communicate the uncertainty in the data to their users. This needs to be done via either or both the analytical and visualization components.

(c) Inter-Observational Dependencies: The most fundamental aspect of spatio-temporal data is that the constituent spatial and temporal aspects are inter-dependent. This fact both constrains and defines how spatio-temporal data is processed and analyzed. In the spatial domain, this results in spatial auto-correlation. The same applies to temporal auto-correlation in the temporal domain. However, despite adding constraints (eg. for statistical analyses, unsuitability of techniques which require inter-observation independence), spatial and temporal dependencies also give rise to new information and possible data processing operations. These include: (1) Interpolation and Extrapolation (useful for populating incomplete data), (2) Spatial and Temporal inference, (3) Operations such as spatial and temporal navigation, and (4) By using references to common locations via spatial overlay(s), integration of different types of information from one or more sources cite book.

It must be noted that the effects of spatial and temporal auto-correlations are not 'straight-cut', and can be fraught with discontinuities and complexities. For example, in geo-spatial and WAMI datasets, the various heterogeneities of the geographical space, as well as natural and artificial barriers play a role. In histopathological data space, factors such as staining boundaries, tears in the tissue samples play a similar role. Additionally, every location has a degree of uniqueness relative to other locations. Temporal dependence can be affected by events involving radical or large-scale changes. Spatial and/or temporal proximity may also be phenomenon dependent. The resulting effect is that it is impossible to account for all the factors which affect spatial and temporal dependence, in the development of fully automated analysis methods for the problem space at hand. Visual analytics systems such as KOLAM play a crucial role in this gap, by leveraging the visualization component to inform the analyst about how and where spatial and/or temporal auto-correlations are being modified by local conditions and thus to make the requisite adjustments to the analysis.

2.3.4 *Spatial-Temporal Dual-Caching* Tile Organization

Motion imagery requires coherent caching both spatially and temporally. To determine temporal caching of tiles as the view varies in time requires knowledge of the next frame in time which may not be immediately adjacent in the sequence. As an investigative tool, simple sequential linear playback of the data may be insufficient for the demands of the user. The playback sequence of frames may occur in a cyclic pattern as opposed to just clamped from the beginning to the end of the sequence. Looping and rocking playback are examples of this cyclical pattern. The user should also be able to play the animation both forwards and in reverse at a specified frame rate. Additionally, the user may wish to stride through the data in order to visualize long sequences more quickly by setting the step size.

Dual-caching [39] is used to accommodate a sequence of files for animation. The first form of caching is at the temporal file level to organize temporal information and the second form of caching is at the spatial level which is discussed subsequently. Each frame in a sequence exists as a separate file. The cost of opening and parsing the file's header and generating the associated tile lookup tables requires that the results of this process be cached for fast access to the tile data. Additionally, because the length of a sequence can be very long, the entire sequence of file pointers and associated pyramid data structures cannot all remain resident due to limited resources–both with respect to memory and also the maximum number of open file handles imposed by the operating system. To manage this two-tiered caching, a separate file buffer is maintained. This buffer is kept full based on playback of time and the animation parameters set by the user. As the display updates time, this file buffer is kept up-to-date. When one frame of the sequence is removed from this buffer, a future frame is loaded from disk. At the time of this loading, no tile requests are actually issued, the file is only opened and parsed to build the necessary structures for accessing the tiles. As a file is paged out of the animation buffer, the tiles associated with that file are marked for recycling. This recycling marks those tiles in the cache as immediately available for reuse by future tile requests.

It is important to note that the order of images in the resident file buffer may not be sequential or in increasing order [39]. Rather the buffer will reflect the parameters used for playback of the animation. For example, the user may have chosen to loop a subset of the entire animation for playback with a negative stride. The cyclic nature of playback (loop and rock) and the need to skip frames (stride or frame dropping) in the sequence creates a

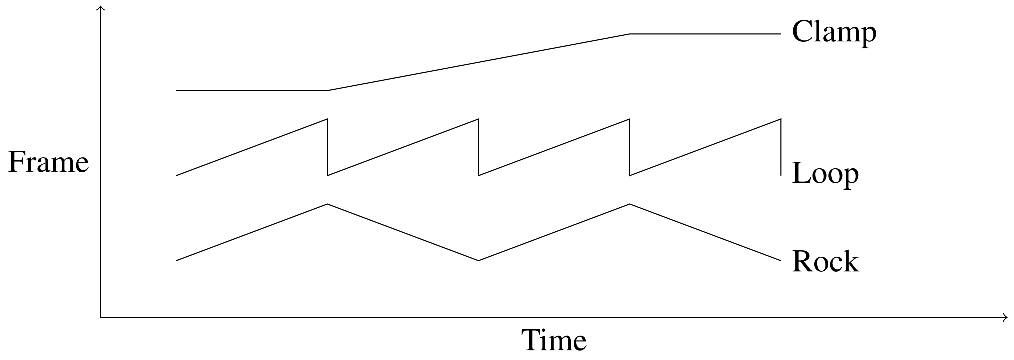


Figure 2.3: Sample ordering of frame sequences as a function of time showing three types of user selected playback modes including rocking back and forth, continuous looping, and play once.

non-linear sequence of frames needed for playback. In order to properly preload and buffer frames of the sequence, the system must be able to properly predict those frames which will be drawn in the future. Toward this goal, the sequence of past and future frames is modeled as a function of time. For any given time, the system must be able to query what frame would be drawn at that time. By monitoring latencies to fulfill a view, the system can choose to load only the sequence of frames which can be successfully retrieved from disk.

This ability to determine only what will actually be available for display as opposed to what would be next without the unavoidable latencies allows the system to manage its limited resources more effectively [39]. Accurately predicting which frame to load based on playback speed and load time, the system can fill the caches without wasting precious resources on frames that will be skipped.

The various modes of playback can be modeled using clamped, sawtooth, and triangle functions as shown in Figure 2.3. These models are used both for preloading data as well as maintaining a temporally coherent file cache. During the preloading phase, the system examines the frame cache and requests tiles for the current view at the next time. The next time is simply the span determined by the frame rate when the system is not under full load and can reliably fulfill all data requests. When the system is under a full load, the next time is the average current latency of successfully filling the view as determined by the display.

We now elaborate on the second form of caching at the spatial tile level [9] [41] [44] [42]. Tiles in the cache are referenced to their associated temporal frame in the animation sequence. A number of approaches can be considered for the tile replacement or *tile paging* strategy including random, minimum, First In First Out (FIFO), Not Recently Used (NRU), Least Recently Used (LRU) and our proposed *spatial multiresolution distance-based* (SMD) tile replacement or *SMD tile paging*. Among the most widely used operating system level paging strategies for managing memory is LRU. Similarly, the LRU *tile paging* or replacement strategy chooses the oldest tile contained in the tile cache to be swapped out for a new incoming tile. This approach is effective in that as the user zooms and pans the image, recently drawn tiles are likely to be needed again in the near future. The approach predicts that the least recently accessed tile will be the best candidate for removal from the tile cache.

While the LRU approach is effective, and has been implemented in KOLAM, it ignores additional spatial information that can be used to improve upon the LRU tile replacement algorithm. Although the user can make large disconnected jumps from one view to another in the image, this is not the typical way in which users visualize and navigate large motion imagery. Rather, users tend to pan around the image and zoom in and out in order to inspect the data. The nature of this user interaction suggests that paging based on a tile's distance from the current view rather than age of tiles would be more effective. Our preferred *SMD tile paging* strategy [9] [41] [44] [42] utilizes this spatial coherency to more efficiently manage the paging of the tile cache. Our approach is to keep all the tiles closest to the current view, and choose as replacement tiles those that are the furthest way in terms of spatial distance. A simple L_1 or Manhattan distance metric is used to update the distance of each tile in cache with respect to the current view. The result of this distance-based caching is a clustering of resident tiles nearest to the current view. When a new tile is requested and the cache is full, then the tile with the greatest distance value is chosen for replacement. **Algorithm 1** describes the two-level caching system (first being the temporal cache for the sequence of images, and second being the spatial cache for each image) that enables KOLAM to interactively animate big data multi-resolution imagery. **Algorithm 3** describes the handling of the different animation modes of KOLAM.

2.4 Symbiosis between Visualization and KDD Technologies

Knowledge Discovery and Data Mining (KDD) methods form the umbrella over the automated analysis methods that were discussed earlier on. While the strength of KDD methodologies lies in fully/semi-automated analysis of massive datasets, they tend to become black boxes in the hands of end-users, at times producing results that do not lead to problem solutions due to relevant expert knowledge being absent. On the other hand, visualization methods find their strength in leveraging human background knowledge, creativity and intuition to steer visual analysis, but fall short when the scale of the data becomes too large to be captured by the human analyst [4]. Modern visual analytics systems such as KOLAM seek to integrate the strengths of both and thus ameliorate their individual weaknesses. The current implementation of KOLAM achieves this goal by incorporating both a tight and loose coupling with object tracking programs, as well as segmentation and classification

Algorithm 1: Spatio-temporal Caching Algorithm

input : **Sequence** of **images** of dimensions $I[w \times h]$, KOLAM prior to loading the sequence

output: KOLAM has loaded the image sequence with the temporal and spatial caches properly initialized and populated. The display has been updated with the first frame of the image sequence

```
while Image sequence has not finished loading do
    bufSize ← User_Buffer_Setting;
    lookAhBufSize ← bufSize;
    // Buffer images around current frame into Temporal
        cache
    startF ← currFrame − $bufSize/2$;
    stopF ← currFrame + $bufSize/2$;
    for ($i$ = startF; $i$ < stopF; $i$ ++) do
        reqdImgList.append(clamped($i$));
        if i is not in imageBuf then
            // Load tiles into image's Spatial cache
            tempImage ← load($files[i]$);
            imageBuf[$i$] ← tempImage;
        end
    end
    // Update curr img after filling Temporal cache
    image ← imageBuf[$i$];
    // Check Temporal cache for images no longer needed,
        set these images as invalid
    if imageBuf.count() > bufSize then
        for ($i = 0$; $i$ < imageBuf.count(); $i$ ++) do
            if !reqdImgList.contains(imageBuf[$i$].key) then
                tempImage ← imageBuf[$i$];
                tempImage.setInvalid();
                invdImgList.append(tempImage);
            end
        end
    end
end
```

Algorithm 2: Spatio-temporal Caching Algorithm (continued)

```
while Image sequence has not finished loading do
    // Code from previous listing
    // Garbage collect invalidated images
    for $(i = 0; i < \mathsf{invdImgList}.count(); i\ ++)$ do
        cache.invalidate($\mathsf{invdImgList}[i]$);
        delete $\mathsf{invdImgList}[i]$;
    end
end
Return to KOLAM's main UI Loop;
```

programs, along-with its interactive visualization system. In doing so, these integrations accomplish multiple highly relevant tasks, including object tracking in Wide-Area Motion Imagery (WAMI) datasets and interactive Segmentation, Relabeling and Classification in big-data Histopathology datasets. Object tracking is explored in further detail in Chapters 3 and 4, while the roles played by segmentation, classification and relabeling for histopathology imagery are explored in Chapter 5.

2.4.1 Data Handling Characteristics

The heterogeneity of data types, data formats and data characteristics, as well as the size of the data, constitute major challenges to the operational implementations of systems integrating interactive visualization and KDD methodologies. The types of data which potentially need to be handled include spatial (geo-spatial, WAMI, histopathology) data, data from databases, data from sensors and video data. The integration of visualization and KDD creates additional handling issues, which are: (a) Data acquired in real-time needs suitable means of management, (b) At least a portion of the data will be of variable quality, necessitating that knowledge about the data be as complete as possible, (c) Incomplete data

Algorithm 3: Animation Algorithm

```
Input  : Image sequence $I$, task to load tiles from next level of the image pyramid.
Output: Updated view, with tiles loaded from either the lower or higher level of the
        image pyramid, based on selected display mode.
UserAct $= \{Play, Stop, Pause, Step, ToStrt, ToEnd\}$;
PlayMode $= \{Play, Loop, Rock, Blink\}$;
PlayDir $= \{Forward, Backward\}$;
PlayParam $= \{Fps, FpStep\}$;
currAct $\leftarrow$ UserAct$[1]$, currDir $\leftarrow$ PlayDir$[0]$;
currMode $\leftarrow$ PlayMode$[0]$, currfps $\leftarrow$ PlayParam$[0]$;
currfpstep $\leftarrow$ PlayParam$[1]$;
while User wishes to animate image sequence do
    if layerList.active().numFrames() $> 1$ then
        if User presses an Action button then
            currAct $\leftarrow$ UserAct$[UserPick]$;
            Update_KolamLoop_GUI(currAct);
        end
        if User changes Play mode then
            currMode $\leftarrow$ PlayMode$[UserPick]$;
            Update_KolamLoop_GUI(currMode);
        end
        if User changes Play direction then
            currDir $\leftarrow$ PlayDir$[UserPick]$;
            Update_KolamLoop_GUI(currDir);
        end
        if User changes Framerate then
            PlayParam$[0]$ $\leftarrow$ UserSetFps, currfps $\leftarrow$ PlayParam$[0]$;
            Update_KolamLoop_GUI(currfps);
        end
        if User changes Frames per Step then
            PlayParam$[1]$ $\leftarrow$ UserSetFpstep, currfpstep $\leftarrow$ PlayParam$[1]$;
            Update_KolamLoop_GUI(currfpstep);
        end
        S-T-C(currAct,currMode,currDir,currfps,currfpstep);
        UpdateDisplay();
    end
end
```

Algorithm 4: Animation Algorithm: Inner ELSE Block

```
// 'Else' corresponds to previous main inner 'If'
else
    if User wishes to load an image sequence then
        newLayer ← load(Layer(User_picked_sequence));
        layerList.active ← newLayer;
        continue;
    end
    else if User makes a sequence layer Active then
        layerList.active ← layerList[$UserPick$];
        continue;
    end
    else
        break;
    end
end
// Main 'While' loop from previous listing ends here
Return to KOLAM's main UI Loop;
```

requires knowledge of what data is missing, and ways to manage the missing data, and (d) Data of different spatial scales needs to be transformed to be compatible with other data. Details regarding KOLAM's spatio-temporal data handling capabilities are covered in this chapter, and other data handling issues are covered in Chapters 3, 4 and 5.

2.4.2 Required Features of Visual Analytics and KDD Integration

Given that the desired system should feature a (preferably tight, but loose also acceptable) coupling between its visual analytics and KDD components, a fundamental challenge is to organize and implement the functionality such that users can easily switch between the visual analytics tools and the data sources. This kind of integration may be achieved via the use of applicable APIs. Furthermore, generic visual analytics and KDD components which can be customized according to varying needs will aid in enhancing system flexibility by preventing the creation of isolated ad-hoc components. In KOLAM (implemented using the Qt cross-platform API), these objectives are achieved by having a master display region upon which various tool interfaces may be positioned as desired. These tools provide transparent access to KDD components such as object tracking programs and segmentation and classification programs. These programs take their inputs from the main display area via user interaction, execute asynchronously and produce results which are visualized immediately in the display area, often overlaid on the original data layers in one or more overlays. Finally, for KDD methods to be usable with visual analytics, the following features are preferred: (a) The methods be fast enough for efficient interaction, (b) Method parameters should be representable and understandable via visual representations, and (c) KDD parameters should be adjustable via visual controls.

2.5 Changes to the KOLAM User Interface

The addition of new functionality in response to needs described previously in this chapter necessitated multiple additions and modifications to KOLAM's UI. These may be broadly classified as changes to existing UI components, and new UI components that were added over time. The rest of this section is organized as follows. Subsection 2.5.1 addresses the design choices considered for the UI components in general. Next, subsection 2.5.2 describes the changes made to the existing UI components in KOLAM toward this thesis. Following this, subsection 2.5.3 provides an overview of the new UI components added to KOLAM. Finally, subsection 2.5.4 provides a detailed description of the evolution of design choices and implementations based on design methodologies and iterative user feedback for a single new UI component: the KOLAM-Loop tool. The other new UI components in KOLAM were conceived and evolved in a similar manner.

2.5.1 Design choices for UI components

The primary goal when designing each of the UI components is to fulfill all user requirements for accomplishing the specific task (eg. target tracking, ground truth generation and segmentation relabeling). The secondary goals were as follows. Since visualization is the primary purpose of KOLAM, each UI component was designed so as to maximize the remaining visible area of the screen, by limiting the size of each component and arranging visible components at pre-determined areas around the display region. These design choices help maximize the user focus on the image content being displayed on-screen and minimize visual distractions. Additional goal, it is highly desirable that the KDD components be implemented in a transparent, cross-platform manner in keeping with the cross-platform nature of KOLAM's implementation.

Users of the system may not all follow the exact same sequence of steps in order to accomplish a given task: one group of users might follow the sequence of steps to the letter without deviation, while another group might wish to repeat certain steps, or skip certain steps altogether. This happens if such users might be more prone to error, or changing their minds about actions they have just performed, or might attempt to interact with KOLAM with prior knowledge of a similar or unrelated tool they have used in the past. KOLAM attempts to address the different types of user interaction as best as possible, via a combination of methods. These include limiting options of interacting with the UI at each step, unobtrusively guiding user choices for next actions that are beneficial for KOLAM's continued stable operation by grabbing user attention towards preferred choices of action instead of providing no feed-forward and making the user fall back on his / her prior knowledge to determine what to do next.

2.5.2 Changes to existing UI components

The pre-existing UI components that underwent any and all types of modifications are described below.

Layer Editor

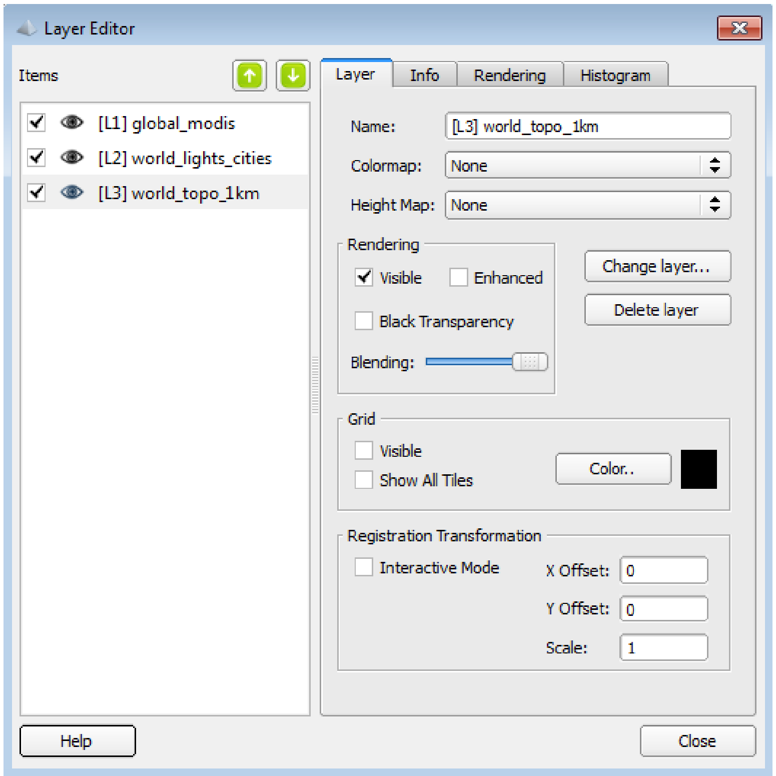
The Layer Editor is KOLAM's interface for managing the different visual properties of the image and image sequence layers that have been loaded for visualization. KOLAM 3.0's Layer Editor is depicted in Figure 2.4. In addition to performing the Qt 3 to Qt 4 port of all the features from KOLAM 2.0 [38], the author added the following features:

- Registration transformation; the ability to perform transformations upon individual layers themselves, as opposed to the default mode of applying transformations to the view. Translation and scaling transformations have been made available via numerical value input widgets to the user.

- Implementation of additional characteristics on the layer listing in the Layer Editor; specifically, the ability to move individual layers both forward and backward in the layer stack, as well as per-layer widgets to interactively toggle individual layer visibility.

- Alpha blending of multiple layers; distinct from the black transparency toggle which has been ported from KOLAM 2.0.

- Improved presentation of active layer information, specifically handling Windows, Linux and Mac OSX issues in a unified manner.



Layer Editor

Items

☑ [L1] global_modis
☑ [L2] world_lights_cities
☑ [L3] world_topo_1km

Layer | Info | Rendering | Histogram

Name: [L3] world_topo_1km
Colormap: None
Height Map: None

Rendering
☑ Visible ☐ Enhanced
☐ Black Transparency
Blending:

Change layer...
Delete layer

Grid
☐ Visible
☐ Show All Tiles
Color..

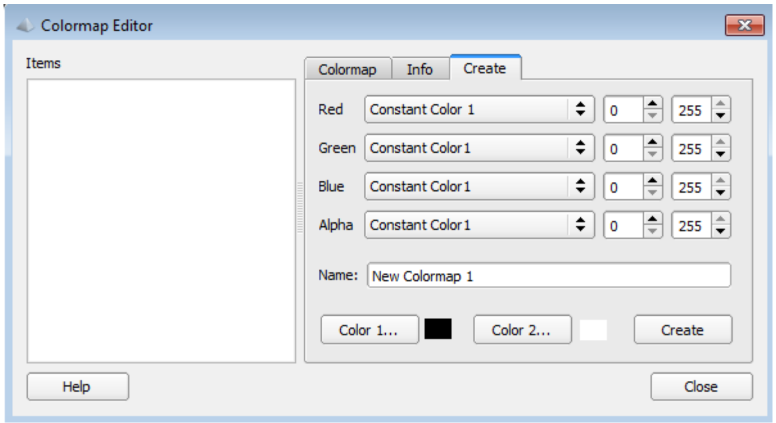
Registration Transformation
☐ Interactive Mode
X Offset: 0
Y Offset: 0
Scale: 1

Help
Close

Figure 2.4: KOLAM's Layer Editor.

Colormap Editor

KOLAM's Color-map Editor (Figure 2.5) provides the interface to load, edit and save any number of color lookup tables (LUTs), which may be applied to appropriate images. Besides performing the Qt version porting from KOLAM 2.0, the author also performed reorganization of the constituent widgets.



Red Constant Color 1 0 255

Green Constant Color 1 0 255

Blue Constant Color 1 0 255

Alpha Constant Color 1 0 255

Name: New Colormap 1

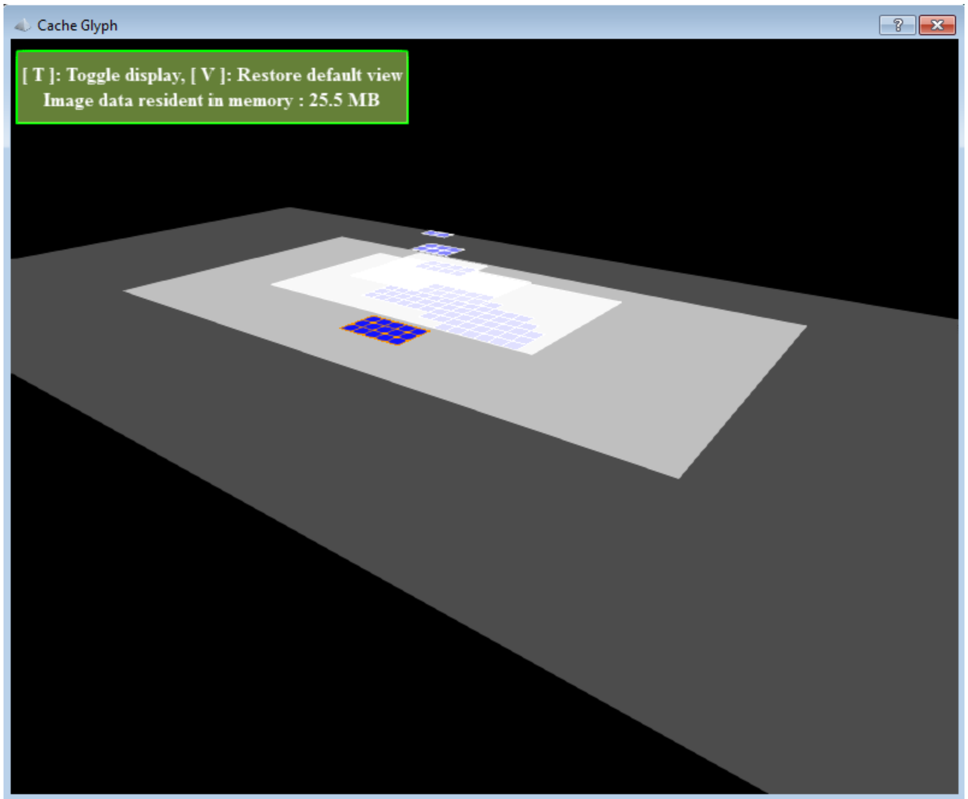
Figure 2.5: KOLAM's Colormap Editor.

Cache Glyph

KOLAM's Cache glyph is depicted in Figure 2.6. It provides up-to-date information about the paging status of image tiles in all states of memory residency: tiles being read in, tiles currently memory resident, tiles marked as old and scheduled for cleanup and so on. The author was responsible for making the user interaction with the tool more natural, correcting the navigation direction of panning, rotating and zooming, and providing the position invariant heads-up numerical display of tile occupancy in memory. The numerical display is an instance of KOLAM 3.0's overlay drawing, further discussed in Section 2.5.3 (pg.64).

2.5.3 New UI Components

The components described below were added to KOLAM 3.0's interface in order to address the need for new features, arising first from WAMI needs, followed later by biomedical imagery analysis needs. All components described here were wholly designed and implemented by the author.



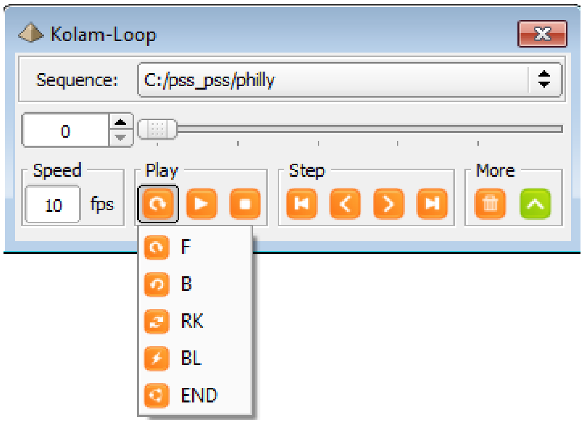
[T]: Toggle display, [V]: Restore default view
Image data resident in memory : 25.5 MB

Figure 2.6: KOLAM's Cache Glyph.

Kolam-Loop

KOLAM's Loop tool provides the user with different types of access to KOLAM's image sequence playback capabilities. The tool is modeled in the form of the most popular video playback interfaces, so that users are not burdened with learning delays even when using the tool for the first time. In addition to the ubiquitous playback features, the tool allows for interactive frame rate adjustment and provides access to multiple playback modes such as forward and backward looping, rocking between the start and end of the image sequence and blinking between two temporally adjacent images. The different parts of the tool are depicted in Figure 2.7.

The various modes of playback may be modeled using clamped, sawtooth and triangle



Kolam-Loop

Sequence: C:/pss_pss/philly

Figure 2.7: KOLAM's Loop Tool.

functions, as shown in Figure 2.8. These functions are used both for pre-loading data as well as maintaining a temporally coherent file cache.

Kolam-Loop is the front-end to KOLAM 3.0's image sequence animation system, which extends the caching mechanism of KOLAM 2.0 to utilize a spatio-temporal dual-caching strategy. This strategy utilizes caching at the temporal file level to organize temporal information, and also caches at the spatial level. Operations such as file opening, header parsing and tile lookup table generation are cached for fast tile data access, by maintaining a small buffer of images that is constantly kept up-to-date as the user animates the image sequence. A future image frame is loaded from disk whenever one frame of the sequence is removed from the buffer. However, this loading only involves file opening and file header parsing in order to gather the information necessary to build the required structures for tile access - no tiles are actually loaded at this point. As the old file is paged out of the animation buffer, its associated tiles are marked for recycling, which in turn marks those tiles in the cache as available for reuse by future tile requests. KOLAM 3.0 predicts what frames will need to be drawn at points in future time by monitoring the latencies to fulfill a view, thereby loading only those frames which can be successfully retrieved from disk.

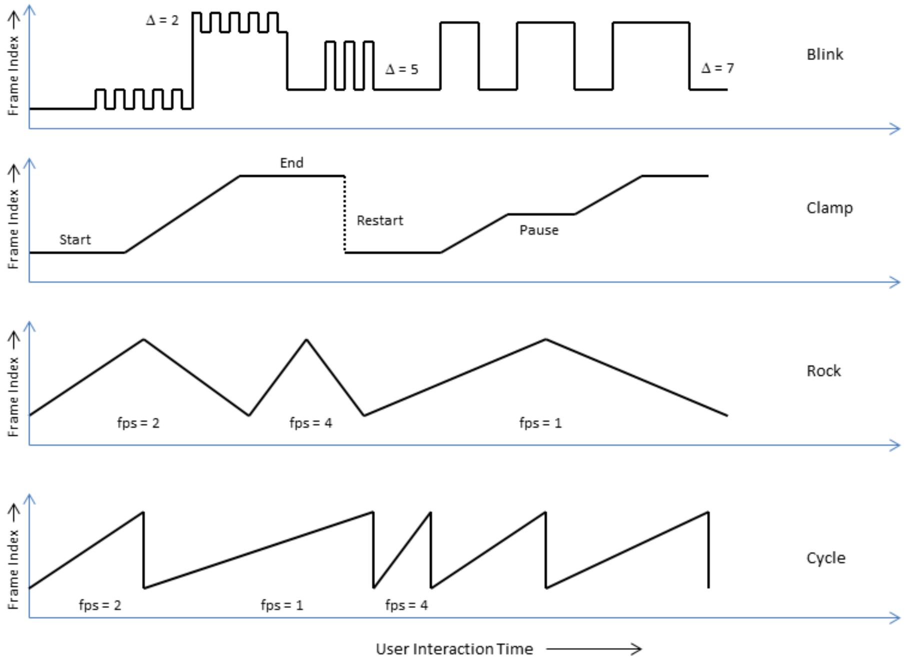
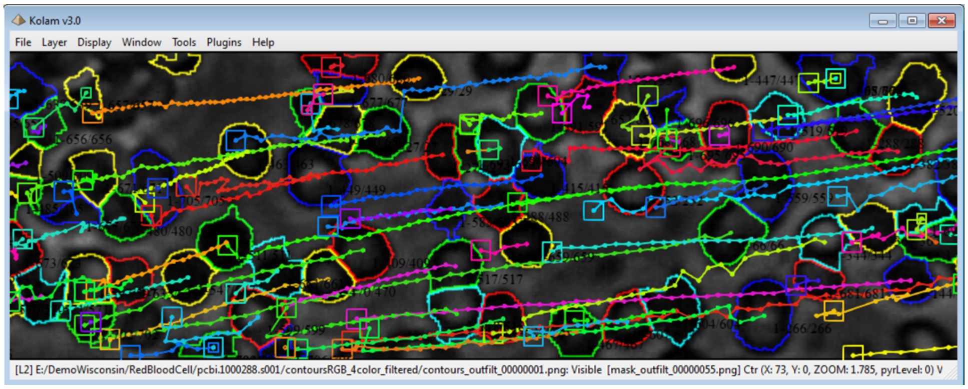


Figure 2.8: Sample ordering of frame sequences as a function of user interaction time showing four different user selected playback modes.

KOLAM 3.0's spatial caching strategy utilizes a spatial multiresolution distance-based tile replacement mechanism [9] [41] [44] [42]. This scheme exploits the navigation behavior of the typical user when examining an image: the user typically pans around and zooms in and out of regions in the image with some spatial proximity. In other words, navigation involving large, disconnected jumps to different portions of the imagery only account for the minority of typical user interaction; the user mainly pans and zooms in or out of adjoining image regions. Rather than using age of residency in the cache (as done by the classic OS LRU strategy), KOLAM uses distance from the current view as the yardstick to determine whether or not a particular tile of the image must remain resident in cache. This distance-based caching results in a clustering of resident tiles nearest to the current view.

KOLAM 3.0 allows for composition of multiple exploratory views of the data being visualized; by combining the end results of several components such as the image sequence animation system, multiple layer loading with per-layer visibility toggling, alpha blending between layers and so on. One example of such a view for biomedical imagery is presented in Figures 2.9 and 2.10. Another example of such a view for geospatial imagery is given in Figure 2.11.



[L2] E:/DemoWisconsin/RedBloodCell/pcbi.1000288.s001/contoursRGB_4color_filtered/contours_outfilt_00000001.png: Visible [mask_outfilt_00000055.png] Ctr (X: 73, Y: 0, ZOOM: 1.785, pyrLevel: 0) V

Figure 2.9: Composite views in KOLAM 3.0. Three image sequences have been loaded: A grayscale RBC sequence, and corresponding colored mask and outline sequences. Here, the mask sequence has been toggled OFF, allowing for visual animation of the cell outlines overlaid on the moving RBCs.

Kolam-Tracker

KOLAM's Tracker tool, seen in Figure 2.12, is the center of KOLAM's capabilities with regards to object tracking that provides facilities for tracker instantiation, ground truth generation, trajectory selection and editing. Taken together with all trajectory loading, saving and appearance manipulation menu items and the drag-and-drop trajectory file loading capability, it defines the sum total of all tracking and trajectory handling capabilities within KOLAM 3.0. The rich feature set in Kolam-Tracker was conceived and implemented in

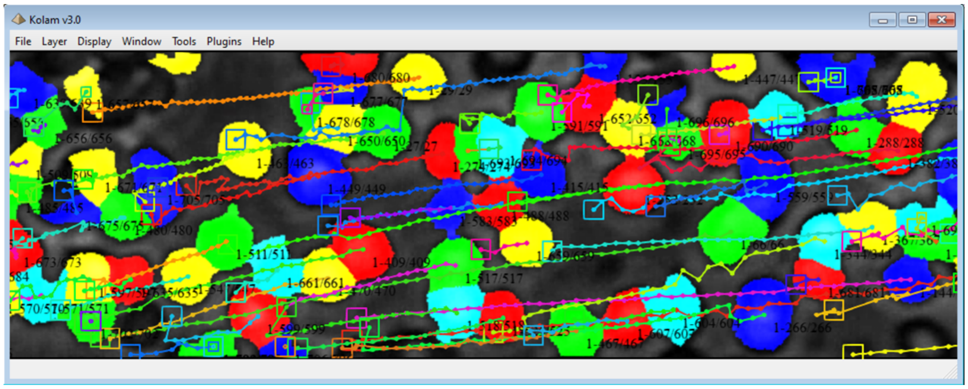
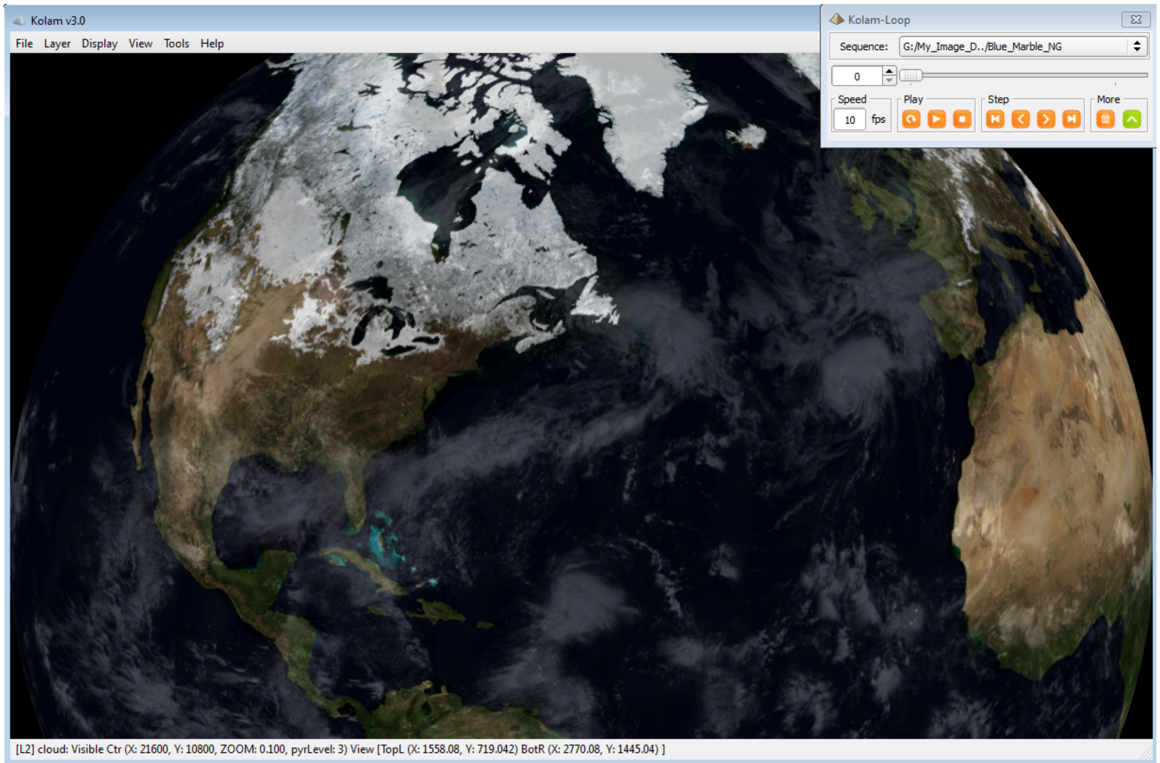


Figure 2.10: Composite views in KOLAM 3.0. Refer Figure 2.9 for details. Here, the cell outline sequence has been toggled OFF. The trajectory drawing seen in both figures is further explored in Section 4.2.

several phases, both in order to address user needs as they arose, and as the logical continuation/extension of features already implemented. This development process is described here. The initial set of requirements that Kolam-Tracker addressed included:

- Interfacing with the automatic tracker algorithms LOFT [45] and CSURF [46]: the user had to be able to supply initializations to the trackers in the form of one (CSURF) or two (LOFT) bounding boxes. Kolam-Tracker serves as the front-end of this process: it triggers KOLAM's overlay drawing system to accept the user's bounding box annotations on the imagery in the display area, signals the recycling of the overlay drawing object when input is complete, handles the writing out of bounding box extents along with information such as image data path and trajectory output path to a parameter file for the trackers to use, and initiates the tracker programs. Finally, Kolam-Tracker provides the interface to terminate executing tracker instance(s) and feedback from the underlying operating system regarding whether tracker instance(s) terminating without user intervention do so normally or otherwise.

- Generation of ground-truth in the form of centroids, lines and bounding boxes, for



Sequence: G:/My_Image_D../Blue_Marble_NG

0

10 fps

[L2] cloud: Visible Ctr (X: 21600, Y: 10800, ZOOM: 0.100, pyrLevel: 3) View [TopL (X: 1558.08, Y: 719.042) BotR (X: 2770.08, Y: 1445.04)]

Figure 2.11: Composite views in KOLAM 3.0. The sequence being visualized is the Blue Marble Next Generation (BMNG) sequence of 1 image per month, for the year 2004. Each of the 12 images has the following dimensions and raw size: 86400 x 43200 px, 11.2 GB. A single cloud cover image has been applied as a static layer with alpha blending while the underlying sequence is animated.

multiple objects over varying numbers of images within the loaded image sequence. The multiple points or bounding boxes that were created in visual association with single objects over several time steps were encapsulated as trajectories for the individual objects.

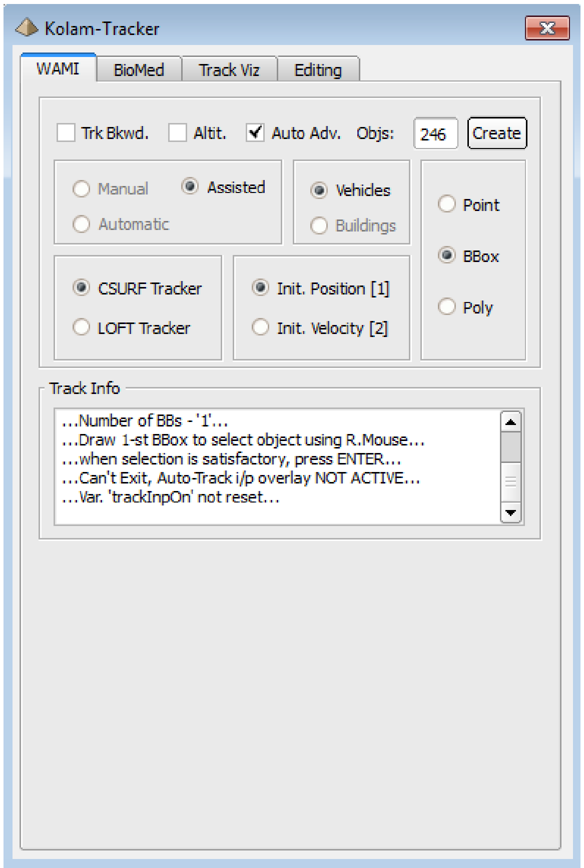
- Setting of per-trajectory properties such as color and visibility, and global trajectory properties such as thickness and trajectory set visibility. Other important properties configurable by the user include whether the view of the display area should be centered on the current vertex/centroid of the current trajectory as the image sequence is being played, whether the display of locally stabilized trajectories (when available) should be toggled on or not, and whether or not direct, temporally non-sequential navigation to a specific point on the current trajectory should be enabled or not.

- Providing feedback to the user regarding object selection for tracker input, object creation for new trajectories, starting of tracker execution and type of tracker termination (normal, abort mid-execution or failed-to-start states), and file locations for trajectories created. KOLAM 3.0 performed these functions via a text window currently located at the bottom of the ‘WAMI’ page of Kolam-Tracker.

Given Kolam-Tracker in its current form, this set of initially developed features comprise the contents of the ‘WAMI’ and ‘Track Viz’ pages respectively.

Upon completion of KOLAM’s UI facilities for enabling automatic and manual tracking, the next goal was to develop additional features to support tightly-coupled assisted tracking of objects, which is the ultimately desired object tracking type and UI to be developed within KOLAM. The Kolam-Tracker interface needs to support the tight, efficient coupling of automatic tracker initialization with manual intervention and correction of trajectories as necessary. The first step in this process was to develop an intuitive interface for the editing of trajectories. KOLAM 3.0’s current trajectory editing system includes several unary and binary trajectory operations, such as ‘ADD’, ‘DELETE’, ‘MOVE’ and ‘JOIN’. The trajectory editing operations are accessible within the ‘Editing’ page of Kolam-Tracker, and are described in detail in Chapter 4.

The final set of additions to Kolam-Tracker were implemented in response to perform single and multi-stage object tracking in biomedical imagery. Examples of such trackers are the MSC Tracker, which is a multi-stage tracking system used for muscle satellite imagery, and the wound tracker for wound imagery. The set of supporting interfaces for these types of trackers, while similar in some aspects to the features available under WAMI tracking, were grouped separately under Kolam-Tracker's 'BioMed' page. This was done so that specific features required by biomedical researchers could be provided independent of the WAMI functionality, and to avoid confusion and widget mis-use if all capabilities were to be included on a single page.



□ Trk Bkwd. □ Altit. ☑ Auto Adv. Objs: 246

○ Manual ● Assisted ○ Automatic

● Vehicles ○ Buildings

○ Point ● BBox ○ Poly

● CSURF Tracker ○ LOFT Tracker

● Init. Position [1] ○ Init. Velocity [2]

Track Info

```
...Number of BBs - '1'...
...Draw 1-st BBox to select object using R.Mouse...
...when selection is satisfactory, press ENTER...
...Can't Exit, Auto-Track i/p overlay NOT ACTIVE...
...Var. 'trackInpOn' not reset...
```

Figure 2.12: KOLAM's Tracker Tool.

Kolam-Preferences

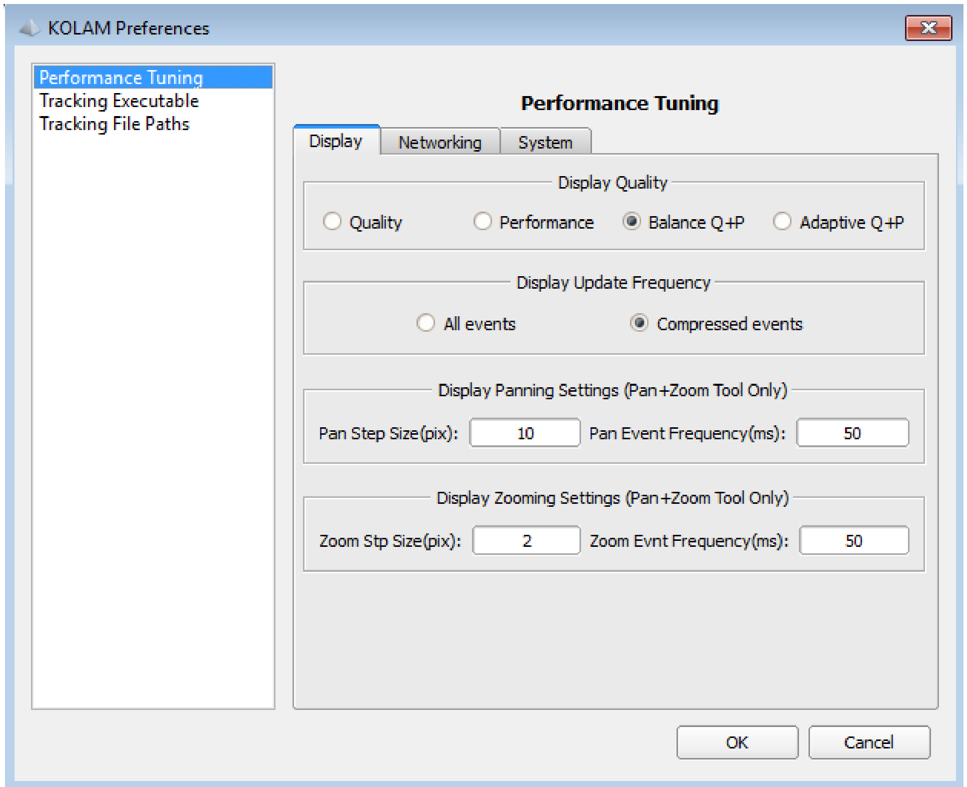
Several of KOLAM 3.0's tools (Tracker, Screen Capture etc.) allow for their specific properties to be modified and maintained locally. Furthermore, for preserving user customized values in between different execution sessions of KOLAM, all desired values are written out to OS-specific locations intended for maintaining persistent program settings (the Registry on Windows OSes, .ini files on Unix/Linux systems, and Library property list (.plist) files on Mac OS systems).

However, despite each tool managing its own attributes in these two ways, certain properties either do not precisely fall into the attribute groups of these individual tools, or are applicable to the functioning of multiple tools. These types of settings thus need to be maintained one level above individual tool attributes. KOLAM 3.0 handles such properties by providing global-level access to them within the Kolam-Preferences tool. Depicted in Figure 2.13, this tool currently manages the following categories of variables, with the ability to support as many more categories as desired:

- Performance Tuning: This category currently includes various display settings (quality, update frequency, and pan-zoom properties), network synchronization frequency settings for the multi-monitor display aspect of KOLAM, and general system settings.

- Tracking Executable: Currently allows for configuration of multiple system paths associated with trackers developed within certain environments, such as MATLAB or OCTAVE.

- Tracking File Paths: Allows for setting of paths for tracker input parameters, trajectory output location for the trackers, location for trajectory archival, and selecting the trajectory database type.



Display Quality

○ Quality ○ Performance ● Balance Q+P ○ Adaptive Q+P

Display Update Frequency

○ All events ● Compressed events

Display Panning Settings (Pan+Zoom Tool Only)

Pan Step Size(pix): 10 Pan Event Frequency(ms): 50

Display Zooming Settings (Pan+Zoom Tool Only)

Zoom Stp Size(pix): 2 Zoom Evnt Frequency(ms): 50

Figure 2.13: KOLAM's tool for user customization of several of KOLAM's properties at the global level.

Overlay Planes for Layer Annotation

Several of KOLAM 3.0's tools from this point utilize the overlay drawing system that has been implemented in KOLAM 3.0 for versatile layer annotation. The different features of KOLAM 3.0 that make it a highly useful tool for multiple domain visualization and analysis - trajectory visualization, annotation and editing, region-of-interest (ROI) selection for position determination and screen capturing, histopathology segmentation relabeling and so forth - which are discussed in the following chapters also rely on the overlay drawing system as a fundamental component. Therefore, KOLAM 3.0's overlay drawing system is now described in detail, so as to provide proper context for all the tool and functionality descriptions that follow. A visual depiction of the overlays among the other elements of KOLAM 3.0's display system is given in Figure 2.14.

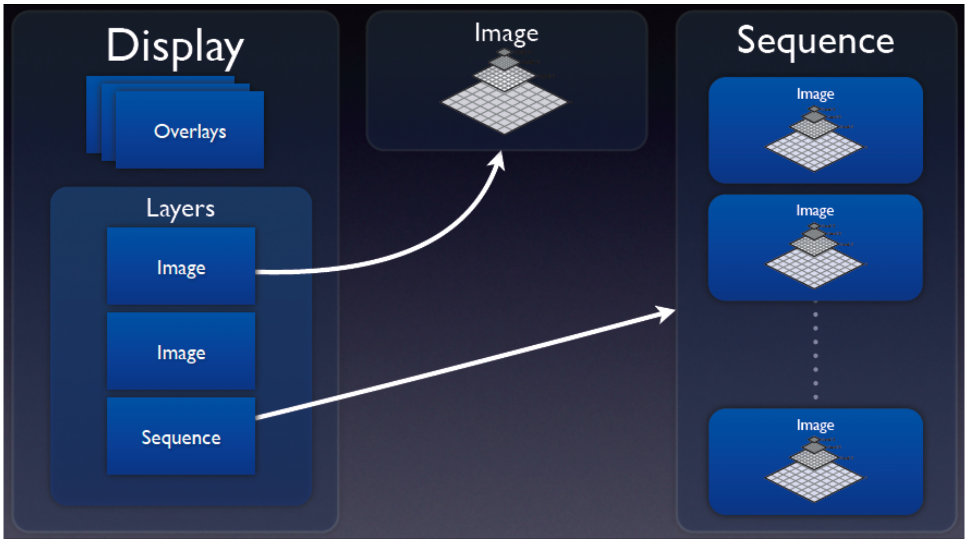


Figure 2.14: Overlays in relation to single image and image sequence layers, with or without embedded images, in KOLAM 3.0.

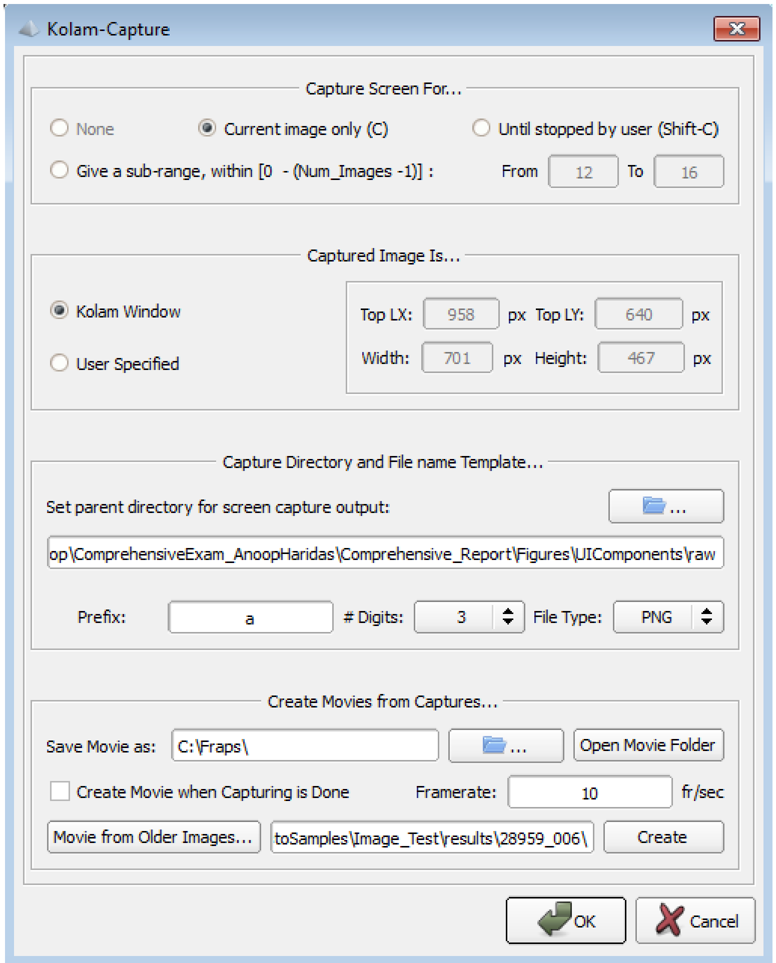
KOLAM 3.0's overlay system is implemented as a set of customizable drawing planes, positioned 'on top' of KOLAM 3.0's layer drawing system. Overlay drawing is independent of the type of the underlying data being displayed, meaning that they can be used with both single image and image sequence layers without needing modifications specific to either. Event management of user interactions appropriately separates the flow of input requests into the *Annotation Overlay* and *Data Layer* event streams. This management is customizable per overlay: for example, KOLAM's manual ground truth handling overlay responds to user events differently from the Region-Of-Interest (ROI) selection overlay.

Drawing on the overlays is a combination of Qt's QPainter class functionality and OpenGL commands. The QPainter instance enables vector drawing of primitives such as points, lines and polygons, font support for text rendering, support for image rendering via pixmaps and several other features. Qt allows OpenGL commands to be interleaved with QPainter calls, thereby allowing for efficient handling and passing of geometry to the GPU for rendering. KOLAM's overlays can be created as needed and destroyed (recycled) when no longer necessary; furthermore, their visibility may be toggled 'ON' or 'OFF' similar to layers. This directly translates to KOLAM functionality such as "Turn all trajectories ON or OFF" and so on. With this description of KOLAM 3.0's overlay system, we now continue with the description of the remaining tools newly created within KOLAM 3.0, several of which also use the overlay functionality.

Screen Capturing

Kolam-Capture, KOLAM 3.0's screen capturing tool, facilitates capture of whole window content or a user-definable sub-region of the whole display window. Depicted in Figure 2.15, it is capable of extracting captures from single or multiple images. The sub-region for capturing may currently be specified via KOLAM's ROI facility (Figure 2.16). The tool also permits dynamic setting of capture save location, along with capture file name and file type specification.

Finally, Kolam-Capture is capable of generating movie files from image sequences. This is made possible by interfacing with the industry standard FFMpeg library for video playback and video file format conversion. The majority of video files generated from KOLAM-playable image sequences have the requirement of being small in size while preserving the highest possible quality. Kolam-Capture does this by encoding videos in the mp4, H.264 format, with highest quality setting, as default. The provided interfaces allow the user to interact with FFMpeg in a limited fashion, through the setting of the output frame rate and setting of the movie save path. While Kolam-Capture allows for movie generation to be a ‘post-processing’ step, to be executed once image capturing is completed; the author posits that the true power of the movie generation feature lies in its ability to encode movies from pre-existing image sequences: in other words, creation of movie files without loading the image sequence in KOLAM is possible. The pre-set high quality factor with high video compression will potentially make this part of KOLAM indispensable to users with the need to generate videos from image sequences on a frequent basis.



Kolam-Capture

Capture Screen For...

○ None ● Current image only (C) ○ Until stopped by user (Shift-C)

○ Give a sub-range, within [0 - (Num_Images -1)] : From 12 To 16

Captured Image Is...

● Kolam Window

○ User Specified

Top LX: 958 px Top LY: 640 px

Width: 701 px Height: 467 px

Capture Directory and File name Template...

Set parent directory for screen capture output:

op\ComprehensiveExam_AnoopHaridas\Comprehensive_Report\Figures\UIComponents\raw

Prefix: a # Digits: 3 File Type: PNG

Create Movies from Captures...

Save Movie as: C:\Fraps\

□ Create Movie when Capturing is Done Framerate: 10 fr/sec

toSamples\Image_Test\results\28959_006\

Figure 2.15: KOLAM’s Screen Capture and Movie Creator Tool.

ROI Selection

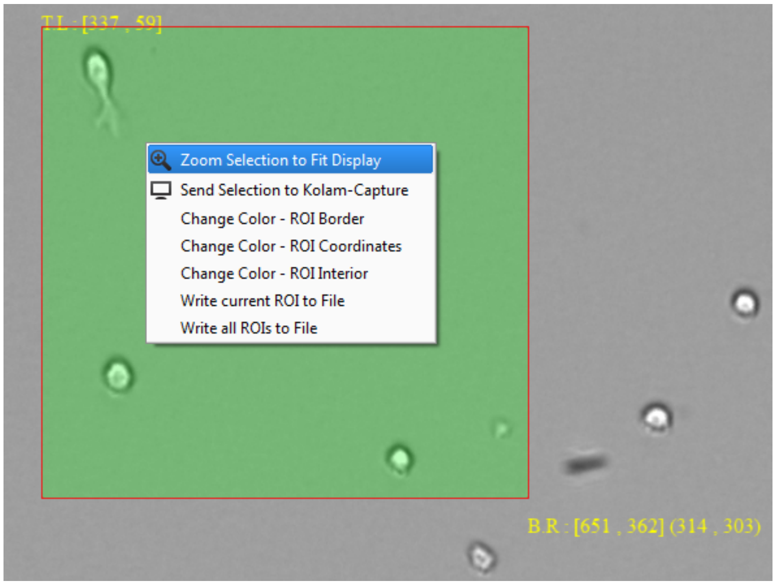

T.L : [337 , 59]
Zoom Selection to Fit Display
Send Selection to Kolam-Capture
Change Color - ROI Border
Change Color - ROI Coordinates
Change Color - ROI Interior
Write current ROI to File
Write all ROIs to File
B.R : [651 , 362] (314 , 303)

Figure 2.16: KOLAM's Region-Of-Interest Selection feature. Note the coordinates of both the top left and bottom right corners of the selected region, as well the ROI width and height in the bottom right. These values interactively update as the user click-drags and changes the selected ROI. Additional related functionality is provided via the context-sensitive menu, which is also displayed here.

In serving its role as the result visualization and examination platform for the iterative correction and refinement process of tracker development, KOLAM needed to provide one simple and crucial feature to users: ***immediate access to the image/pixel dimensions of objects of interest*** such as vehicles or cells. One example of how such information was used was in determining appropriate sizes for sliding windows for the tracker algorithms. Obtaining the object pixel dimensions was also vital when users needed to re-verify that their algorithms had indeed acted upon the desired region of interest. KOLAM 3.0's ROI selection facility was developed in response to such needs. Over time, the ROI selection feature was also interconnected with the main display and screen capturing modules to allow for the fitting of the user-selected region to the display and allowing the user to graphically determine sub-regions of the display for single or multiple image captures. Figure 2.16 clearly depicts all available ROI Selection features: the bounding box enclosing the region of interest may be redrawn any number of times by the user without exiting the ROI selection mode, and includes useful information such as image coordinates of the top left and bottom right corners of the ROI, and ROI width and height. The context menu attached to the ROI provides one-click access to view zoom-in to the ROI, screen capture coordinate specification, single and multiple ROI output to file, and options to alter the coloring of each ROI element.

Pan and Zoom

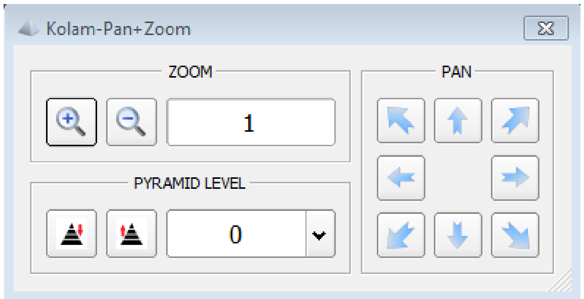
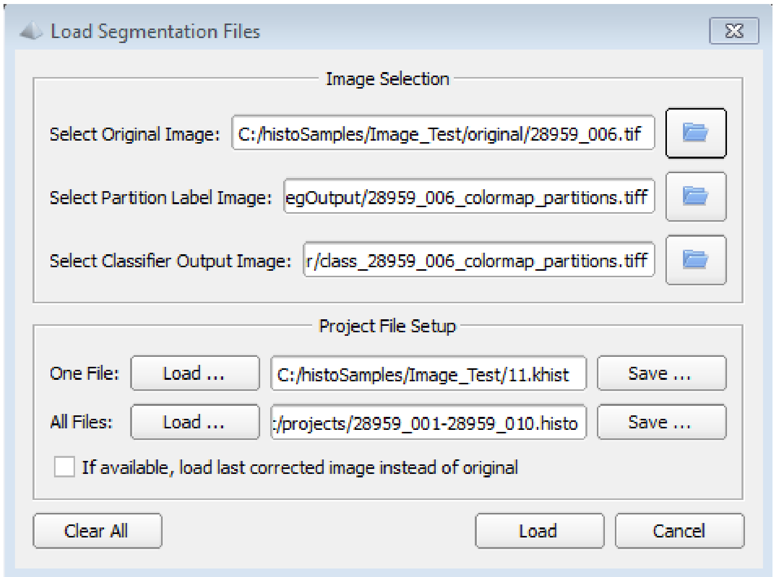


Figure 2.17: KOLAM's Pan and Zoom Tool.

KOLAM 3.0's Pan-and-Zoom tool facilitates non-mouse based roam and zoom navigation of the display area. Additionally, the user may configure the step size and granularity of the roam and zoom operations on-the-fly via Kolam-Preferences (Figure 2.13). Finally, the tool allows the user to navigate to the different levels of the image pyramid, when multi-resolution imagery is being displayed. The Pan-and-Zoom tool is also KOLAM 3.0's first tool with a **scalable interface**: when re-sized, all visual contents of the tool scale in order to maintain the default view (Figure 2.17) as closely as possible. Achieving this level of widget content ‘elasticity’ was made possible by customizing the components and layouts of Pan-and-Zoom for maximum spatial flexibility, as well as overcoming an undocumented loop-hole within the Qt API itself.

File Handling for Segmentation Relabeling

This tool (see Figure 2.18) is KOLAM’s interface for all file handling operations with regards to visualization and interactive relabeling of single and multi-level segmentation results. It also provides the interface for handling sequences of histopathology images as image collections: results of label editing are saved in a single location and are accessible via an extensible project file mechanism. Further details of the tool are presented alongside the description of interactive segmentation relabeling for histopathology imagery in Chapter 5.



Load Segmentation Files

Image Selection

Select Original Image: C:/histoSamples/Image_Test/original/28959_006.tif

Select Partition Label Image: egOutput/28959_006_colormap_partitions.tiff

Select Classifier Output Image: r/class_28959_006_colormap_partitions.tiff

Project File Setup

One File: C:/histoSamples/Image_Test/11.khist

All Files: :/projects/28959_001-28959_010.histo

□ If available, load last corrected image instead of original

Figure 2.18: KOLAM’s Segmentation Relabeling File and Project handling interface.

2.5.4 Evolution of Design

Here, the evolution of the KOLAM-Loop tool (from its initial conception to its final production code form) is examined in detail. This is a representative example: all other new UI components in KOLAM were designed and implemented in a similar manner; and preexisting UI components were enhanced and/or modified using the same set of steps. The process is sequentially illustrated from Figure 2.19 to Figure 2.31. The first step is the formulation of the **Problem Statement**, which states the need and the problems which must be solved, in general terms. It has been broken down as follows:

- Given that the image dimensions are much greater than the screen dimensions (especially for big data imagery), how can the user visualize, interact and analyze the data of interest in a useful way using KOLAM?
- Given that the temporal aspect of the data is represented as a sequence of big data imagery, how can such data be temporally navigated in a useful way?
- If the user has points of interest to focus on, how can this be productively reconciled with the temporal navigation of the data?
- If the user wants to follow point(s) of interest temporally, how does this affect the design of the interface to be provided to the user?
- Would the user like to save Region(s) of Interest (ROIs) for later analysis?
- Would the user like to visualize and/or interact with multiple temporal sequences simultaneously?

With the Problem Statement formally defined, the next step involves identifying the **Challenges** in solving these problems. The challenges identified are now listed below:

- Is it possible to translate prior user experience with temporal image navigation software (movie players) to provide an interface with familiar features and no additional learning curve?

- Unlike movies, which are typically sequences containing a relatively smaller number of important frames amongst a larger number of relatively unimportant frames, spatio-temporal data is a sequence of distinct frames: each of which is potentially important data. Therefore, should additional (or different) interaction options be provided for navigating and examining such data? If so, what are these?

- What are the interaction rules for data being animated versus data that is not being animated?

- Given that the temporal dimension of data is represented as sequence of imagery, how can such data be navigated temporally in a useful way?

- How can the interaction smoothness expectations of users be accommodated given limited system hardware resources?

- Given temporal sequences of longer lengths, will users having time constraints expect different means of navigating the data at rapid rates? If so, what are these?

Given both the problems and the challenges involved in solving them, we first systematically outlined a **set of HCI principles** for crafting an initial solution (set) to the same, following which iterative user-feedback-driven refinement was guided by appropriate **UI Design Methods**. Shneiderman et al. [5] provide a standard template for the former. The different aspects of this template are as follows:

- Navigation of the Interface: In order to facilitate the most productive navigation of the UI to accomplish the desired tasks, all task sequences must be standardized, unique and descriptive headings must be used where applicable, and appropriate widgets must be used whenever binary choices are involved.

- Facilitating Data Entry: Ensuring consistent and efficient data entry requires consistency of data entry transactions, that input actions needed to be made by the user be as close to the minimum as possible, minimizing the burden of interface-related knowledge which the user is required to memorize, compatibility with the display of the data, and flexibility of user control with regards to data entry.

- Display Organization: Factors playing a role in the best possible organization of the display include: consistency of data display, making assimilation of information as efficient as possible for the user, minimizing the number of display features and characteristics that the user needs to remember in order to operate the display, keeping the data display compatible with data entry, and allowing for flexible user control of the data display.

- Determine Users' Skill Levels: Based on skill levels, there can be 3 types of users: novice / first time users, knowledgeable intermittent users, and expert frequent users.

- Golden Rules for Interface Design: These are generalized rules that are applicable to all types of UI design and implementation, and are: Maintain consistency, Allow for universal usability, Provide informative feedback, design dialogs to yield closure, prevent errors, support internal locus of control, and reduce short-term memory load on the users.

While seeking suitable candidates for the aforementioned Design Methods, an extensive search was performed in the UI Design Method literature, following which the most suitable candidates for the proposed iterative refinement to KOLAM and this thesis were identified as **Scenario-based Design** [47] and **Participatory Design** [48] [49]. These are further elaborated on below.

- **Scenario-based Design**: This design method first requires creating a narrative that explores future use of a product from the user's point of view, allowing the designer to reason about its place in the real world application. The focus of the design is on what technology enables rather on the technology itself. The design is written based on the design team's understanding of the target users.

- **Participatory Design**: The hallmark of this design method is the active user engagement throughout the research and design process. The method includes flexible modeling and creative toolkit and design workshops. The design process is inspired and guided by the participants' creative insights. Paired with design expertise, this method supports the creative authority of designers to translate collaborations with the users into design criteria, services and artifacts.

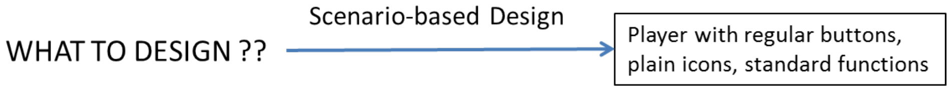

WHAT TO DESIGN ??
Scenario-based Design
Player with regular buttons, plain icons, standard functions

Figure 2.19: Evolution of the KOLAM-Loop User Interface Tool (I). Scenario-based design was used to obtain the initial UI, with full-sized buttons, plain icons and standard functions.

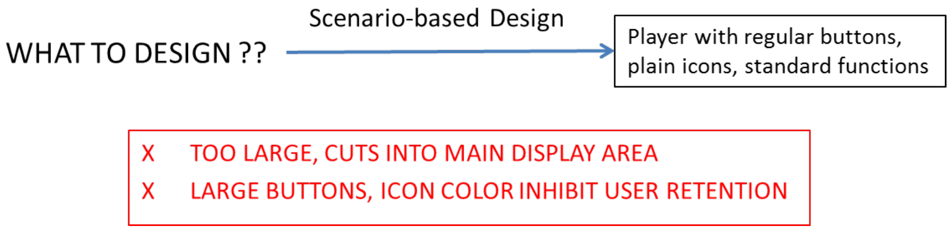

WHAT TO DESIGN ??
Scenario-based Design
Player with regular buttons, plain icons, standard functions
X TOO LARGE, CUTS INTO MAIN DISPLAY AREA
X LARGE BUTTONS, ICON COLOR INHIBIT USER RETENTION

Figure 2.20: Evolution of the KOLAM-Loop User Interface Tool (II). User feedback highlighted two problems: the UI was too large and obstructed KOLAM's main display area significantly. Also, the large buttons and the non-colored button icons were inhibiting user retention of individual button functionality for future use. Both defects identified per [5].

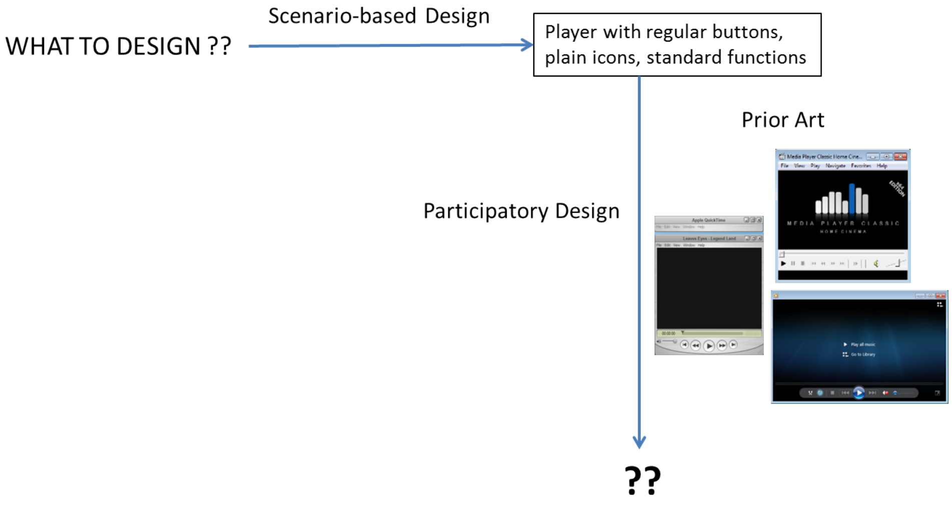


Figure 2.21: Evolution of the KOLAM-Loop User Interface Tool (III). Participatory Design and Prior Art research [5] were leveraged to discern avenues for improvement and change.

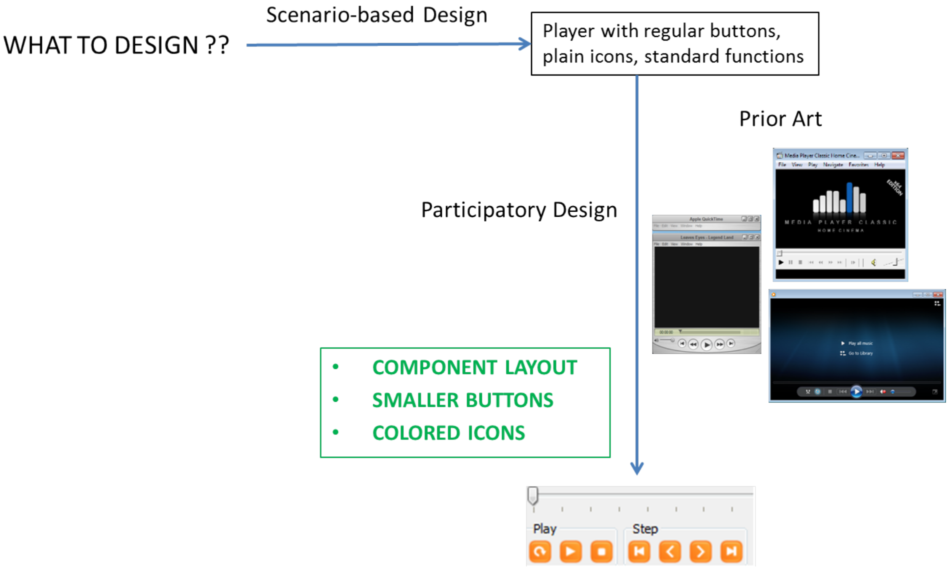


Figure 2.22: Evolution of the KOLAM-Loop User Interface Tool (IV). In response to Prior Art research and P.D. feedback, the component layout shown was chosen. Also, smaller buttons and colored icons for the buttons were decided upon. The choice of colored button icons was per guidelines in [5].

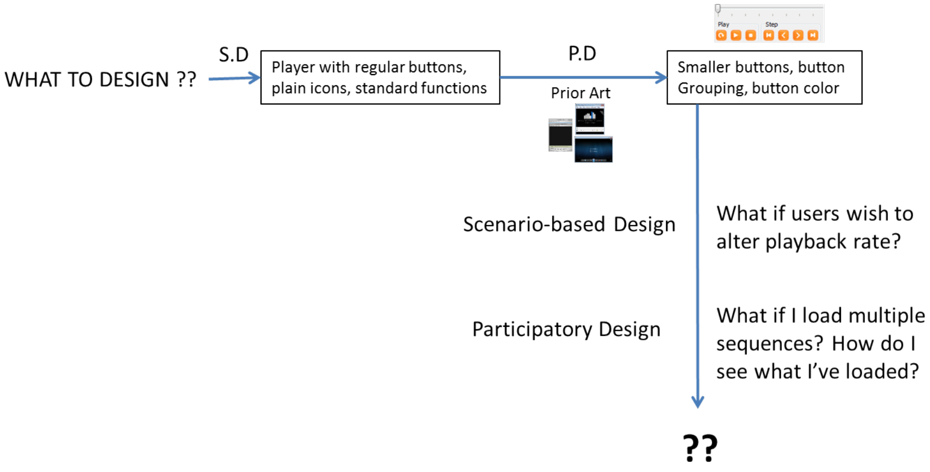

WHAT TO DESIGN ??
S.D
Player with regular buttons, plain icons, standard functions
P.D
Prior Art
Smaller buttons, button Grouping, button color
Scenario-based Design
What if users wish to alter playback rate?
Participatory Design
What if I load multiple sequences? How do I see what I've loaded?
??

Figure 2.23: Evolution of the KOLAM-Loop User Interface Tool (V). Scenario-based Design reveals the user need to dynamically alter the playback rate of the sequence. Participatory Design reveals the need to access individual sequence details in the event multiple sequences are loaded for playback.

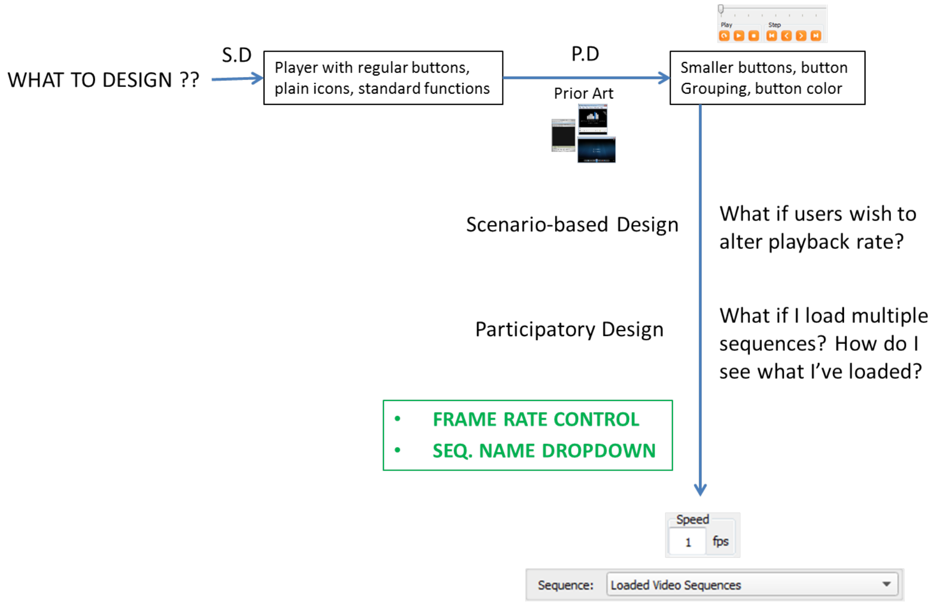

WHAT TO DESIGN ??
S.D
Player with regular buttons, plain icons, standard functions
P.D
Prior Art
Play
Step
Smaller buttons, button Grouping, button color
Scenario-based Design
What if users wish to alter playback rate?
Participatory Design
What if I load multiple sequences? How do I see what I've loaded?
FRAME RATE CONTROL
SEQ. NAME DROPDOWN
Speed
1
fps
Sequence:
Loaded Video Sequences

Figure 2.24: Evolution of the KOLAM-Loop User Interface Tool (VI). In response to the S.D. need of users to alter playback rate, and the P.D. need of users to access individual sequence details if multiple sequences are loaded, the frame rate control and sequence name dropdown widgets were added to KOLAM-Loop.

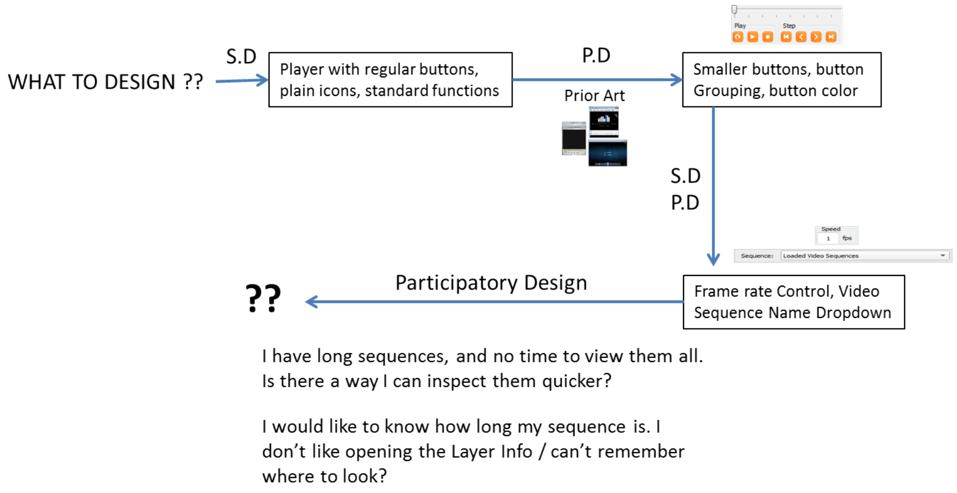
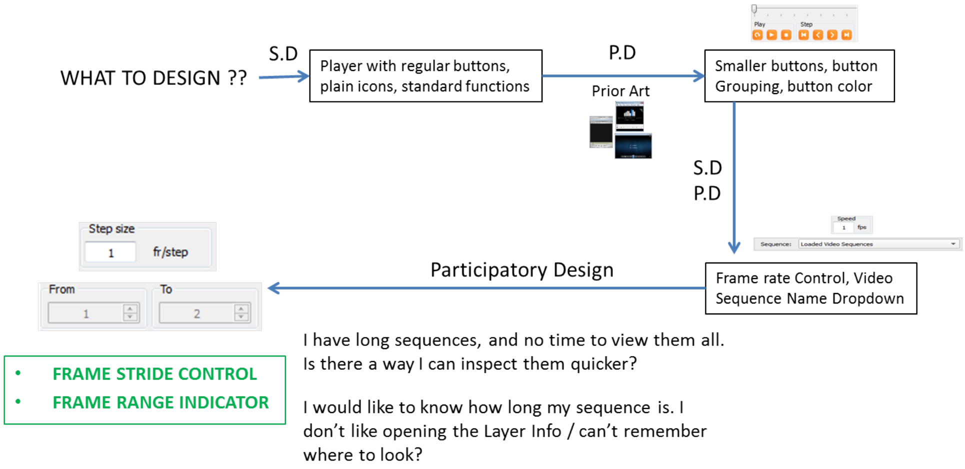

WHAT TO DESIGN ??
S.D
Player with regular buttons, plain icons, standard functions
P.D
Prior Art
Play
Step
Smaller buttons, button Grouping, button color
S.D
P.D
Speed
1
fps
Sequence:
Loaded Video Sequences
??
Participatory Design
Frame rate Control, Video Sequence Name Dropdown
I have long sequences, and no time to view them all. Is there a way I can inspect them quicker?
I would like to know how long my sequence is. I don't like opening the Layer Info / can't remember where to look?

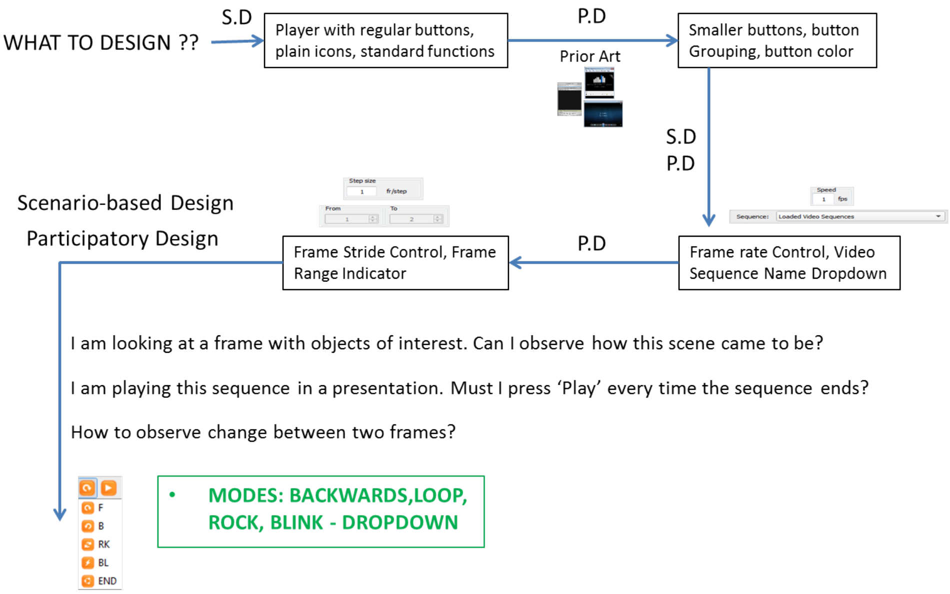
Figure 2.25: Evolution of the KOLAM-Loop User Interface Tool (VII). Participatory Design brings to light 2 issues: Given sequences of greater length, how can the users inspect them quicker? Next, how can info about length of the sequence be easily accessed?



I have long sequences, and no time to view them all. Is there a way I can inspect them quicker?

I would like to know how long my sequence is. I don't like opening the Layer Info / can't remember where to look?

Figure 2.26: Evolution of the KOLAM-Loop User Interface Tool (VIII). In response to the P.D. issues, the frame stride control and frame range indicator widgets were added to KOLAM-Loop.


WHAT TO DESIGN ??
S.D
Player with regular buttons, plain icons, standard functions
P.D
Prior Art
Smaller buttons, button Grouping, button color
S.D
P.D
Speed
1
fps
Sequence:
Loaded Video Sequences
Frame rate Control, Video Sequence Name Dropdown
P.D
Step size
1
fr/step
From
1
To
2
Frame Stride Control, Frame Range Indicator
Scenario-based Design
Participatory Design
F
B
RK
BL
END
MODES: BACKWARDS,LOOP, ROCK, BLINK - DROPDOWN

I am looking at a frame with objects of interest. Can I observe how this scene came to be?

I am playing this sequence in a presentation. Must I press 'Play' every time the sequence ends?

How to observe change between two frames?

Figure 2.28: Evolution of the KOLAM-Loop User Interface Tool (X). In response to S.D. and P.D. feedback, the alternate play modes 'Bkwd', 'Loop', 'Rock' and 'Blink' were added and were compactly grouped together in a drop-down within KOLAM-Loop.

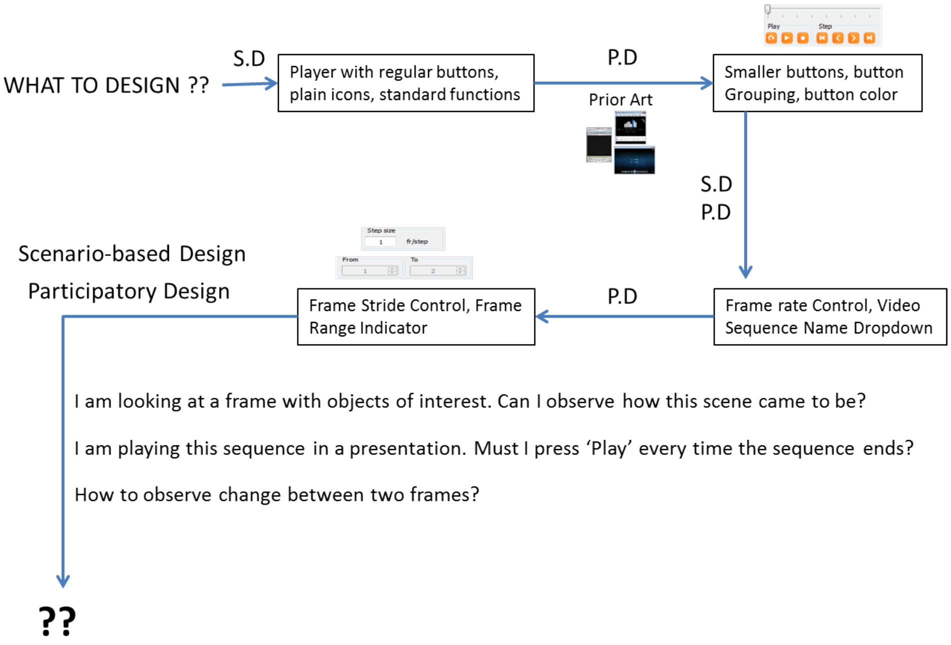

WHAT TO DESIGN ??
S.D
Player with regular buttons, plain icons, standard functions
P.D
Prior Art
Play
Step
Smaller buttons, button Grouping, button color
S.D
P.D
Frame rate Control, Video Sequence Name Dropdown
P.D
Step size
fr/step
From
To
Frame Stride Control, Frame Range Indicator
Scenario-based Design
Participatory Design
I am looking at a frame with objects of interest. Can I observe how this scene came to be?
I am playing this sequence in a presentation. Must I press 'Play' every time the sequence ends?
How to observe change between two frames?
??

Figure 2.27: Evolution of the KOLAM-Loop User Interface Tool (IX). Feedback from both Scenario and Participatory Design methods highlight the need for alternate modes of data playback.

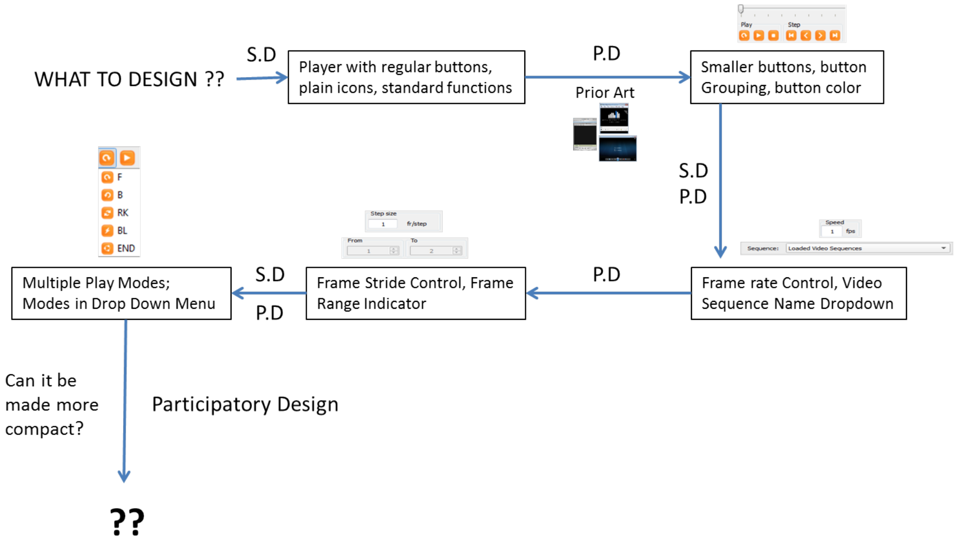

WHAT TO DESIGN ??
S.D
Player with regular buttons, plain icons, standard functions
P.D
Prior Art
Play
Step
Smaller buttons, button Grouping, button color
S.D
P.D
Speed
fps
Sequence:
Loaded Video Sequences
Frame rate Control, Video Sequence Name Dropdown
P.D
Step size
fr/step
From
To
Frame Stride Control, Frame Range Indicator
S.D
P.D
F
B
RK
BL
END
Multiple Play Modes; Modes in Drop Down Menu
Can it be made more compact?
Participatory Design
??

Figure 2.29: Evolution of the KOLAM-Loop User Interface Tool (XI). All tool features are now available. Participatory Design brings in the final requirement of making KOLAM-Loop more visually compact.

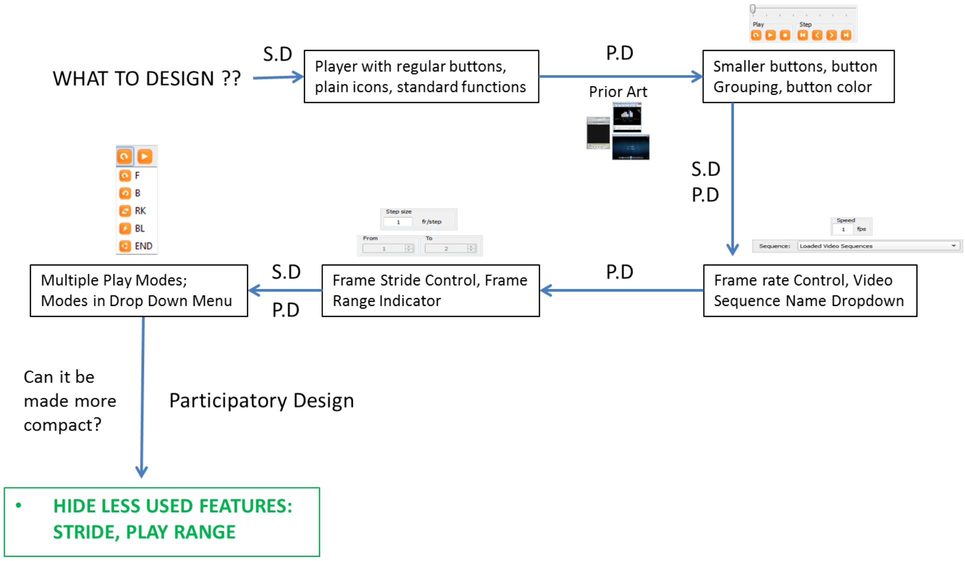

WHAT TO DESIGN ??
S.D
Player with regular buttons, plain icons, standard functions
P.D
Prior Art
Smaller buttons, button Grouping, button color
S.D
P.D
Frame rate Control, Video Sequence Name Dropdown
P.D
Frame Stride Control, Frame Range Indicator
S.D
P.D
Multiple Play Modes; Modes in Drop Down Menu
F
B
RK
BL
END
Can it be made more compact?
Participatory Design
HIDE LESS USED FEATURES: STRIDE, PLAY RANGE

Figure 2.30: Evolution of the KOLAM-Loop User Interface Tool (XII). In response to the P.D. requirement, the stride and frame range features were identified as less used. These are thus candidates for hiding from default view, thereby allowing for visual compactness.

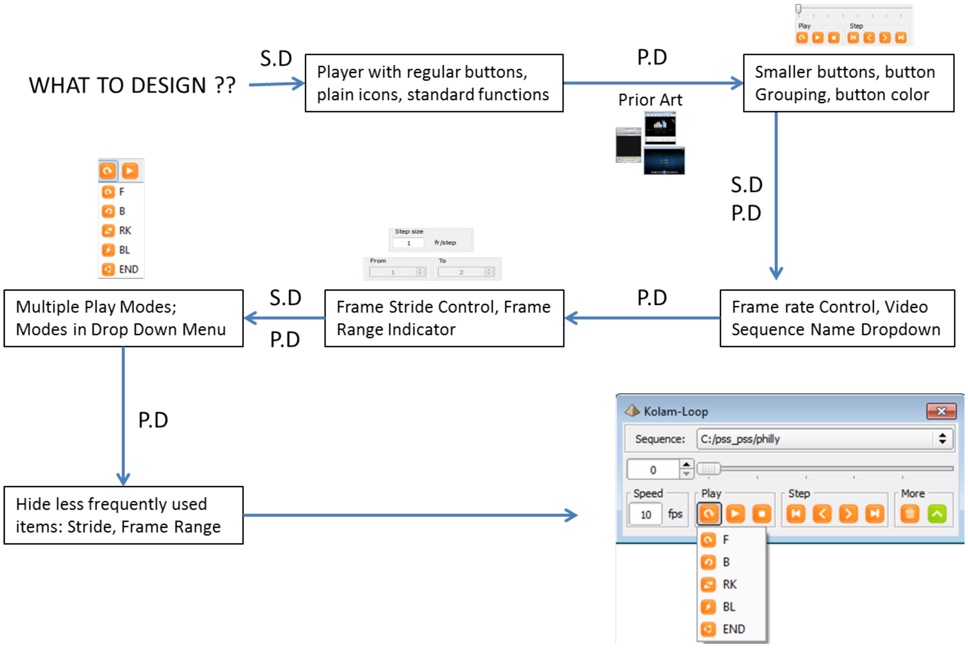

WHAT TO DESIGN ??
S.D
Player with regular buttons, plain icons, standard functions
P.D
Prior Art
Smaller buttons, button Grouping, button color
S.D
P.D
Frame rate Control, Video Sequence Name Dropdown
P.D
Frame Stride Control, Frame Range Indicator
S.D
P.D
Multiple Play Modes; Modes in Drop Down Menu
F
B
RK
BL
END
P.D
Hide less frequently used items: Stride, Frame Range
Kolam-Loop
Sequence: C:/pss_pss/philly
0
Speed
10 fps
Play
Step
More
F
B
RK
BL
END

Figure 2.31: Evolution of the KOLAM-Loop User Interface Tool (XIII). Once the less used stride and frame range widgets are hidden from default view, we obtain the final form of KOLAM-Loop.

The **Take-Away Lessons** from KOLAM's HCI component design process are now enumerated below:

- Design and Development Time: The total development time for the KOLAM-Loop tool was approximately 6 months. Multiple iterations were involved, and each successive iteration was of a longer duration than the previous one, as the tool matured and users became more used to the tool features.
- Aspects of User Behavior: At every step, it is crucial to reconcile user expectations with the actual system requirements. An important point to note is that at times, users 'do not know what they want'. This may be elaborated upon as follows: Users may not give useful information during the requirements gathering process, they may oppose certain features based on their negative past experiences (with KOLAM or even with other tools) or other factors, and and may make do with sub-optimal feature set without raising flags about factors potentially reducing their work efficiency.

2.6 Porting, Version Control and Multi-platform Support

This portion of the chapter describes the tasks needed to be performed in order to 'morph' KOLAM 2.0 as it was, into the starting point for KOLAM 3.0. They are as follows, with descriptions following thereafter: (a) Porting of KOLAM from Qt 3 to Qt 4, moving through several versions of Qt 4, and finally porting KOLAM from Qt 4 to Qt 5, (b) Creation of a version-controlled source-code repository for KOLAM, which has been continuously updated to the current day, and (c) Support for KOLAM on multiple OS platforms - specifically, the Windows, Linux and Mac OSX systems. It is important to note that these were not one-time tasks: they either needed to be continuously performed throughout KOLAM

3.0's life cycle, or at intervals coinciding with the release of a stable and more enhanced version of the Qt API.

In moving from KOLAM 2.0 to KOLAM 3.0, the first task completed was the migration of KOLAM from Qt version 3 (Qt 3) to Qt 4. This version change of the Qt GUI development API was extremely significant: the majority of all classes were affected. This change was felt in the KOLAM 2.0 codebase as well, as member functions of several derived or re-implemented classes were completely deprecated, in some cases completely removed. Mapping of the old functionality to the new was provided in a number of cases, but not all. Challenges in the porting process were ameliorated somewhat by the tool 'qt3to4', provided by the creators of Qt and which semi-automated some of the porting. The tool did not handle those classes which were completely deprecated and replaced: these needed to manually addressed over a lengthy period of time. In the time following the port and after several features had been added to KOLAM 3.0, porting was again performed in moving between minor Qt versions; specifically Qt 4.4.3, Qt 4.6.3 and Qt 4.7.4. While the final minor version of Qt 4 was Qt 4.8, stability issues pertaining to the drawing classes prevented its adoption until Qt 5 was released. In fact, the most stable version (for all three OSes mentioned above) of KOLAM 3.0 is built using Qt 4.7.4. As with the Qt 4.8 releases, the early Qt 5 releases were not satisfyingly stable either, which delayed the porting of KOLAM 3.0 to Qt 5. The Qt minor version release Qt 5.3.2 was to the first relatively stable version of Qt 5; which motivated KOLAM 3.0 being ported to Qt 5.3.2 for the multiple OSes. KOLAM was then ported to Qt 5.6.1. The final port of KOLAM prior to the author's Dissertation Defense was to Qt 5.7.0.

The KOLAM 3.0 code base was set up as a repository using the Subversion (SVN) version control system. Updated at regular intervals, the source trunk has gone through several hundred code commits to date. Taking advantage of Qt being a cross-platform UI development API, the author has maintained KOLAM 3.0 for all three major OSes, with varying degrees of success. Development tools and environments have been consistent under the Windows and Linux environments, with a few issues needing to be addressed. Porting for Mac OSX, however, has been a challenge from day 1. Factors contributing to this include the closed source development model of Apple, with non-native graphical APIs being denied direct access to system resources unlike the Cocoa and Carbon APIs; sporadic documentation and ad-hoc solutions for Mac OSX specific issues. Problems encountered include font resizing errors, widget layout overflow/underflow, inconsistent focus stealing by child windows, inconsistencies with inter-session saving of settings and inconsistencies with the trajectory drawing, to name the most prominent.

2.7 Scope of Research on KOLAM

The remainder of this thesis details the problems and challenges tackled; as well as the author's solutions and unique contributions to these, in KOLAM. Three problems in three different application domains were tackled and are presented as case studies.

Chapter 3 presents the 1st Case Study; involving the problems and challenges faced in automated, manual and assisted object tracking in wide-area motion imagery (WAMI), as well as the solutions. Chapter 4 presents the 2nd Case Study, and details the trajectory visualization and editing features that were implemented in KOLAM. Chapter 5 presents the 3nd Case Study; involving the problems and challenges pertaining to guiding the output of epithelia-stroma segmentation and classification algorithms when applied to histopathology whole-slide imagery (WSI), when examining the role played by stroma in determining short- and long-term breast cancer patient prognoses.

Chapter 6 provides details about KOLAM extensions for tiled wall displays. This covers aspects common to all three case studies that had to be uniquely extended and/or modified to fit this new display paradigm. In order for a visual analytics system (whose front end is entirely UI based) to be considered reliable enough to use by a wider audience, it must be comprehensively evaluated. Chapter 7 does exactly this, and involves the discussion about quantitative and qualitative evaluation (ie Usability Testing) of KOLAM. Finally, Chapter 8 outlines directions for possible future work, derived from the directions which were unexplored (or partially explored) in this thesis; and directions for the implementation of these ideas within the KOLAM framework.

Chapter 3

CASE STUDY 1: Object Tracking

3.1 Motivation

As mentioned in Chapters 1 and 2, performing object tracking on different WAMI datasets was the primary driving force for the definition of the problem statement and subsequently identifying the associated challenges by the author. In devising the solutions to these, the author conceived, designed and implemented all the object tracking related functionality in KOLAM 3.0. In its present form, KOLAM 3.0 allows for the exploratory analysis of WAMI data sets, thereby providing dense spatio-temporal coverage of wide field-of-view urban regions. This makes the system useful for applications requiring accurate multiple target tracking, scalable to very large numbers of objects. Besides addressing our immediate research needs, KOLAM 3.0's object tracking capabilities were motivated by needs from both the civilian and defense sectors. On the civilian side, the need by law enforcement agencies for software capable of forensic traffic analysis and video summarization features, the need of city planners for urban planning based on analysis of traffic patterns and existing buildings and emergency response planning needs continue to motivate the maintenance and addition of features to KOLAM 3.0's object tracking feature set. In a similar vein, the tasks of improving situational awareness, persistent target observation, reconnaissance, force protection and rapid targeting and response in the defense context also continue to motivate object tracking feature development in KOLAM 3.0.

3.2 Introduction

Tracking in wide-area motion imagery is a challenging research domain that is receiving a lot of current interest. The visualization of hundreds to thousands of tracks resulting from automated and manual tracking of objects offers new challenges in visualization and visual analytics. Even with standard video sequences, meaningful comparisons of tracking algorithm behavior with quantitative performance metrics were difficult to perform due to the paucity of standard video datasets with associated manual labeled groundtruth. However, recent work has led to the creation of extensive, open repositories of *non-wide area* video datasets, tools and ground-truth. Prominent examples include open source tools such as ViPER-GT [50], which allow for concept selection, ground truth generation and annotation of video; the ViSOR project [51], which comprises a dynamic, open repository of annotated surveillance video sequences accessible via a web interface; and the Scoring, Truthing And Registration Toolkit (START), for semi-automated ground-truth generation using a keyframe approach [52]. Several workshops, such as the PETS series and the VSSN series, and national-level projects, such as I-LIDS [53] and ETISEO [54], utilize the ViPER-XML annotation format in their video databases. Another important example is the ground-truth motion database developed along with the layer segmentation and motion annotation tools at MIT CSAIL [55].

Given the established importance of standard video annotation ground-truth databases and performance metrics, increasing attention is currently being focused in the domain of wide-area motion imagery (WAMI). Wide-area surveillance first emerged as a new area of interest around 10 years ago and continues to be a highly active research area, due to its large scale continuous coverage of urban regions for a variety of applications [10, 56]. Working with WAMI data presents a unique set of challenges, not common to standard video surveillance data, including interactive visualization of very large time-sequence imagery, efficient algorithms for mosaicing, georegistration, stabilization and tracking [10] [57]. The difficulties of tracking vehicles in low frame rate aerial wide-area motion imagery are many, including large object displacements, parallax and occlusions from tall structures, low contrast, moving seams across cameras, significant object appearance changes with viewing direction that are described in recent publications including [10, 57–61]. Unlike regular video surveillance databases, only few WAMI datasets are available in the public domain. One example is the Columbus Large Image Format (CLIF) dataset, collected in a flyover of the Ohio State University Campus in October 2007 [62]. Visualizing the statistical distribution of a dense set of automatically estimated tracks in a limited geographical region by overlaying a large number of trajectories is described in [60, 61]. The WAMI database utilized here for visualization and ground-truth generation purposes, is the Persistent Surveillance Systems (PSS) database of event management, law enforcement and emergency response video collections.

WAMI datasets were collected using an eight camera array on an airborne platform producing 256 megapixel mosaiced georegistered images. The example described here involves the WAMI PSS data consisting of several thousand frames collected over Philadelphia (March 13, 2008). Some of the features of the Philadelphia WAMI dataset are listed in Table 3.1.

Frame Rate:	1 frame per second (fps)
Altitude:	3,500 – 12,500 ft.
Coverage:	4 square miles, 80°x 60°fov
GSD:	25 – 50 cm
Pixel Type:	Grayscale
Bandwidth:	1 TB/hr; 16K x 16K pixels / frame
File Format:	Tiled JPEG pyramids

Table 3.1: WAMI North Philadelphia, Pennsylvania dataset characteristics (Datasets courtesy of PSS).

3.3 KOLAM Interface for Visualization and Tracking

KOLAM allows users to interact with and manipulate a time sequence of very large imagery in pyramid format (or same-sized image files in TIFF, PNG, JPEG, etc.) in an exploratory fashion. In addition to image visual analytics, KOLAM also supports the interactive visualization of extremely large, spatially and spectrally varying geospatial imagery rendered as a 3D globe. KOLAM uses the Qt application programming interface and UI framework to provide interactive tracking and management of trajectories for WAMI datasets, as well as the ability to interface with and invoke external tracking algorithms.

3.3.1 Visualization of Multiple Layers of Analytic Information

KOLAM can simultaneously display and combine multiple layers of raster or vector information interactively to produce composite analytic visualizations using a layer selection interface. A layer is a high level visual representation of a dataset that encapsulates both the image information and relevant metadata [38]. It is possible to simultaneously visualize multiple layers, and each layer may have one each of several types of viewers and navigators associated with it. The user can interactively move a given layer up or down the layer-stack currently being visualized to control visibility and blending of layers. Since the portion of a given layer that is occluded depends on its position on the overall stack of layers, this feature allows for rapid control of the visibility of a large number of layers. Additionally, alpha blending enables exploratory visualization by combining different layers. The user can interactively switch between applying a global navigation transformation to affect all visible layers of information or apply a local transformation to align each image in the layer which is a useful property for interactive mosaicing and georegistration.

3.3.2 Visualizing Motion Imagery

KOLAM provides a highly compact user interface via the KOLAM-Loop tool with a readily usable set of functionality that comprehensively addresses user needs when playing back WAMI sequences (as well as other sequences) of image data. KOLAM supports playback of a given sequence, in any direction, with or without looping, and with on-the-fly frame rate adjustment. The user can choose to step through the sequence one frame at a time (with a skip/stride factor), or jump to an arbitrary frame. The user can also construct collections of related time-varying data and switch between them. This enables comparison and collation of related motion imagery sequences that may be spread over multiple datasets.

3.3.3 Tracking and Trajectory Visualization Subsystems

The tracking subsystem in KOLAM was specifically designed and implemented to accommodate the visual analytic needs of tracking objects of interest in both WAMI datasets as well as more general video sequences. The KOLAM-Tracker tool accomplishes this by providing a comprehensive environment for manual ground-truth (target) annotation, rapid, simultaneous tracking of multiple objects, editing of trajectories, and assisted tracking. Sample manual and automatic tracking results for two different data sets are shown in Figure 3.1. The 'master' Target Tracking algorithm (**Algorithm 5**) involves the invocation of either the automatic tracking algorithm, the visual tracking algorithm or the assisted tracking algorithm; depending on the needs of the user.

3.4 Tracking types in KOLAM 3.0

The Kolam-Tracker tool (Figure 2.12) supports three modes of operation: fully automatic tracking, manual tracking (or ground-truth generation) and a mixed mode of assisted tracking.

3.4.1 Track file formats supported by KOLAM

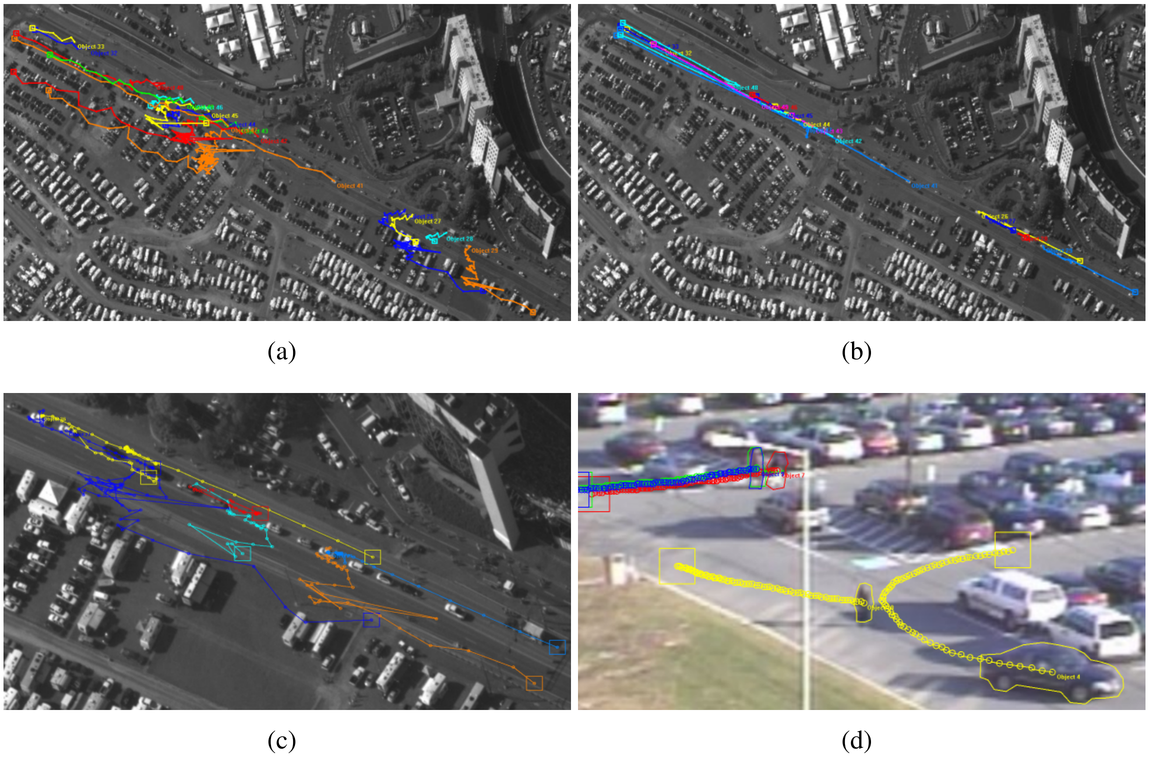
KOLAM provides support for two track file formats - an in-house designed *flat file format*, and the *KW-18 file format*, developed by Kitware Inc and adopted at CIVA lab for collaborative data sharing efforts. The specifications for both formats are presented below. Users desiring to have their programs output track files in either format for display in KOLAM are advised to **strictly adhere to the format details** presented in the *illustrations*

Algorithm 5: Target Tracking Algorithm

Input : Image sequence I, task to track object(s) purely via **automated methods**, **manual intervention**, or a **combination of both**; based on pre-defined spatio-temporal parameters, tracker program(s)

Output: *Tracker/User/Combined*-generated **trajectories**; loaded in memory (data structure) and visualized by KOLAM

```
TrajectoryPoint(x, y, ...);
Trajectory(ptList list¡TrajectoryPoint ¿, color, thickness);
TrajectoryDS(list¡Trajectory ¿, ...);
while User still needs to track objects in the data do
    if Object tracked purely via Tracking Algorithms then
        Perform_Automatic_Tracking();
    end
    if Object tracked purely via Manual Intervention then
        Perform_Visual_Tracking();
    end
    if Object is tracked via Combination of Both then
        Human_In_The_Loop_Tracking();
    end
    if Object Trajectory needs to be Modified then
        Perform_Track_Editing();
    end
    if No more objects to be tracked then
        break;
    end
end
Return to KOLAM's main UI Loop;
```



(a)

(b)

(c)

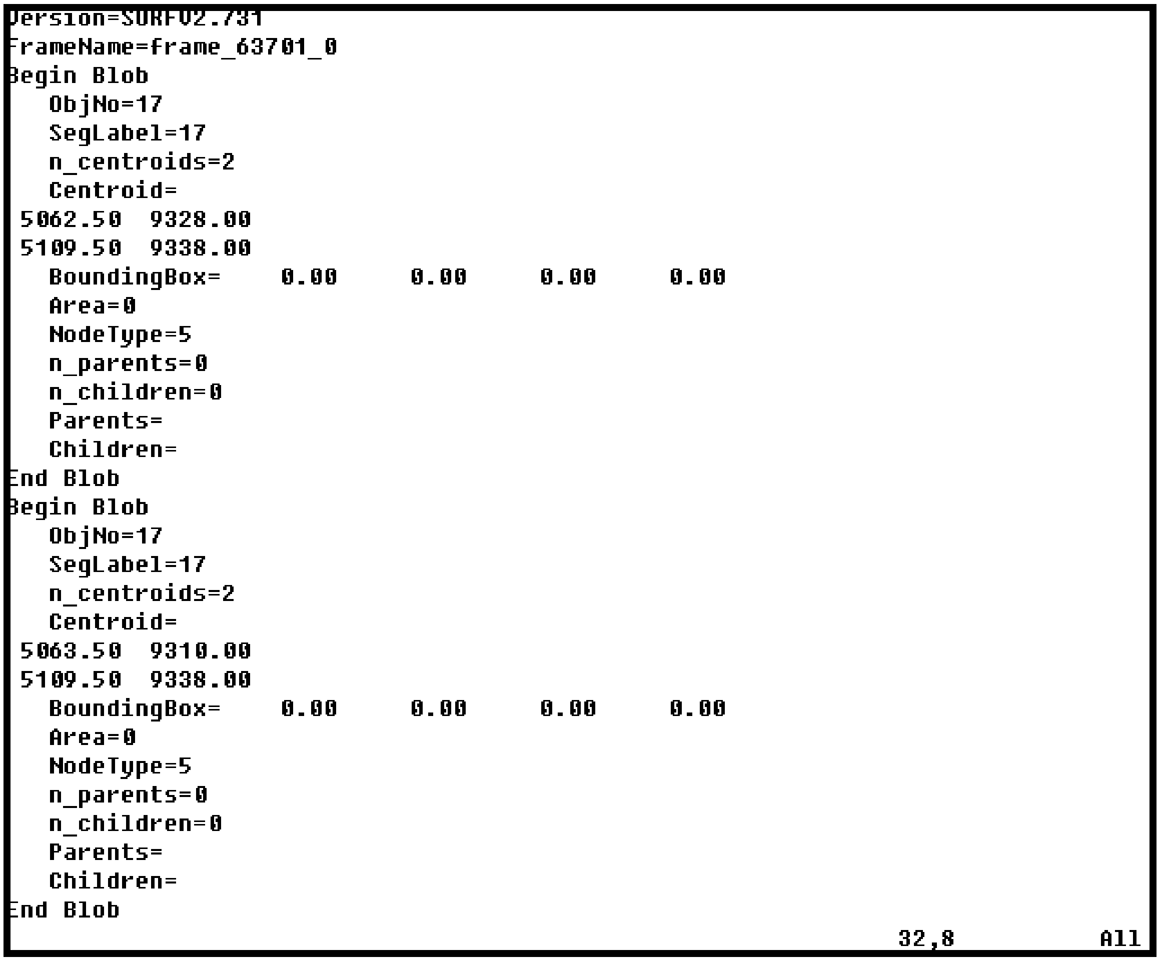
(d)

Figure 3.1: Visualization of multiple tracked objects and their trajectories in KOLAM using different vector primitives. The images show: (a) Tracked vehicle trajectories without local registration (*i.e.* unstabilized), (b) Vehicle trajectories that are locally registered (*i.e.* stabilized), (c) Unstabilized and stabilized pairs of vehicle trajectories, and (d) Ground-truth polygons marked for tracked objects. The first three images display results for a PSS WAMI dataset collected over Charlotte, NC and the last shows ground-truth for a non-WAMI dataset (Army Research Lab FPSS [63]).

and *descriptions*.

Flat file format

The flat file format is an **unstructured** (vis-a-vis structured formats like XML) file format (file extension - **.txt**) that was **designed specifically to handle the unique challenges** presented by certain wide-area datasets (extreme parallax, poor registration, wrong elevation model, inferior IMU etc.). For every trajectory, *each time step is represented by a* *single file*, that contains the centroids on the trajectory upto that particular time step. In other words: if a trajectory has a length of 100 time steps, the flat file system associates 100 files with this ***single*** trajectory. Unfortunately, such a data management system is the *only means* of addressing a scenario in which *all centroids on the trajectory are different for every time step* (the non-registered nature of the data necessitates that the trajectories themselves be registered). The structure of KOLAM's flat files is illustrated in Figure 3.2.



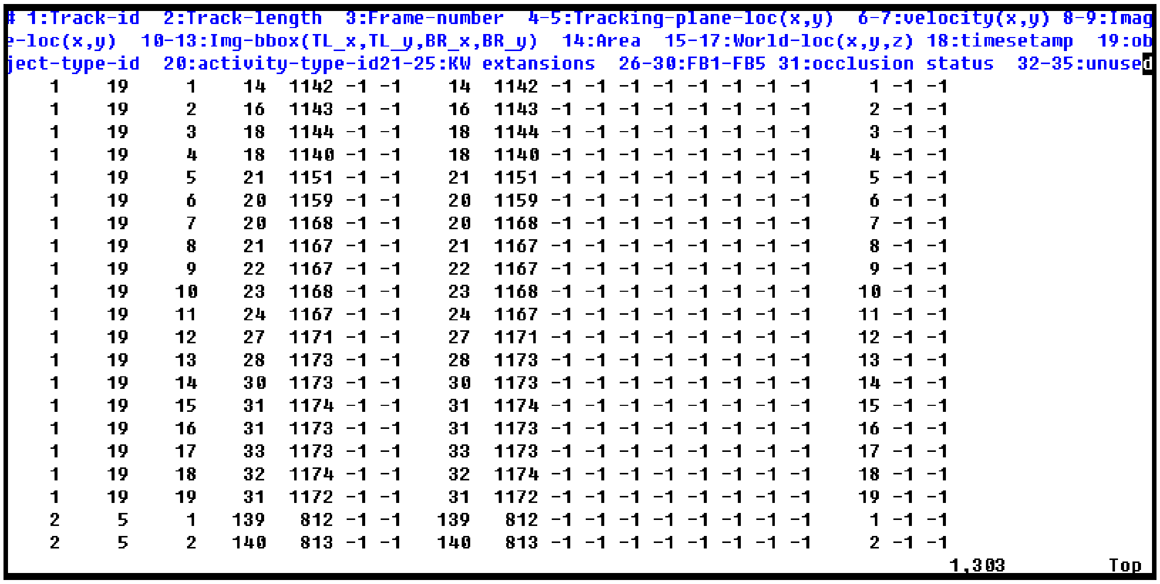
```
Version=SURFV2.731
FrameName=frame_63701_0
Begin Blob
   ObjNo=17
   SegLabel=17
   n_centroids=2
   Centroid=
 5062.50  9328.00
 5109.50  9338.00
   BoundingBox=    0.00      0.00      0.00      0.00
   Area=0
   NodeType=5
   n_parents=0
   n_children=0
   Parents=
   Children=
End Blob
Begin Blob
   ObjNo=17
   SegLabel=17
   n_centroids=2
   Centroid=
 5063.50  9310.00
 5109.50  9338.00
   BoundingBox=    0.00      0.00      0.00      0.00
   Area=0
   NodeType=5
   n_parents=0
   n_children=0
   Parents=
   Children=
End Blob
                                                                  32,8          All
```

Figure 3.2: Screen shot of a typical KOLAM flat file, for *one time step* on a trajectory.

All lines in a flat file that lie with a **'Begin Blob' - 'End Blob'** block correspond to information pertinent to the Object-ID indicated by the **ObjNo** field, for *this particular time step*. Thus, the flat file illustrated in Figure 3.2 contains data for two trajectories (since there are **2 Blobs**) corresponding to Object-ID '17', for its 2nd time step (indicated by the number of (*x*,*y*) pairs under the **Centroid** field). While all fields after the 'BoundingBox' field must be written out while creating the flat track file(s), these fields merely have a nominal presence in the file. They are not currently used by KOLAM for drawing trajectories.

KW-18 file format

The KW-18 file format is a **semi-structured** file format (file extension - **.kw18**). Its data organization scheme allows multiple trajectories representing a complete tracking database to be stored in a single file. A KW-18 file is partly illustrated in Figure 3.3.



```
# 1:Track-id  2:Track-length  3:Frame-number  4-5:Tracking-plane-loc(x,y)  6-7:velocity(x,y) 8-9:Imag
e-loc(x,y)  10-13:Img-bbox(TL_x,TL_y,BR_x,BR_y)  14:Area  15-17:World-loc(x,y,z) 18:timesetamp  19:ob
ject-type-id  20:activity-type-id21-25:KW extansions  26-30:FB1-FB5 31:occlusion status  32-35:unused
    1     19     1    14  1142 -1 -1    14  1142 -1 -1 -1 -1 -1 -1 -1 -1     1 -1 -1
    1     19     2    16  1143 -1 -1    16  1143 -1 -1 -1 -1 -1 -1 -1 -1     2 -1 -1
    1     19     3    18  1144 -1 -1    18  1144 -1 -1 -1 -1 -1 -1 -1 -1     3 -1 -1
    1     19     4    18  1140 -1 -1    18  1140 -1 -1 -1 -1 -1 -1 -1 -1     4 -1 -1
    1     19     5    21  1151 -1 -1    21  1151 -1 -1 -1 -1 -1 -1 -1 -1     5 -1 -1
    1     19     6    20  1159 -1 -1    20  1159 -1 -1 -1 -1 -1 -1 -1 -1     6 -1 -1
    1     19     7    20  1168 -1 -1    20  1168 -1 -1 -1 -1 -1 -1 -1 -1     7 -1 -1
    1     19     8    21  1167 -1 -1    21  1167 -1 -1 -1 -1 -1 -1 -1 -1     8 -1 -1
    1     19     9    22  1167 -1 -1    22  1167 -1 -1 -1 -1 -1 -1 -1 -1     9 -1 -1
    1     19    10    23  1168 -1 -1    23  1168 -1 -1 -1 -1 -1 -1 -1 -1    10 -1 -1
    1     19    11    24  1167 -1 -1    24  1167 -1 -1 -1 -1 -1 -1 -1 -1    11 -1 -1
    1     19    12    27  1171 -1 -1    27  1171 -1 -1 -1 -1 -1 -1 -1 -1    12 -1 -1
    1     19    13    28  1173 -1 -1    28  1173 -1 -1 -1 -1 -1 -1 -1 -1    13 -1 -1
    1     19    14    30  1173 -1 -1    30  1173 -1 -1 -1 -1 -1 -1 -1 -1    14 -1 -1
    1     19    15    31  1174 -1 -1    31  1174 -1 -1 -1 -1 -1 -1 -1 -1    15 -1 -1
    1     19    16    31  1173 -1 -1    31  1173 -1 -1 -1 -1 -1 -1 -1 -1    16 -1 -1
    1     19    17    33  1173 -1 -1    33  1173 -1 -1 -1 -1 -1 -1 -1 -1    17 -1 -1
    1     19    18    32  1174 -1 -1    32  1174 -1 -1 -1 -1 -1 -1 -1 -1    18 -1 -1
    1     19    19    31  1172 -1 -1    31  1172 -1 -1 -1 -1 -1 -1 -1 -1    19 -1 -1
    2      5     1   139   812 -1 -1   139   812 -1 -1 -1 -1 -1 -1 -1 -1     1 -1 -1
    2      5     2   140   813 -1 -1   140   813 -1 -1 -1 -1 -1 -1 -1 -1     2 -1 -1
                                                                                 1,303          Top
```

Figure 3.3: Screen shot of a typical KW-18 file, for *ALL trajectories*.

Each trajectory in a KW-18 file is represented by a *contiguous group* of **lines** arranged in *increasing order* of **time**. Such trajectories are then arranged in *increasing order* of

Track-ID. Each line represents a single time step on the trajectory, and stores multiple pieces of information about that single time step in a *space-separated* fashion. All **20 data columns** (see Figure 3.3) are **mandatory**.

Examining Figure 3.3, additional details become apparent. First, fields (**4,5**) and (**8,9**) are identical, and field (**18**) is identical to field (**3**). Second, *ALL* fields for which data are unavailable are populated with the placeholder '**-1**'.

3.4.2 Automatic Tracking

Support for visualization and analysis of the results of automated tracker execution in KOLAM was added to support the algorithm development effort for automated tracking of objects of interest in low frame rate wide area motion imagery. The automatic tracking mode refers to the tracking of objects solely through the invocation of external tracker software such as CSURF, LOFT or *any other program* that has been configured to accept KOLAM 3.0's input parameter format. The loose coupling that was created between KOLAM 3.0 and external tracker software such as CSURF and LOFT, and the simple means of input generation and tracker invocation from within KOLAM 3.0, together form a system that is among the first of its kind: the survey of relevant literature reveals that competing tracker programs have only been tested within the limited environment of their development: for example, trackers developed using MATLAB or OCTAVE have test results and tables derived from MATLAB and OCTAVE simulations. The strength of such environments lies in rapid algorithm prototyping, and not in providing the user with a scalable, extensible interface for visualizing ultra-high resolution WAMI imagery, interactively executing the tracker algorithms and performing visual and analytic result inspection while interfacing with such a system. In this regard, the conception and implementation of KOLAM 3.0's Automatic Tracking capability, and the testing and refinement of trackers such as CSURF and LOFT is a unique contribution to the field of object tracking in WAMI imagery.

The various tracker and trajectory properties set via Kolam-Tracker prior to tracker initialization are encapsulated in the specific container class, instances of which are created for each distinct target. The data and control flows between Kolam-Tracker and the automatic tracking algorithms are depicted in Figure 3.4. The multi-step process of automatic tracking, which begins with Kolam-Tracker, is described below:

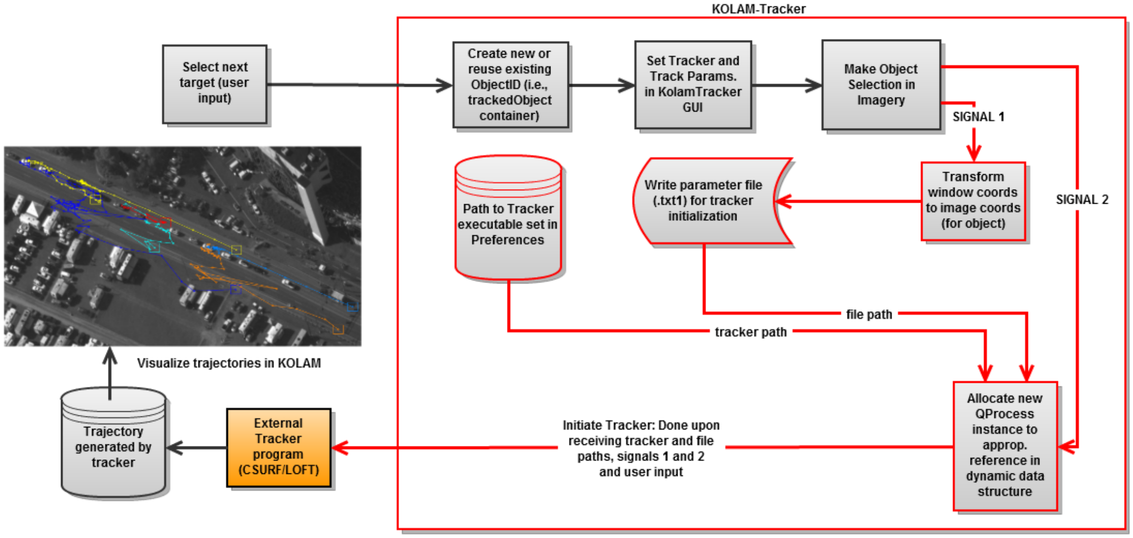

KOLAM-Tracker
Select next target (user input)
Create new or reuse existing ObjectID (i.e., trackedObject container)
Set Tracker and Track Params. in KolamTracker GUI
Make Object Selection in Imagery
SIGNAL 1
SIGNAL 2
Path to Tracker executable set in Preferences
Write parameter file (.txt1) for tracker initialization
Transform window coords to image coords (for object)
file path
tracker path
Visualize trajectories in KOLAM
Trajectory generated by tracker
External Tracker program (CSURF/LOFT)
Initiate Tracker: Done upon receiving tracker and file paths, signals 1 and 2 and user input
Allocate new QProcess instance to approp. reference in dynamic data structure

Figure 3.4: Automatic Tracking mode in Kolam-Tracker. Interaction steps and flow of processing in invoking an external tracker program for object tracking and trajectory visualization. The black boxes denote user interaction steps, and the red ones denote steps of algorithm execution.

- First, the user sets the tracker type (currently, either CSURF or LOFT) and draw primitive parameters (either point, bounding box or polygon) via the interface. Tracker selection initializes a temporary drawing overlay on the display area, which is utilized in the next step.

- Given that both CSURF and LOFT trackers expect object selection by means of bounding boxes, this draw primitive is assumed as default in all object selection descriptions henceforth. The user annotates a rectangular region around the object to be tracked on the drawing overlay. This enclosing bounding box may be re-drawn until the most satisfactory selection has been made. Each selected object is internally maintained in two parts. First is the drawable component, ie. the visual representation of the trajectory that will be generated by the tracker (under normal and favorable operational conditions) once its execution commences, and which is used by KOLAM's trajectory drawing class with visual properties dynamically modifiable by Kola-Tracker. Second is the process component: each object selected for tracking is internally maintained in a data structure of references to QProcess instances. This list is used to start and optionally stop auto-tracking program execution applied to multiple objects in a distributed processing fashion.

- The completion of object selection is an event which when received by Kolam-Tracker causes it to signal the start of execution of the selected tracker for the selected object. Automatic tracker invocations are initialized as forked processes on Unix-based systems or as independent child processes on Windows-based systems. This ensures that KOLAM at large is neither blocked, left in a wait state nor unpredictably interrupted by the external auto-tracker. This implies that KOLAM interactivity is independent of computationally intensive tracker while it is executing.

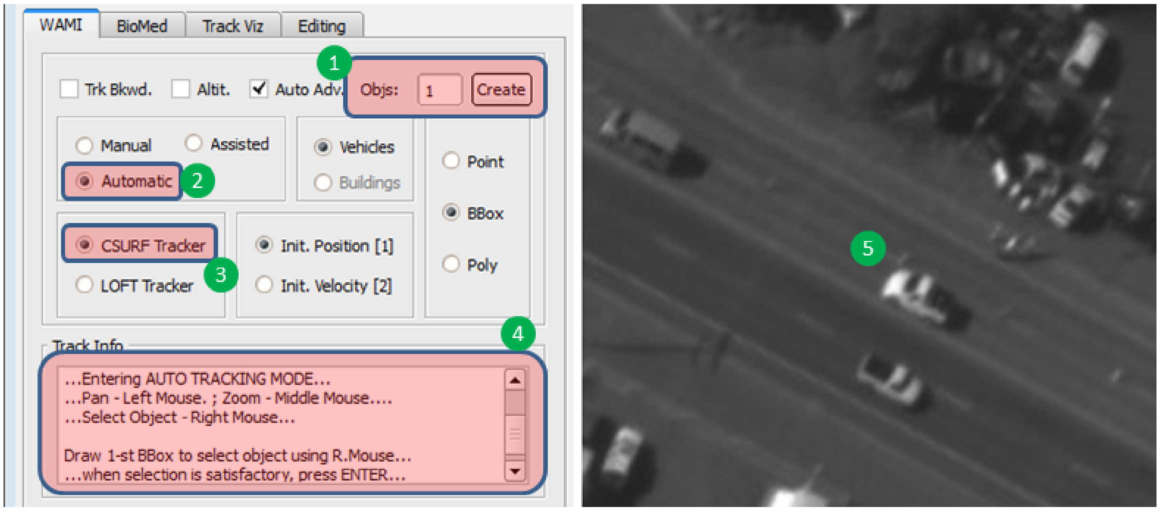
- The output of the tracker for each object is saved to disk and also incrementally updated in KOLAM's trajectory data structure for immediate, interactive review of the results. Object IDs that have been used to track specific objects may be used to track other objects, however this will result in the (prior) existing trajectory data being overwritten.

Algorithm 6 lists steps involving user-selected input, KOLAM-to-tracker message passing, the resulting asynchronous tracker invocation and finally the on-the-fly display of the generated results. The steps involved in the invocation and execution of the SURF Tracker are illustrated in Figures 3.5, 3.6 and 3.7.

Similarly, the steps involved in the invocation and execution of the LOFT Tracker are illustrated in Figures 3.8, 3.9, 3.10 and 3.11.

3.4.3 Manual Tracking

Support for manual track creation and manipulation was added in response to the need for effective ground truth generation capability, as well as to benchmark automated tracker performance.



WAMI | BioMed | Track Viz | Editing

☐ Trk Bkwd. ☐ Altit. ☑ Auto Adv. Objs: 1 Create

○ Manual ○ Assisted ● Automatic

● Vehicles ○ Buildings

○ Point ● BBox ○ Poly

● CSURF Tracker ○ LOFT Tracker

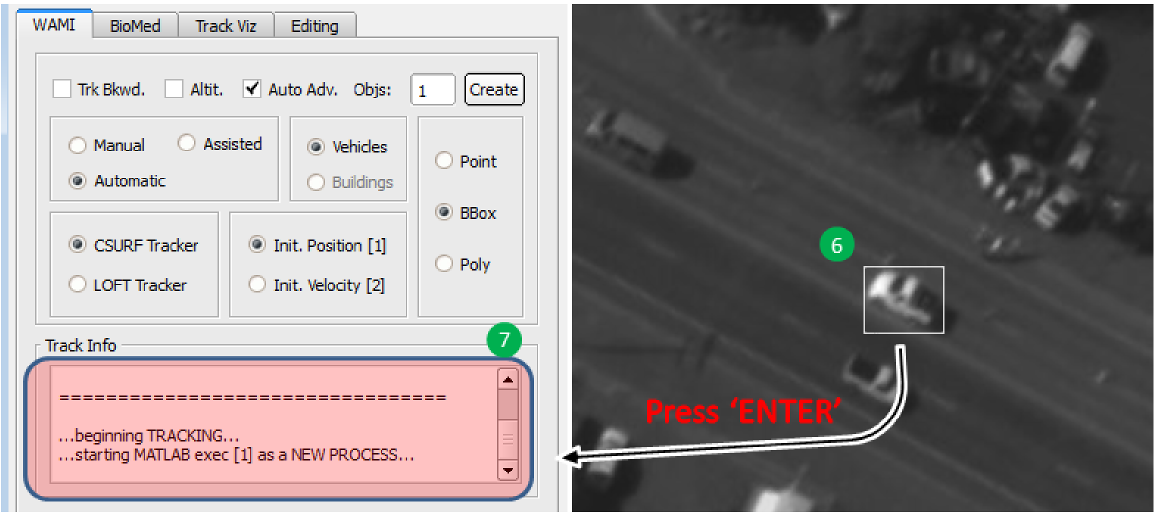
● Init. Position [1] ○ Init. Velocity [2]

Track Info

```
...Entering AUTO TRACKING MODE...
...Pan - Left Mouse. ; Zoom - Middle Mouse....
...Select Object - Right Mouse...

Draw 1-st BBox to select object using R.Mouse...
...when selection is satisfactory, press ENTER...
```

Figure 3.5: **Step I** of tracking an object using the CSURF Tracker. The user first *creates a new Object-ID* (Highlighted by rounded rectangle **1**). Creating a new Object-ID in KOLAM always *sets the Tracking Mode to Automatic* (**2**) and *sets CSURF as the selected tracker* (**3**). KOLAM *gives the user feedback and the next step to perform* in the tracking process (**4**). The object (vehicle in this case) to be tracked is also highlighted (**5**).



WAMI | BioMed | Track Viz | Editing

☐ Trk Bkwd. ☐ Altit. ☑ Auto Adv. Objs: 1 Create

○ Manual ○ Assisted ● Automatic

● Vehicles ○ Buildings

○ Point ● BBox ○ Poly

● CSURF Tracker ○ LOFT Tracker

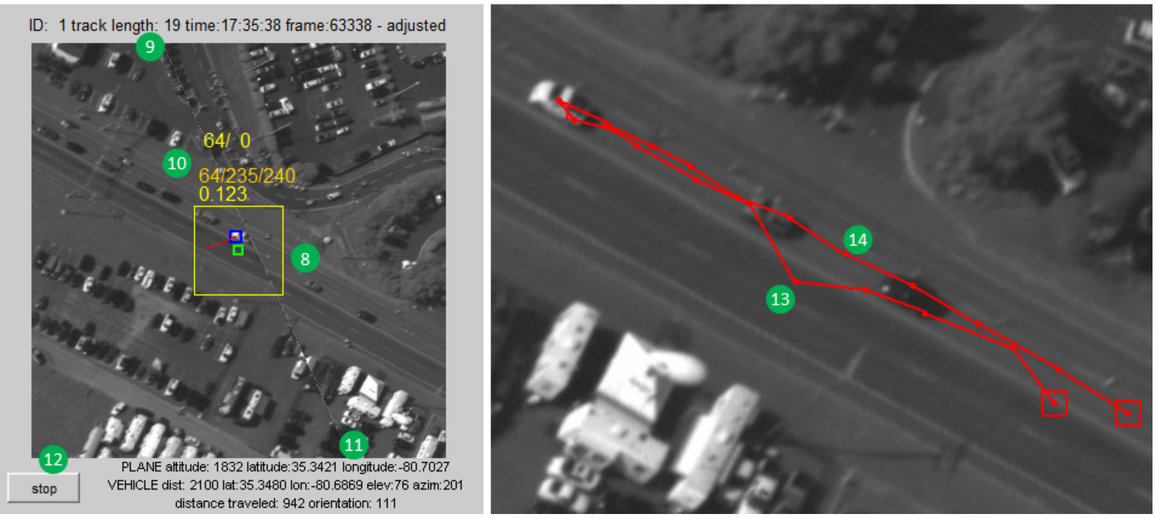
● Init. Position [1] ○ Init. Velocity [2]

Track Info

```
==================================

...beginning TRACKING...
...starting MATLAB exec [1] as a NEW PROCESS...
```

Figure 3.6: **Step II** of tracking an object using the CSURF Tracker. The user now draws a bounding box around the object to be tracked with the *right mouse button* (**6**). The drawn box must include the object and a *small* amount of background (again, see (**6**)). This box *may be re-drawn as many times as the user needs* to get it right. When satisfied with the box drawn, the user presses the 'ENTER' key. This action *ends the object selection phase* of tracking; *creates a .txt1 parameter file* containing information about the drawn box, dataset path, output file path etc. *and invokes the tracker program* as an external process, with the .txt1 file as a parameter. Feedback regarding the same is presented by KOLAM to the user (**7**).

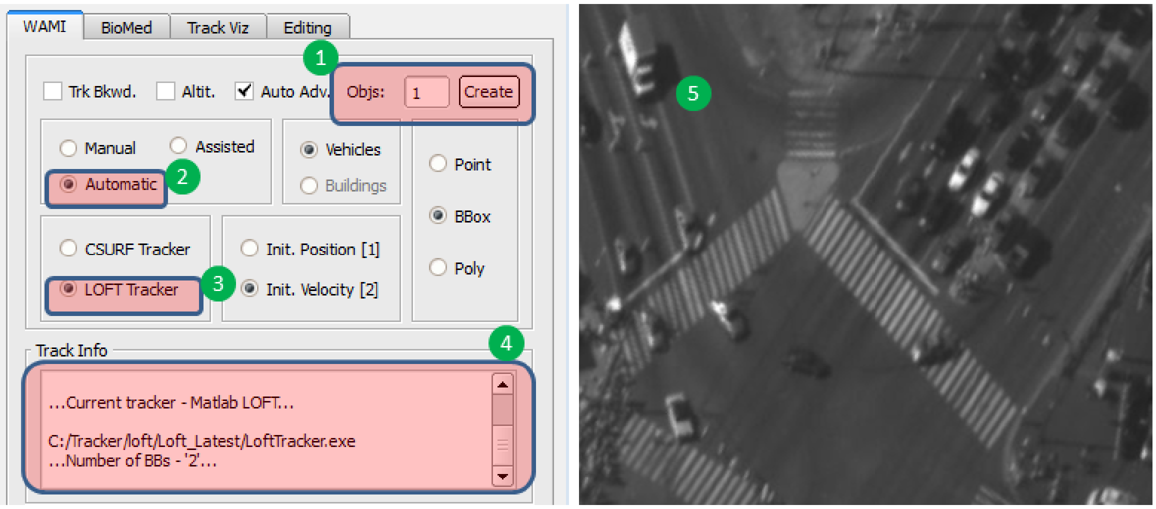

ID: 1 track length: 19 time:17:35:38 frame:63338 - adjusted

PLANE altitude: 1832 latitude:35.3421 longitude:-80.7027
VEHICLE dist: 2100 lat:35.3480 lon:-80.6869 elev:76 azim:201
distance traveled: 942 orientation: 111

Figure 3.7: **Step III** of tracking an object using the CSURF Tracker. Once initialized, the CSURF program begins tracking the object (**8**). While tracking, CSURF presents *additional useful information* to the user (**9**, **10**, **11**) as well as the means to terminate itself (**12**). The trajectories for the object - *unregistered* (**13**) and *registered* (**14**) - are drawn in KOLAM *as soon as they are generated* by the tracker.

Software systems such as Kitware Inc's vpView and MIT Layer Annotator feature ground-truth generation by manual tracking, and feature a number of optimizations which attempt to increase user productivity, such as a simple, five to ten-frame tracker incorporated into the former, allowing users to perform actual object marking annotations on the data every five to ten frames; and rich annotation generation facilities per object and replication of such annotation properties to successive objects being tracked, in the case of the latter.

In KOLAM 3.0's manual tracking mode, the user tracks objects by hand marking the location of the target in each frame. These points can be visualized as a connected trajectory (vector plot in the overlay plane) as in the automated tracking mode. KOLAM currently supports marking of objects using points, bounding boxes or polygons.



□ Trk Bkwd. □ Altit. ☑ Auto Adv. Objs: 1 Create

○ Manual ○ Assisted ● Automatic

● Vehicles ○ Buildings

○ Point ● BBox ○ Poly

○ CSURF Tracker ● LOFT Tracker

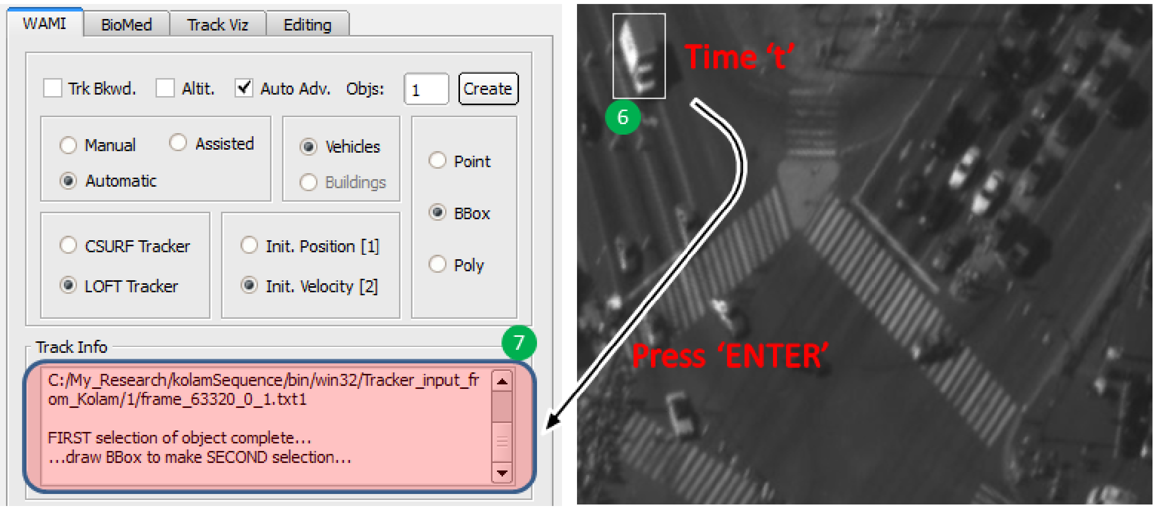
○ Init. Position [1] ● Init. Velocity [2]

Track Info

```
...Current tracker - Matlab LOFT...

C:/Tracker/loft/Loft_Latest/LoftTracker.exe
...Number of BBs - '2'...
```

Figure 3.8: **Step I** of tracking an object using the LOFT Tracker. The user first *creates a new Object-ID* (Highlighted by rounded rectangle **1**). Creating a new Object-ID in KOLAM always *sets the Tracking Mode to Automatic* (**2**) and sets CSURF as the selected tracker. The user *needs to change the tracker to LOFT* (**3**). KOLAM *gives the user feedback and the next step to perform* in the tracking process (**4**). The object (vehicle in this case) to be tracked is also highlighted (**5**).



□ Trk Bkwd. □ Altit. ☑ Auto Adv. Objs: 1 Create

○ Manual ○ Assisted ● Automatic

● Vehicles ○ Buildings

○ Point ● BBox ○ Poly

○ CSURF Tracker ● LOFT Tracker

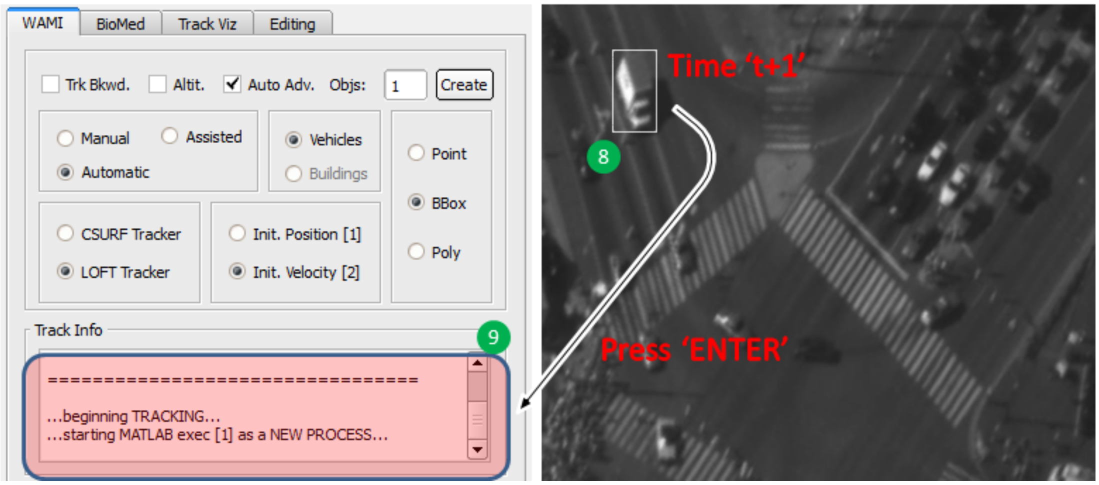
○ Init. Position [1] ● Init. Velocity [2]

Track Info

```
C:/My_Research/kolamSequence/bin/win32/Tracker_input_fr
om_Kolam/1/frame_63320_0_1.txt1

FIRST selection of object complete...
...draw BBox to make SECOND selection...
```

Figure 3.9: **Step II** of tracking an object using the LOFT Tracker. The user now draws a bounding box around the object to be tracked with the *right mouse button* (**6**). The drawn box must include the object and a *small* amount of background (again, see (**6**)). This box *may be re-drawn as many times as the user needs* to get it right. When satisfied with the box drawn, the user presses the 'ENTER' key. Since LOFT takes **TWO user-drawn BBoxes as input**, pressing 'ENTER' *advances the sequence by 1 time step* and lets the user draw the next BBox (**7**).



☐ Trk Bkwd. ☐ Altit. ☑ Auto Adv. Objs: 1 Create

○ Manual ○ Assisted
● Automatic

● Vehicles
○ Buildings

○ Point
● BBox
○ Poly

○ CSURF Tracker
● LOFT Tracker

○ Init. Position [1]
● Init. Velocity [2]

Track Info

```
================================

...beginning TRACKING...
...starting MATLAB exec [1] as a NEW PROCESS...
```

Figure 3.10: **Step III** of tracking an object using the LOFT Tracker. The user now draws a *SECOND* bounding box around the object to be tracked, for the *SECOND time step* (**8**). When satisfied, the user presses the 'ENTER' key. This action *ends the object selection phase* for LOFT; *creates a .txt1 parameter file* containing information about the drawn boxes, dataset path, output file path etc. *and invokes the tracker program* as an external process, with the .txt1 file as a parameter. Feedback regarding the same is presented by KOLAM to the user (**9**).

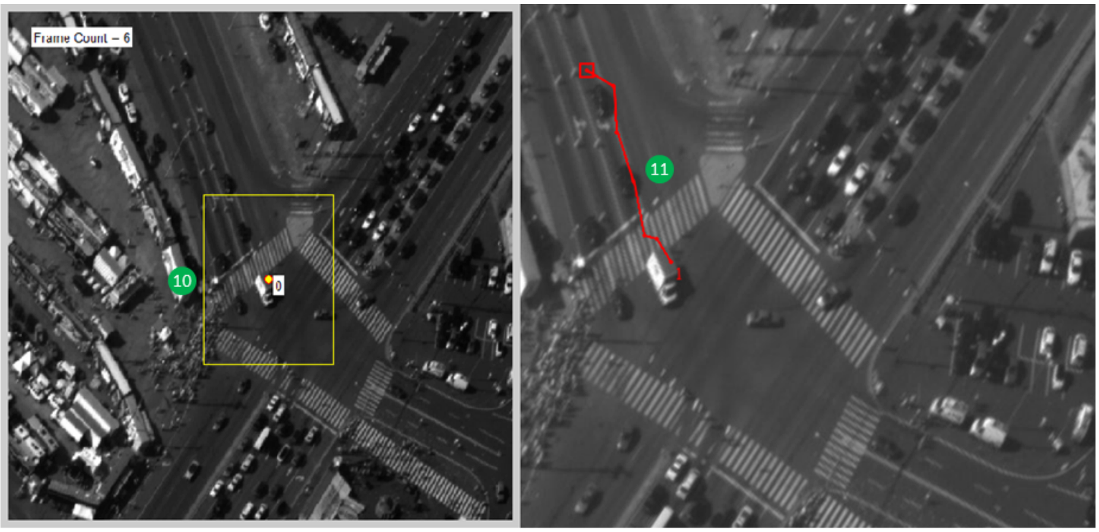


Figure 3.11: **Step IV** of tracking an object using the LOFT Tracker. Once initialized, the LOFT program begins tracking the object (**10**). While tracking, LOFT presents *additional useful information* to the user. The trajectory for the object (**11**) - is drawn in KOLAM *as soon as it is generated* by the tracker.

The fastest and easiest means of manual tracking involves using the *auto-advance* option to automatically step to the next frame as soon as the user marks the location of the object. In this mode of operation, an expert user can quickly generate long ground-truth trajectories. The auto-advance feature in combination with frame advance aims to provide flexibility in quickly jumping to the frames where the target is moving (more rapidly) and skip frames when the target is stationary for example; the skipped frames are connected via a constant velocity trajectory. To update an incorrect target location on the current frame, *auto-advance* should be disabled. KOLAM 3.0 also provides select features for enhancing efficiency of manual tracking: draw primitives such as bounding boxes or polygons may be spatially replicated, orientations of annotations may be altered for a group of objects, and so on. **Algorithm** 8 describes how an object of interest may be visually tracked by hand by the user; the different possible user operations are covered by the algorithm 'Perform_Track_Editing' (**Algorithm** 14) and the algorithms invoked therein.

Ideas for future work in this regard include the following: small lengths of trajectories may be linearly interpolated under special conditions in the data (for example, a vehicle moving in a near-straight line along a straight road for several frames, at constant velocity); and properties of moving objects in past frames such as direction of motion and velocity may also be used to aid in annotating once every few frames instead of for every single frame.

3.4.4 Assisted Tracking

Also known as semi-automated/supervised target tracking, human-in-the-loop tracking, and tracking with expert intervention; it is the ultimate goal for tracking purposes. First, it moves away from pure manual object annotation from frame to frame of the image se-

Algorithm 6: Perform_Automatic_Tracking

```
Input  : Image sequence $I$, task to track object(s) purely via automated methods
         based on pre-defined spatio-temporal parameters, tracker program(s)
Output: Tracker-generated trajectories; loaded in memory (data structure) and
         visualized by KOLAM
TrkType = $\{CSURF, LOFT, ...\}$;
TrajParam = $\{Visibility, Color, Thickness\}$;
TrajInitTypes = $\{Point, BBox\}$;
currTrajIdx, NumTrkInits = 0, TrajInitType = 0;
Process P, list¡Process¿ ProcessList;
Activate AutoOverlay instance for Automatic Tracking;
Prepare ProcessList for tracker invocation instances;
Prepare TrajectoryDS trajectory data structure;
while AutoOverlay is active do
    for Every ID the User Creates or Re-uses do
        NumTrkInits = TrkType [0].inits;
        if User Creates New ID then
            TrajectoryDS.append (new Trajectory);
            currTrajIdx = TrajectoryDS.$size - 1$;
        end
        // Choose Object Selection Primitive type
        for $i \leftarrow 0$ to TrajInitTypes.$size - 1$ do
            if User selects TrajInitTypes $[i]$ then
                TrajInitType = TrajInitTypes $[i]$;
            end
        end
        // Choose Tracker type
        for $i \leftarrow 0$ to TrkType.$size - 1$ do
            if User selects TrkType $[i]$ then
                NumTrkInits = TrkType [i].inits;
            end
        end
        // FOR Loop continues in Listing 7
    end
end
Return to KOLAM's main UI Loop;
```

Algorithm 7: Perform_Automatic_Tracking: FOR Loop continued

```
// Create Tracker Init(s) by highlighting the object
    using the selected primitive
for $i \leftarrow 0$ to NumTrkInits.$size - 1$ do
    User draws Initialization $i$ using TrajInitType;
    if User wants to change Initialization $i$ then
        $i - -$;
    end
    Add Initialization $i$ to TrkInitList;
end
// Create and Initiate Tracker Process
Init P {TrkType, TrkInitList, $imgPath, resPath$};
ProcessList.append(P);
startASync(P);
while P.isRunning() do
    TrajectoryDS [currTrajIdx ].append(P.$output()$);
    for $i \leftarrow 0$ to TrajParam.$size - 1$ do
        if User alters TrajParam $[i]$ then
            TrajectoryDS [currTrajIdx ].TrajParam $[i] \leftarrow$ TrajParam $[i]$;
        end
    end
    Update Trajectory display;
end
deleteProcess(P);
```

Algorithm 8: Perform_Visual_Tracking

```
Input  : Image sequence $I$, task to track object(s) purely via user intervention,
         based on pre-defined spatio-temporal parameters
Output: User-generated trajectories; loaded in memory (data structure) and
         visualized by KOLAM
TrajParam $= \{Visibility, Color, Thickness\}$;
TrajInitTypes $= \{Point, BBox, Polygon\}$;
currTrajIdx, TrajInitType = 0, AutoAdvance = true;
Activate VizOverlay instance for Visual Tracking;
Prepare TrajectoryDS trajectory data structure;
while VizOverlay is active do
    for Every ID the User Creates or Re-uses do
        if User Creates New ID then
            TrajectoryDS.append (new Trajectory);
            currTrajIdx = TrajectoryDS.$size - 1$;
        end
        // Choose Object Selection Primitive type
        for $i \leftarrow 0$ to TrajInitTypes.$size - 1$ do
            if User selects TrajInitTypes $[i]$ then
                TrajInitType = TrajInitTypes $[i]$;
            end
        end
    end
end
Return to KOLAM's main UI Loop;
```

Algorithm 9: Perform_Visual_Tracking: FOR Loop continued

```
// Add Points to Trajectory
if User Adds Time Steps to a Trajectory then
    if TrajInitType $== 1$ then
        TrajectoryDS [currTrajIdx ].append(Point);
    end
    else
        while !Primitive.$Completed()$ do
            Primitive.append(Point);
            if Primitive.$Completed()$ then
                break;
            end
        end
        TrajectoryDS [currTrajIdx ].append(Primitive);
    end
end
// Modify Existing Trajectory
if User Modifies an Existing Trajectory then
    Call Perform_Track_Editing() for this ID;
end
// Update Trajectory Parameters and Display
for $i \leftarrow 0$ to TrajParam.$size - 1$ do
    if User alters TrajParam $[i]$ then
        TrajectoryDS [currTrajIdx ].TrajParam $[i] \leftarrow$ TrajParam $[i]$;
    end
end
Update Trajectory display;
```

quence, thus offering significant time savings over the same while being less error prone. Second, given that current automatic tracking algorithms are not trustworthy enough in dense urban environments with many movers, assisted tracking offers a way to handle such scenarios by utilizing automatic tracking programs in combination with manual intervention in the form of trajectory editing. The assisted tracking mode is especially useful when tracking involves many similar targets maneuvering through multiple occlusions and shadows in complex environments.

In the assisted tracking mode, the user is able to stop the automatic tracker as necessary, manually correct trajectory errors, or add target location information in a difficult to track region, then quickly switch back to the auto-tracker mode in a seamless fashion. The sequence of operations in assisted tracking includes use of the automatic tracking process, combined with manual tracking and trajectory editing procedures which are performed by the user in an iterative fashion until the supervised tracking task is completed. In supervised tracking and video indexing, the user learns to anticipate the most common failure modes of the automatic tracker, thereby significantly increasing the productivity of the analyst. Preventative intervention by the user aims at avoiding long invalid track segments from being generated by the automatic tracker running unsupervised that then need to be manually inspected and corrected. For simultaneous tracking of multiple objects, the user needs to time-slice between the objects in order to enhance productivity.

KOLAM 3.0 is capable of performing assisted tracking, albeit in a non-seamless manner. This is primarily due to the fact that multiple user interaction steps are currently needed at the interfaces of the different operations. Firstly: the identification of non-accurate behavior of the automatic tracker and transitioning the created trajectory into the manual or editing modes of operation is not seamless. Secondly: the various editing operations in their current state are not seamless enough for efficient assisted tracking; a few steps like the repetitive need to access the GUI to perform editing operation switching still need to be streamlined or eliminated. Finally: the hand-off of the edited trajectory to the automatic tracker for re-initialization and continuation of tracking is currently a series of separate steps. **Algorithm 10** describes how analysts may use KOLAM to perform a tightly-coupled combination of tracking automation (automatic tracking) and human-in-the-loop intervention (visual tracking) in order to complete object tracking tasks that are either too erroneous or too time inefficient to be completed by solely one or the other means of object tracking.

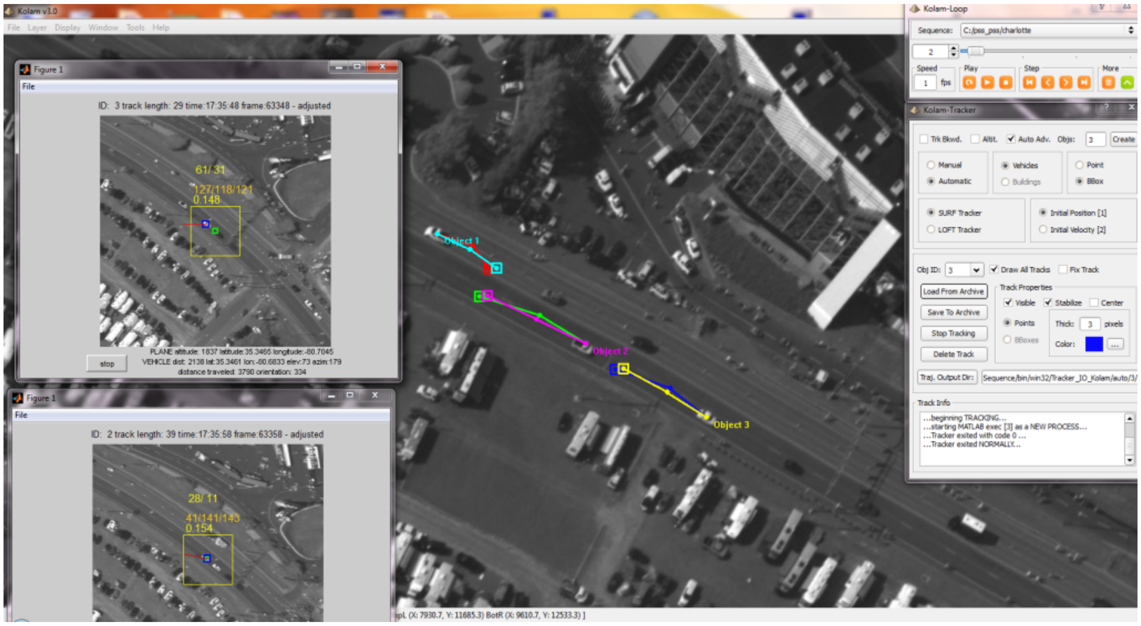
Implementation of a seamless and robust form of assisted tracking is part of the author's future work. To this end, a number of points being followed up on are listed below:

- Goals toward achieving seamless assisted tracking can be broadly given as:
  - Faster ground truth generation.
  - Higher productivity via one analyst monitoring multiple targets
- The need to match the speed of the tracking algorithm versus the ability of the human operator to verify the result. This needs to be done:
  - Frame by Frame
  - Chunk by Chunk (a 'chunk' referring to a group of frames)
  - A combination of the above two
  - Event by Event. Examples of events are failure to track, and lost target.
- Seamless transfer of data and control between the tracker and editing interfaces - by reducing user input and decision steps, caching one or more of the most recent edited points in anticipation of a tracker re-initialization etc.

Algorithm 10: Human_In_The_Loop_Tracking

```
Input  : Image sequence $I$, task to track object(s) purely via *tightly coupled
         combination* of **automated methods** and **user intervention** based on
         pre-defined spatio-temporal parameters, tracker program(s)
Output: *Tracker-and-User*-generated **trajectories**; loaded in memory (data
         structure) and visualized by KOLAM

TrkType $= \{CSURF, LOFT, ...\}$;
TrajParam $= \{Visibility, Color, Thickness\}$;
TrajInitTypes $= \{Point, BBox, Polygon\}$;
currTrajIdx, TrajInitType = $0$, AutoAdvance = true;
Activate AssistOverlay instance for Human in the Loop Tracking;
Prepare list<Process> ProcessList for tracker invocation instances;
Prepare TrajectoryDS trajectory data structure;
while AssistOverlay is active do
    for Every ID the User Creates or Re-uses do
        while Tracking for this ID is incomplete do
            Perform_Automatic_Tracking();
            if Tracking is complete for this ID then
                break;
            end
            currTraj $\leftarrow$ TrajectoryDS[currTrajIdx];
            if currTraj has wrong points then
                while currTraj.$currPt.isWrong()$ do
                    currTimeStep = currTimeStep$-1$;
                end
            end
            while Automatic Tracking cannot be done do
                Invoke the desired editing operation of Perform_Trajectory_Editing(),
                  invoked from Perform_Visual_Tracking(), on currTraj.currPt;
                currTimeStep = currTimeStep$+1$;
            end
        end
    end
end
Return to KOLAM's main UI Loop;
```



ID: 3 track length: 29 time:17:35:48 frame:63348 - adjusted

PLANE altitude: 1837 latitude:35.3465 longitude:-80.7045
VEHICLE dist: 2138 lat:35.3461 lon:-80.6833 elev:73 azim:179
distance traveled: 3790 orientation: 334

ID: 2 track length: 39 time:17:35:58 frame:63358 - adjusted

Sequence: C:/pss_pss/charlotte

2

Speed: 1 fps

□ Trk Bkwd. □ Altit. ☑ Auto Adv. Objs: 3 Create

○ Manual
● Automatic

● Vehicles
○ Buildings

○ Point
● BBox

● SURF Tracker
○ LOFT Tracker

● Initial Position [1]
○ Initial Velocity [2]

Obj ID: 3 ☑ Draw All Tracks □ Fix Track

☑ Visible ☑ Stabilize □ Center

● Points
○ BBoxes

Thick: 3 pixels

Color: 🟦 ...

Traj. Output Dir: Sequence/bin/win32/Tracker_IO_Kolam/auto/3/

```
...beginning TRACKING...
...starting MATLAB exec [3] as a NEW PROCESS...
...Tracker exited with code 0 ...
...Tracker exited NORMALLY...
```

pL (X: 7930.7, Y: 11685.3) BotR (X: 9610.7, Y: 12533.3)]

Figure 3.12: Simultaneous visualization of tracker algorithm execution and trajectory visualization in KOLAM. There are three instances of the tracking algorithm running to track three objects. The two inlaid (MATLAB) windows shown on the left are the realtime (heads-up or cursor-on-target) displays generated by the automatic tracking algorithms. The main window shows both stabilized and unstabilized computed trajectories for the three objects.

- A way to minimize data navigation - fast forward or rewind of the image sequence as this is highly distracting to the analyst.

- Provision of multiple view windows - preferably a single main display window accompanied by multiple display sub-windows; so as to keep the global view context while simultaneously monitoring multiple objects, each in visual isolation.

3.5 Conclusions

The utility of KOLAM for exploratory visualization and analysis of both static and time-varying imagery of different types and formats conveys the usefulness of the system for a variety of applications requiring accurate multiple target tracking that can scale to a large number of objects and image sizes. This was the primary driving force for the definition of the problem statement and subsequently identifying the associated challenges by the author. In devising the solutions to these, the author conceived, designed and implemented all the object tracking related functionality in KOLAM 3.0.

The loose coupling between KOLAM and the external automatic tracking algorithm processes enables immediate display of the computed trajectory up to that point in time as shown in Figure 3.12. For WAMI imagery KOLAM's ability to interactively animate gigapixel-sized images as well as perform automatic or manual tracking and trajectory visualization and analysis translates into applications in both the civilian and defense sectors. On the civilian side, the tracking, visualization, and annotation capabilities may be utilized by law enforcement for forensic analysis, for urban planning and traffic patterns, video summarization for long duration surveillance and emergency response [10]. In the defense context KOLAM can be used for improved situational awareness, persistent observation of targets, reconnaissance, force protection, rapid targeting and response.

Chapter 4

CASE STUDY 2: Trajectory Visualization and Editing

4.1 Introduction

The detection and tracking of objects of interest (both moving and stationary) forms an essential component of the exploitation of WAMI data. The same is true to some extent for a number of cell biology datasets as well, in particular those wherein analyzing cell motility characteristics over time is a subject of research interest. The problems and challenges, as well as the solutions implemented as KOLAM's capabilities regarding handling the detection of such moving objects were discussed in Chapter 3. The visualization and editing of object trajectories play a vital role in the generation of ground truth trajectory generation for datasets of interest, and are essential components of a human in the loop (assisted) object tracking system. This chapter describes the details the author's work in conceiving and implementing KOLAM's trajectory visualization system as well as KOLAM's trajectory editing system.

4.2 Trajectory Visualization in KOLAM

The tracking subsystem in KOLAM 3.0 has been specifically designed to accommodate the visual analytic needs of tracking objects of interest in both WAMI datasets as well as more general video sequences. The KOLAM-Tracker tool accomplishes this by providing interfaces for manual ground-truth (target) annotation, simultaneous tracking of multiple objects, editing of trajectories and assisted tracking. Visualization of the trajectories generated, modified and saved in these different ways is handled by KOLAM 3.0's trajectory visualization module, the aspects of which are now discussed in detail.

4.2.1 Current Trajectory Visualization Features

The trajectory visualization subsystem of KOLAM 3.0 was conceived with an initial set of requirements, and has evolved over time to serve new needs as well as modifications to the original requirement set. Details about the subsystem such as design and implementation choices, and the justifications for the same are best conveyed against the backdrop of the current feature set that the trajectory visualization system supports, and the features that are required to address emerging needs. As such, said features are outlined here.

- Local Stabilization: KOLAM 3.0 can visualize both unstabilized trajectories (those generated by trackers operating on unregistered imagery) and locally stabilized versions of these trajectories, either separately or together, for each tracked object. The visual toggling of these trajectory types is done via Kolam-Tracker. Stabilized trajectories are internally handled by KOLAM 3.0 in two ways:

  - As a complete second trajectory independent of the unstabilized trajectory, for each tracked object, and
  - As a set of offset values, one corresponding to each point on the unstabilized trajectory. KOLAM 3.0 constructs and displays the stabilized trajectory by appropriately accumulating these offsets over the length of the unstabilized trajectory.

- Appearance: As briefly described under the functionality of Kolam-Tracker in Section 2.5.3, KOLAM 3.0 allows users to alter the visual representation of trajectories, either individually or in groupings, by permitting properties such as thickness, color and visibility to be modified. While the default values of color and thickness are determined independent of any properties of the tracked objects, such properties can be taken into account in the visualization. An example is the utilization of properties such as velocity, acceleration and geographical location to determine object trajectory color and thickness.

  Another modifiable visual property is the draw type of the currently used primitive. A good example of this is provided by KOLAM 3.0's trajectory editing facility, wherein trajectories selected for editing are highlighted by enclosing them within tight boundaries. The modified draw type feature is also available for polygon drawing within KOLAM 3.0. Besides modifying the primitive draw type, KOLAM 3.0 also supports drawing of any additional image markers, at user-determined offsets from select or all points along trajectories. This is currently limited to a single text label for each point on a trajectory, but may be easily extended.

- Intelligent navigation: This feature of KOLAM 3.0's trajectory drawing was incorporated so that users could determine spatial proximity of objects being tracked to certain features of interest in the imagery. For example, was a vehicle traveling along a particular path in proximity to a particular building or parked vehicle at any point in time. Effectively, a single user event (such as a mouse click) at a point of interest on the imagery will rewind or fast-forward the image sequence such that any one of all currently loaded object trajectories is within a pre-determined proximity of the user event point. While apparent, it is reiterated that the sequence rewind or fast forward causes the state of all other object trajectories to also be updated. Thus, the positions of objects that are not of immediate interest to the user are also made available, and are potentially useful for examining the relationship between object positions, if any. If none of the tracked objects was or will ever be within proximity of the user selected point of interest, KOLAM doesn't respond; thereby indicating to the user that no object association with the point of interest can be found. The process is a spatial search problem and is made efficient by organizing the bounding box extents of the trajectories in an interval tree, which allows for O(log n) search time given n trajectories.

4.2.2 Metadata Encoding for Trajectories

Imagery produced within different research domains have different significance for researchers. For example, the majority of quantifiable properties of tracked objects that a biomedical researcher is interested in (cell age, cell division status, cell malignancy etc.) are completely dissimilar to those properties that an analyst of WAMI imagery needs to follow (vehicle type, traffic patterns, specific vehicle movement patterns etc.). Some object properties, such as object velocity, acceleration and direction of motion, are of interest to users from multiple domains (they are relevant to WAMI and Biomedical users, in particular) in similar or dissimilar ways.

Such metadata that are associated with different properties of objects are primarily used for different types of statistical analyses. The difference in available visualizations and types of visualizations depends primarily on whether users expect the visualization software to incorporate such statistical analysis capabilities into the work-flow. In the case of KOLAM 3.0, its current users perform the prototyping and analysis of their research implementations within environments such as MATLAB, which provide a multitude of mature and stable statistical analysis and result display features, often accessible via a single function call. So while making such facilities within KOLAM 3.0 is attractive, their absence does not hamper the productivity of such users. Nevertheless, the author posits that making such features available within KOLAM 3.0 is still critical: environments such as MATLAB are incapable of the imagery display flexibility that KOLAM 3.0 possesses, and the ability to perform all required analyses in the global, visual backdrop of the image data being analyzed will potentially lead to new observations and insights for researchers.

4.2.3 Challenges in Trajectory Representation

KOLAM 3.0 needs to provide a scalable environment for handling the smooth, interactive, simultaneous display of tens of thousands (or even greater numbers) of trajectories. It is therefore incumbent that KOLAM 3.0 provide a memory representation or data structure that can serve as answer to these challenging requirements. Listed below are factors that further increase the complexity of implementation of an optimum data structure(s) for trajectory representation.

- As part of its current operational requirements, KOLAM 3.0 cannot make strict assumptions about trajectory data for the sake of optimizing the memory representation of trajectory database(s). For example, trajectories generated for unregistered imagery make caching of trajectory history difficult, since all position values for all objects change with each time step. Caching can still be done, but at the expense of increased time and space complexity. One means of doing so (and which is implemented and active) would be incorporating local registration modules into KOLAM to apply incremental positional corrections at each time step: 'base' position values will remain unchanged over time and could be cached, but the aforementioned module(s) will have to be invoked for each vertex of each trajectory, for each time step; in order to obtain the actual positions of objects.

- KOLAM 3.0 cannot make assumptions about the temporal navigation patterns that different users follow while observing the imagery. For example, some users might be content solely with unidirectional playback of the image sequence(s). Other users might play forward, backward, play back-and-forth between a small subset of images, or perform jumps to non-consecutive parts of the image sequences. This implies that potentially all trajectory data must be readily displayable at potentially all times: a worst case scenario which makes optimizing for temporal locality/coherence of data navigation difficult.

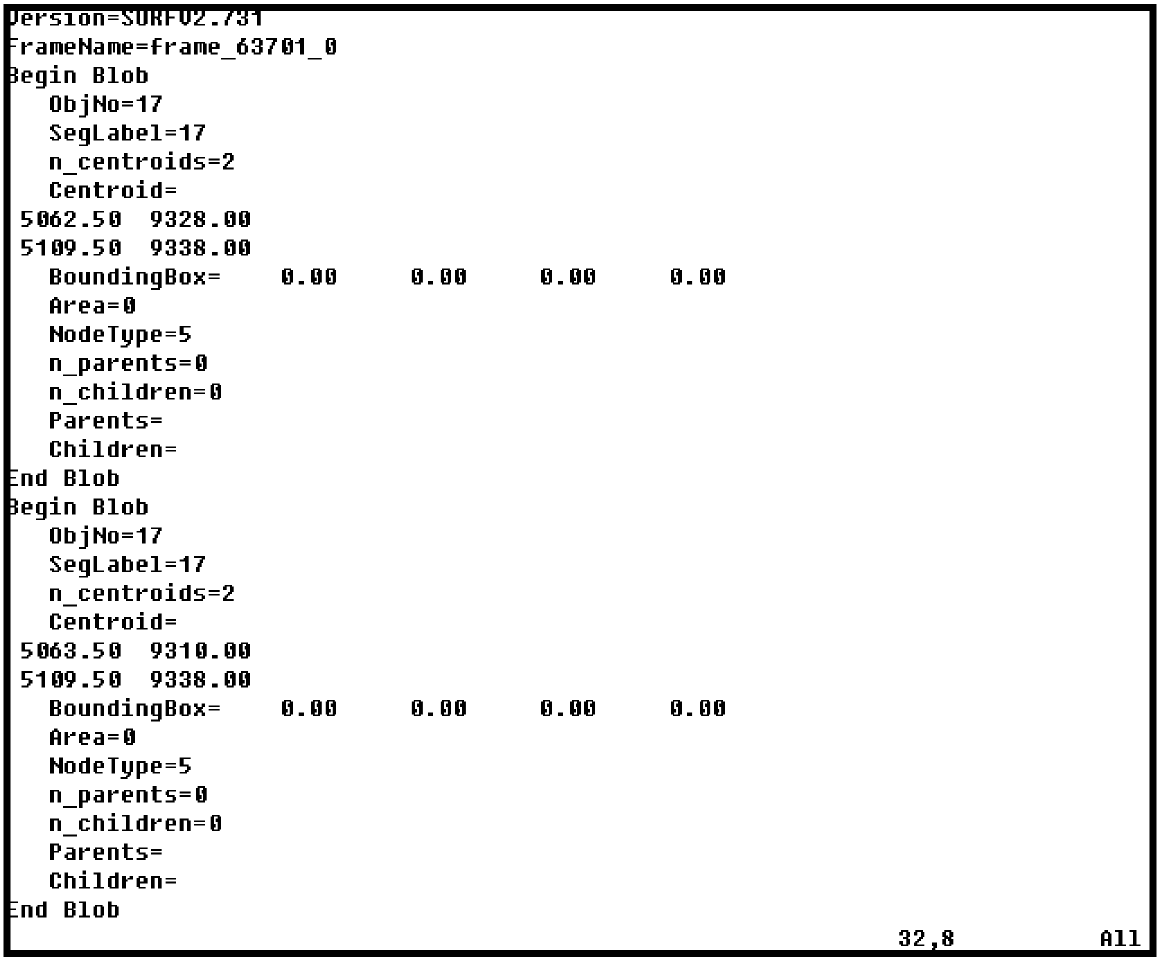
4.2.4 Trajectory Drawing in KOLAM 3.0

Given the different features taken into account for trajectories, the extra handling that has to be accounted for in terms of metadata and the visualization challenges, we now present the details of KOLAM 3.0's trajectory drawing system.

Drawing of trajectories is performed by utilizing a combination of Qt's QPainter drawing capabilities and OpenGL commands on a dedicated overlay plane that is enabled when trajectories are being visualized in KOLAM 3.0. OpenGL is leveraged for the matrix computations involved in all coordinate system transformations that are performed, and the QPainter object is utilized for its extensive, customizable and vector drawing of draw primitives, and image element and font drawing support.

Trajectory datasets are currently stored on disk in two formats: a flat, unstructured text file format and a semi-structured kw18 file format. The former was developed during the initial phase of trajectory drawing development in KOLAM 3.0 and is no longer generated by any tracker programs, though KOLAM 3.0 continues to support it to continue visualizing any old trajectory datasets. The kw18 file format developed by Kitware Inc. to represent trajectories is the file format currently used by KOLAM 3.0 and which is also generated by all trackers developed in-house. Regardless of file format, KOLAM 3.0 loads the trajectories read from file via appropriate readers, into a hierarchical data structure which represents all loaded trajectories, from one or more datasets, across the duration of the underlying image sequence. During image sequence animation, provided that trajectory drawing is enabled, KOLAM 3.0's trajectory drawing system navigates the data structure in sync with the animated frames, and draws the trajectories with a mix of QPainter and OpenGL calls. Samples of the text file format and the kw-18 file format are provided in Figure 4.1 and Figure 4.2.

Given now is the description for both the data structure for storing trajectory datasets in memory, as well as the algorithm for drawing the trajectories. The scenarios encountered in the various applications are enumerated, followed by a description of the design choices



```
Version=SURF02.731
FrameName=frame_63701_0
Begin Blob
   ObjNo=17
   SegLabel=17
   n_centroids=2
   Centroid=
 5062.50  9328.00
 5109.50  9338.00
   BoundingBox=    0.00     0.00     0.00     0.00
   Area=0
   NodeType=5
   n_parents=0
   n_children=0
   Parents=
   Children=
End Blob
Begin Blob
   ObjNo=17
   SegLabel=17
   n_centroids=2
   Centroid=
 5063.50  9310.00
 5109.50  9338.00
   BoundingBox=    0.00     0.00     0.00     0.00
   Area=0
   NodeType=5
   n_parents=0
   n_children=0
   Parents=
   Children=
End Blob
                                                                  32,8          All
```

Figure 4.1: Flat file trajectory storage format for KOLAM.

which permit a single implementation to be capable of handling all the disparate cases. See **Algorithm 11** and **Algorithm 12** for more detail. **Algorithm 13**, which is the continuation of **Algorithm 12**, details the process of trajectory stabilization.

Given the details of trajectory drawing, the following section describes trajectory editing in KOLAM 3.0.

Algorithm 11: Trajectory Drawing Algorithm

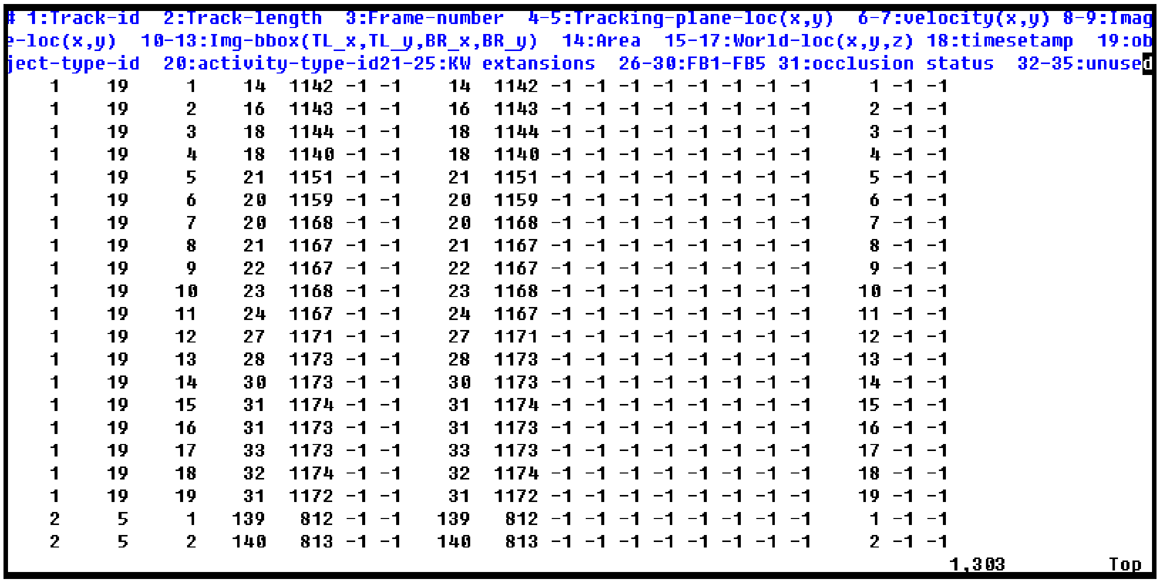
```
input  : Sequence(s) of images of dimensions $[w \times h]$ loaded in KOLAM; trajectory
         dataset(s) loaded in KOLAM, with the task of visualizing these dataset(s)
output: KOLAM correctly overlays and visualizes the trajectory dataset(s), both
         spatially and temporally as well as with all user modifications; on the image
         sequence
Initialize variables: painter, penWidth, dbType, dbPointer, currObjID,
  currFrNum, showIDs, showBBs, showCentroids, showLines, toggleName,
  toggleColGroup, toggleEdit, sendInfoToLoc, stabilValue, centering;
while Trajectory overlay is active do
    dbPointer ← getDbPointer();
    dbType ← dbPointer.getType();
    ... or any of the other variables ...
    Handle state changes from mousePress(), mouseRelease(), mouseMove() and
      keyPress();
    if dbType == $'FLAT\_FILES'$ then
        Init_Data_Structure_Setup(FLAT_FILES);
    else
        Init_Data_Structure_Setup(KW18_FILES);
    end
    if Trajectory overlay visibility is enabled then
        Save previous OpenGL drawing state;
        painter.init(display);
        painter.set(pen(penColor,penWidth), font(f));
        if toggleName then
            painter.saveState();
            Draw all trajectory dataset names with painter;
            painter.restoreState();
        end
        draw_Trajectories_Aux();
        Restore previous OpenGL drawing state;
    end
end
```

Algorithm 12: draw_Trajectories_Aux()

```
for Each Trajectory dataset 'h' do
    for Each Object ID 'i' do
        start ← ObjID[i].startIndex;
        end ← ObjID[i].endIndex;
        if curFrame >= start && curFrame <= end then
            if toggleColGroup then
                penColor ← color(Dataset 'h');
            else
                penColor ← color(ObjID 'i');
            end
            painter.set(pen(penColor, penWidth));
            for (j = currFrame-start; j >= 0; j−−) do
                painter.saveState();
                if stabilValue == 0 then
                    x ← ObjID[i].centroid.x;
                    y ← ObjID[i].centroid.y;
                    [scrX, scrY] ← imgToScreen(x, y);
                    painter.translate(scrX, scrY);
                    painter.scale(getZoom(), getZoom());
                    if $j == currFrame - start$ then
                        drawCentroid(currPt);
                        if showIDs then
                            painter.drawID();
                        end
                        if showBBs then
                            getBBs(ObjID[i]);
                            drawBBs(ObjID[i]);
                        end
                        Highlight trajectory end;
                    else
                        drawCentroid(currPt);
                    end
                end
            end
        end
    end
end
```

Algorithm 13: draw_Trajectories_Aux(); Inner ELSE block and remainder

```
// Continued from previous listing
// 'Else' corresponds to innermost 'If'
else if stabilValue $== 1$ then
    if $j == curFrame - start$ then
        accum.setX(0);
        accum.setY(0);
    end
    [X1, Y1] $\leftarrow$ ObjID[i].pt[curFrame];
    X2 $\leftarrow$ X1 $-$ accum.X;
    Y2 $\leftarrow$ Y1 $-$ accum.Y;
    [scrX, scrY] $\leftarrow$ imgToScreen(X2, Y2);
    accum.set(accum + ObjID[i].pt[j].diff);
    painter.translate(scrX, scrY);
    painter.scale(getZoom(), getZoom());
end
if $j >= 1$ && showLines then
    if toggleEdit then
        painter.drawBB(currPt, prevPt);
    end
    painter.drawLine(currPt, prevPt);
end
painter.restoreState();
```



```
# 1:Track-id  2:Track-length  3:Frame-number  4-5:Tracking-plane-loc(x,y)  6-7:velocity(x,y) 8-9:Imag
e-loc(x,y)  10-13:Img-bbox(TL_x,TL_y,BR_x,BR_y)  14:Area  15-17:World-loc(x,y,z) 18:timesetamp  19:ob
ject-type-id  20:activity-type-id21-25:KW extansions  26-30:FB1-FB5 31:occlusion status  32-35:unused
    1     19     1    14  1142 -1 -1    14  1142 -1 -1 -1 -1 -1 -1 -1 -1     1 -1 -1
    1     19     2    16  1143 -1 -1    16  1143 -1 -1 -1 -1 -1 -1 -1 -1     2 -1 -1
    1     19     3    18  1144 -1 -1    18  1144 -1 -1 -1 -1 -1 -1 -1 -1     3 -1 -1
    1     19     4    18  1140 -1 -1    18  1140 -1 -1 -1 -1 -1 -1 -1 -1     4 -1 -1
    1     19     5    21  1151 -1 -1    21  1151 -1 -1 -1 -1 -1 -1 -1 -1     5 -1 -1
    1     19     6    20  1159 -1 -1    20  1159 -1 -1 -1 -1 -1 -1 -1 -1     6 -1 -1
    1     19     7    20  1168 -1 -1    20  1168 -1 -1 -1 -1 -1 -1 -1 -1     7 -1 -1
    1     19     8    21  1167 -1 -1    21  1167 -1 -1 -1 -1 -1 -1 -1 -1     8 -1 -1
    1     19     9    22  1167 -1 -1    22  1167 -1 -1 -1 -1 -1 -1 -1 -1     9 -1 -1
    1     19    10    23  1168 -1 -1    23  1168 -1 -1 -1 -1 -1 -1 -1 -1    10 -1 -1
    1     19    11    24  1167 -1 -1    24  1167 -1 -1 -1 -1 -1 -1 -1 -1    11 -1 -1
    1     19    12    27  1171 -1 -1    27  1171 -1 -1 -1 -1 -1 -1 -1 -1    12 -1 -1
    1     19    13    28  1173 -1 -1    28  1173 -1 -1 -1 -1 -1 -1 -1 -1    13 -1 -1
    1     19    14    30  1173 -1 -1    30  1173 -1 -1 -1 -1 -1 -1 -1 -1    14 -1 -1
    1     19    15    31  1174 -1 -1    31  1174 -1 -1 -1 -1 -1 -1 -1 -1    15 -1 -1
    1     19    16    31  1173 -1 -1    31  1173 -1 -1 -1 -1 -1 -1 -1 -1    16 -1 -1
    1     19    17    33  1173 -1 -1    33  1173 -1 -1 -1 -1 -1 -1 -1 -1    17 -1 -1
    1     19    18    32  1174 -1 -1    32  1174 -1 -1 -1 -1 -1 -1 -1 -1    18 -1 -1
    1     19    19    31  1172 -1 -1    31  1172 -1 -1 -1 -1 -1 -1 -1 -1    19 -1 -1
    2      5     1   139   812 -1 -1   139   812 -1 -1 -1 -1 -1 -1 -1 -1     1 -1 -1
    2      5     2   140   813 -1 -1   140   813 -1 -1 -1 -1 -1 -1 -1 -1     2 -1 -1
                                                                                  1,303          Top
```

Figure 4.2: kw-18 file trajectory storage format for KOLAM.

4.3 Interactive Trajectory Editing

Tracking algorithms such as CSURF and LOFT are capable of predicting the movement and future positions of moving objects with high accuracy for a reasonable period of time. However, factors such as object illumination and shape change, parallax distortion, image registration errors and other erroneous elements in the imagery have made the goal of perfect automated tracking impossible to achieve, to date. Assisted tracking, as discussed in Section 3.4.4 is an accepted approach of addressing this deficiency, as follows: (a) Supplement the tracking algorithm with a robust manual trajectory generation and editing module, (b) Implement efficient interfaces for the inter-module interactions, and (c) Finally, the system UI must also be optimized to maximize user productivity. This in turn successfully positions KOLAM as a viable solution for large scale object tracking endeavors in multiple research domains.

In order to be editable, the trajectory of interest must first be **selected** by the user.

Trajectory selection is a one-click operation that greatly reduces the closest point search space for the user clicks performed during each editing operation. The overall UI workflow for trajectory editing is given in **Algorithm 14**.

4.3.1 Connecting Automatic Tracking and User Intervention

As discussed in Section 3.4.4, the typical scenario tackled by an assisted tracking system involves alternating automated tracker execution and manual trajectory editing steps. We assume that the algorithm implementations are as efficient as possible, in order to eliminate them as a limiting factor. Subsequently, the overall efficiency (the combination of user productivity and time spent) of the assisted tracking system is dependent on:

- Efficiency of the *Tracker-to-Editing* data interface,
- Aggregate efficiency of the various operations of the *Editing system*, and
- Efficiency of the *Editing-to-Tracker* data interface.

The *Tracker-to-Editing* interface handles the transfer of detected trajectory points of the tracked object to the visualization and user interaction module. The underlying limitation of this operation is the ceiling imposed on trajectory points generated per unit time by the tracker's detection speed. Therefore, the interface design must make this periodic delay as transparent as possible to the human operator while minimizing its own operational delays. A file I/O interface is a straightforward solution, simple to implement and offers the benefit of maximal decoupling of the two modules by minimizing inter-module interaction. However, incorporation of such a system into KOLAM 3.0 has demonstrated that the effect of file I/O delays scales proportionally with the number of objects being simultaneously

Algorithm 14: Perform_Trajectory_Editing

```
Input  : Image sequence $I$, task to edit object position(s) for one (or more)
         trajectories purely via user intervention based on visually perceived
         spatio-temporal parameters
Output: One or more points on One or more trajectories modified by the user;
         action and result interactively visualized by KOLAM
TrajOp $= \{Add, Delete, Move, Join, Split, Select\}$;
Prepare SelectTrajs list¡Trajectory ¿;
Prepare TrajectoryDS data structure;
Prepare minHeap CandidateTraj;
Activate EditOverlay instance for Human in the Loop Tracking;
while EditOverlay is active do
    for Every User Right-Click do
        UserPt $\leftarrow$ ScrToImg(User-Click);
        // Find trajectory point closest to user-click
        for $i \leftarrow 0$ to TrajectoryDS$.size - 1$ do
            currPt $\leftarrow$ TrajectoryDS $[i]$.currTimeStep.trajPt;
            currDist $\leftarrow$ EuclidDist(UserPt,currPt);
            if currDist $< \epsilon$ then
                CandidateTraj.addTo($i$,currDist);
            end
        end
    end
    SelectIdx $\leftarrow$ CandidateTraj.minimumDist();
    if !TrajectoryDS [SelectIdx].isSelected() then
        SelectTrajs.append(TrajectoryDS [SelectIdx]);
        Refresh display showing newly selected trajectory;
    end
    // Perform User-selected Editing operation
    (If TrajOp $==$ 1) Trajectory POINT_ADD();
    (eIf TrajOp $==$ 2) Trajectory POINT_DELETE();
    (eIf TrajOp $==$ 3) Trajectory POINT_MOVE();
    (eIf TrajOp $==$ 4) Trajectory SPLIT();
    (eIf TrajOp $==$ 5) Trajectory JOIN();
    (eIf TrajOp $==$ 6) Trajectory DESELECT();
end
Return to KOLAM's main UI Loop;
```

tracked. The viability of this approach is thus restricted to a small number of simultaneously tracked objects. Another approach involves tighter coupling of the modules and data transfer via auxiliary libraries. Point generation delays may be ameliorated by transferring them in groups.

The *Editing-to-Tracker* interface performs a relatively simple task: communication of coordinate information to the tracker for restarting its execution. Since this interface is activated by the human operator, even multiple tracker invocations (performed by one or more operators) are multiple isolated events involving single data transfers. It is important that the transfer of control from user to tracker be seamless. The library communication strategy outlined for the Tracker-to-Editing interface above can handle all requirements of this interface as well.

Unlike the two interfaces described above, the *Editing system* is solely driven by user interaction. The efficiency of this system thus depends on minimizing user interaction steps and maximizing the throughput of each step. Besides efficiency, all editing operations must be intuitive and reliably interactive from the user perspective. The interactive editing module in KOLAM 3.0's trajectory editing system fulfills these requirements by minimizing visual distraction during user interaction with the visualization, and minimizing the number of widget-to-visualization navigation steps by the user.

4.3.2 Factors Governing Trajectory Editing in KOLAM 3.0

The number of trajectories for a given data set is limited only by the number of distinct, trackable objects or features in the data set - implying potentially thousands, tens of thousands or an even greater number of trajectories. A simple method for maintaining the efficiency and interactivity of editing operations given this fact is to reduce the total number of trajectories that need to be examined to match the user editing events to, via some filtering mechanism. KOLAM 3.0 currently implements this filtering via user-driven trajectory selection. Thus, of all trajectories currently being displayed, the editing operations are active only for the user-selected subset. The status of a given trajectory as being selected is visually communicated to the user by enclosing said trajectory in a tight outline/boundary. While a simplistic scheme, the following factors make it a viable one:

- Trajectory editing is a *focus-intensive* task: Typical object tracking scenarios given time-sequenced imagery (which is either large in spatial dimensions, lengthy in the temporal dimension, or both) require that the analyst track as many objects as possible with complete accuracy in a given time frame. For the average user, the highest accuracy is typically guaranteed by focusing on these objects sequentially. Such per-trajectory focus is also highly desirable because incorrectly marking trajectories due to attention lapses will result in said trajectories having to be either partially or completely re-annotated. Thus, the average user is highly likely to select only a small number of related (part of a single longer trajectory, spatio-temporal proximity etc.) trajectories at any given time.

- The *Temporal aspect* of the data: Given that the visualization of time-sequenced imagery is governed by the linear flow of time, the trajectories corresponding to different tracked objects start being displayed at certain points in time, continue to be visible and may be interacted with for different durations, and are no longer displayed beyond their ending points. Accounting for this as well as for the perceptual and processing limitations of the human visual system, the average user will only select as many trajectories as he or she can completely and coherently focus attention on. This implies that the number of selected trajectories will always be small in comparison to the total number of trajectories being visualized.

4.3.3 Trajectory Editing Operations

Given these factors, the user typically selects a small set of trajectories that are highly likely to be related. As mentioned above, such relationships may be based on membership in a set (for example, individual trajectories forming part of a single longer trajectory), spatial and/or temporal proximity, the nature of the data and the objects being tracked therein, and possibly other factors. KOLAM 3.0 provides a small number of distinct editing operations which may either be used individually or in multiple combinations. These operations completely satisfy all user editing needs for trajectories, and are the following:

- ADD point(s) to a trajectory,
- DELETE point(s) from a trajectory,
- MOVE existing trajectory point(s), and
- JOIN trajectories.

Employing these operations to rapidly edit multiple trajectories requires that the user be presented with all relevant information. A basic component of such information is the complete history of the selected trajectories of interest, regardless of the current animation progress of the input imagery. KOLAM 3.0 allows for this alternative visualization of trajectories, thereby allowing the user to switch between time-dependent and time-independent trajectory views as desired. The editing operations are further described below.

ADD point(s) to a trajectory

The ADD operation is used to create track segments in portions of the WAMI sequence where no previous track data exists (*i.e.* extending a trajectory at the ends). It enables inserting primitive(s) before the beginning of a track, after the end of a track, or creating a track segment in-between two (or more) existing tracks in order to create a single, longer track.To add a single point, the user navigates to either one frame before the start of, or one frame after the end of the trajectory and drops the new point. To add multiple points, the user drops a new point several frames before the trajectory start or after the trajectory end. This results in trajectory interpolation - new points are added by KOLAM for the missing frames. The new points generated are currently positioned using simple distance interpolation. However, object position prediction methods, involving features like last known object position, last known object velocity, trajectory point density and point distribution and so on may be incorporated in a straightforward manner. Such methods serve to minimize (if not eliminate) the user correction needed for each newly added point. An example of the ADD operation is illustrated in Figure 4.3. **Algorithm 15** outlines the procedure for adding point(s) anywhere along the current trajectory. In particular, this operation **creates a new trajectory** if one with the desired ID does not exist.

DELETE point(s) from a trajectory

The DELETE operation allows for the removing the vector data associated with an object at either end of or at any point within a selected trajectory. Deleting point(s) within the trajectory splits it in two, with the section prior to the deleted portion retaining the original trajectory ID, and the following section being given the first new available ID. The trajectory retains its ID when point(s) are deleted from either end. Deleting the whole tra-

Algorithm 15: Trajectory POINT_ADD algorithm

```
Input  : Image sequence $I$, task to add object position(s) for one (or more)
         trajectories purely via user intervention based on visually perceived
         spatio-temporal parameters
Output: One or more points on one or more trajectories added by the user; action
         and result interactively visualized by KOLAM
Activate EditOverlay instance for Trajectory Editing;
Prepare SelectTrajs list<Trajectory >;
Prepare TrajectoryDS data structure;
while EditOverlay is active do
    // ADD point to New trajectory
    if TrajectoryDS.newIDRequested then
        newTraj (new Trajectory);
        newTraj.append(new TrajectoryPoint);
        TrajectoryDS.append(newTraj);
        SelectTrajs.append(newTraj);
    end
    // Or, ADD point to Existing trajectory
    // -ELSE BLOCK HERE-, CONTINUED in next listing.
end
Return to KOLAM's main UI Loop;
```

Algorithm 16: Trajectory POINT_ADD algorithm: Inner 'Else'

```
else
    // Select trajectory to ADD point to
    User selects SelectTrajs.currID();
    pickedID ← SelectTrajs.currID();
    pickedTraj ← TrajectoryDS [pickedID];
    // If current time step is BEFORE trajectory start
       time, ADD user-clicked point to current time and
       extrapolate points for the time steps in between
    if currTime < pickedTraj.startTime then
        timeDiff ← startTime - currTime;
        for $i \leftarrow 0$ to timeDiff$-2$ do
            pickedTraj.prepend(new ExtrapolatePoint);
        end
        pickedTraj.prepend(new TrajectoryPoint);
    end
    // Or, If current time step is AFTER trajectory end
       time, ADD user-clicked point to current time and
       extrapolate points for the time steps in between
    else if currTime > pickedTraj.endTime then
        timeDiff ← currTime - endTime;
        for $i \leftarrow 0$ to timeDiff$-2$ do
            pickedTraj.append(new ExtrapolatePoint);
        end
        pickedTraj.append(new TrajectoryPoint);
    end
    else
        pickedTraj.insert(new TrajectoryPoint);
    end
end
```

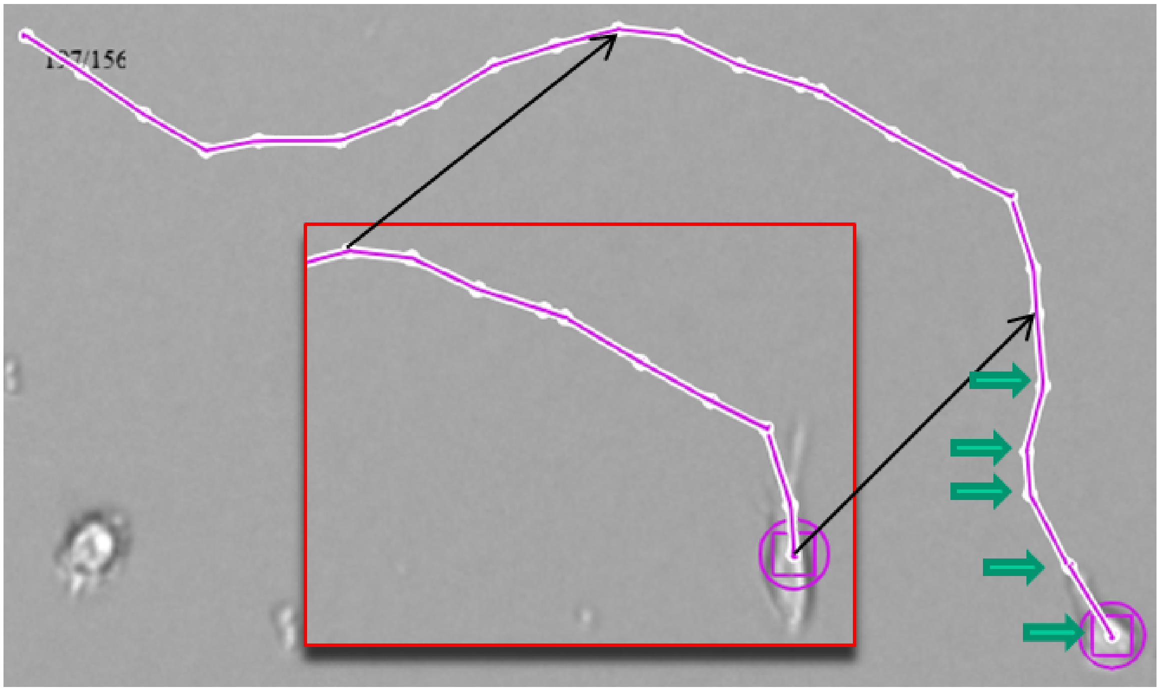


Figure 4.3: ADD Operation of Track Editing. The red inset denotes the original tailing portion of the trajectory. The points indicated by the green arrows were generated by KOLAM 3.0 by interpolation when the user clicked on the last point, enclosed by the square and circle annotation. The MOVE operation was then applied to move the points into their correct positions w.r.t motion of the object. The MOVE operation is described further on.

jectory is a special case of this operation. **Algorithm 17** outlines the procedure for deleting point(s) from anywhere along the current trajectory. In particular, this operation **deletes the current trajectory** when the user deletes the last available point on the trajectory. Furthermore, deleting point(s) that are not at either end of the trajectory results in the **splitting of the trajectory**; this case is handled separately as outlined in **Algorithm 19**. In both latter cases, the IDs of the remaining trajectories are updated as required. Examples of both variants of the DELETE operation are illustrated in Figures 4.4 and 4.5. **Algorithm 19** outlines the procedure for splitting the current trajectory anywhere along its length. This is a special case of deleting a non-terminal point from a trajectory. Splitting creates a new trajectory with a new ID, the new ID being the first freely available ID. This new trajectory is formed from the points that were *temporally antecedent* to the point that was deleted. Points that were *temporally precedent* to the deleted point remain on the original trajectory.

Algorithm 17: Trajectory POINT_DELETE algorithm

```
Input  : Image sequence $I$, task to delete object position(s) for one (or more)
         trajectories purely via user intervention based on visually perceived
         spatio-temporal parameters
Output: One or more points on One or more trajectories deleted by the user; action
         and result interactively visualized by KOLAM
DeleteOp = $\{ToStart, ToEnd, InBetween\}$;
Activate EditOverlay instance for Trajectory Editing;
Prepare SelectTrajs list<Trajectory >;
Prepare TrajectoryDS data structure;
while EditOverlay is active do
    // Select trajectory to DELETE point from
    User selects SelectTrajs.currID();
    pickedID ← SelectTrajs.currID();
    pickedTraj ← TrajectoryDS [pickedID];
    // DELETE from current time to start time
    if DeleteOp == ToStart then
        timeDiff ← currTime - pickedTraj.startTime;
        for $i \leftarrow 0$ to timeDiff$-1$ do
            pickedTraj.startTime $+=1$;
            pickedTraj.popFront();
        end
    end
    // REMAINING CASES continued in Listing 18
end
Return to KOLAM's main UI Loop;
```

Algorithm 18: Trajectory POINT_DELETE algorithm: REMAINING CASES

```
// DELETE from current time to end time
else if DeleteOp == ToEnd then
    timeDiff ← pickedTraj.endTime - currTime;
    for $i \leftarrow 0$ to timeDiff$-1$ do
        pickedTraj.endTime $- = 1$;
        pickedTraj.popBack();
    end
end
// Invoke Trajectory SPLIT operation
else if DeleteOp == InBetween then
    Trajectory SPLIT();
end
// DELETE Trajectory if no more points on it
if pickedTraj $(endTime - startTime) == -1$ then
    TrajectoryDS.delete(pickedTraj);
    Adjust all Trajectory IDs following pickedID;
end
```

MOVE existing trajectory point(s)

The MOVE operation allows for the translation of one or more point(s) of the trajectory. Unlike other operations, MOVE does not involve change of trajectory ID. A noteworthy point is that extending this operation to MOVE the whole trajectory gives one of three cases of a general trajectory transformation operation, the other two cases involving trajectory rotation and trajectory scaling. Generalized transformation operations may be applied to either a part of or to the entire trajectory. Trajectory transformation methods have been explored for volumetric data in fields such as robotics. **Algorithm 20** outlines the procedure for moving point(s) anywhere along the current trajectory. An example of the MOVE operation is illustrated in Figure 4.6.

Algorithm 19: Trajectory SPLIT algorithm

```
Input  : Image sequence $I$, edit op. POINT_DELETE selected with current
         Trajectory in SelectTrajs, task to split Trajectory purely via user
         intervention based on visually perceived spatio-temporal parameters
Output: Trajectory split into two or more trajectories by user; action and result
         interactively visualized by KOLAM; Trajectory with newer start time
         appended to TrajectoryDS by default
Activate EditOverlay instance for Trajectory Editing;
SelectTrajs is ready from POINT_DELETE;
TrajectoryDS is ready from POINT_DELETE;
pickedID is ready from POINT_DELETE;
pickedTraj is ready from POINT_DELETE;
while EditOverlay is active do
    // Screen-to-Image conversion
    UserPt $\leftarrow$ ScrToImg(User-Click);
    for $i \leftarrow 0$ to pickedTraj.ptList.$size - 1$ do
        // Find point on trajectory to perform SPLIT
        currPt $\leftarrow$ pickedTraj.ptList $[i]$;
        testDist $\leftarrow$ EuclidDist(UserPt, currPt);
        // Make sub-trajectory following SPLIT point a new
           trajectory
        if testDist $<$ clickThresh then
            TrajectoryDS.append(new Trajectory);
            for $j \leftarrow i + 1$ to pickedTraj.ptList.$size - 1$ do
                TrajectoryDS [size-1].ptList.append(pickedTraj.ptList $[j]$);
                delete pickedTraj.ptList $[j]$;
            end
            SelectTrajs.append(TrajectoryDS [size-1]);
        end
    end
end
Return to KOLAM's main UI Loop;
```

Algorithm 20: Trajectory POINT_MOVE algorithm

```
Input  : Image sequence $I$, task to move object position(s) for one (or more)
         trajectories purely via user intervention based on visually perceived
         spatio-temporal parameters
Output: One or more points on one or more trajectories moved to new coordinate
         position(s) by the user; action and result interactively visualized by
         KOLAM
Activate EditOverlay instance for Trajectory Editing;
Prepare SelectTrajs list<Trajectory >;
Prepare TrajectoryDS data structure;
clickThresh: Threshold distance to pick trajectory point in proximity to user-click;
while EditOverlay is active do
    // Select Trajectory to MOVE point(s) on
    User selects SelectTrajs.currID();
    pickedID $\leftarrow$ SelectTrajs.currID();
    pickedTraj $\leftarrow$ TrajectoryDS [pickedID];
    // While User MOVEs point(s) on trajectory
    while User moves point(s) on pickedTraj do
        // Screen-to-Image conversion
        UserPt $\leftarrow$ ScrToImg(User-Click);
        // Find the point to MOVE
        for $i \leftarrow 0$ to pickedTraj.ptList.$size - 1$ do
            currPt $\leftarrow$ pickedTraj.ptList [$i$];
            testDist $\leftarrow$ EuclidDist(UserPt, currPt);
            if testDist $<$ minDist then
                minDist $\leftarrow$ testDist;
                dragPt $\leftarrow$ currPt;
            end
        end
        // Sync selected pt display with User Drag
        while User performs drag do
            dragPt $\leftarrow$ ScrToImg(UserPt);
        end
    end
end
Return to KOLAM's main UI Loop;
```

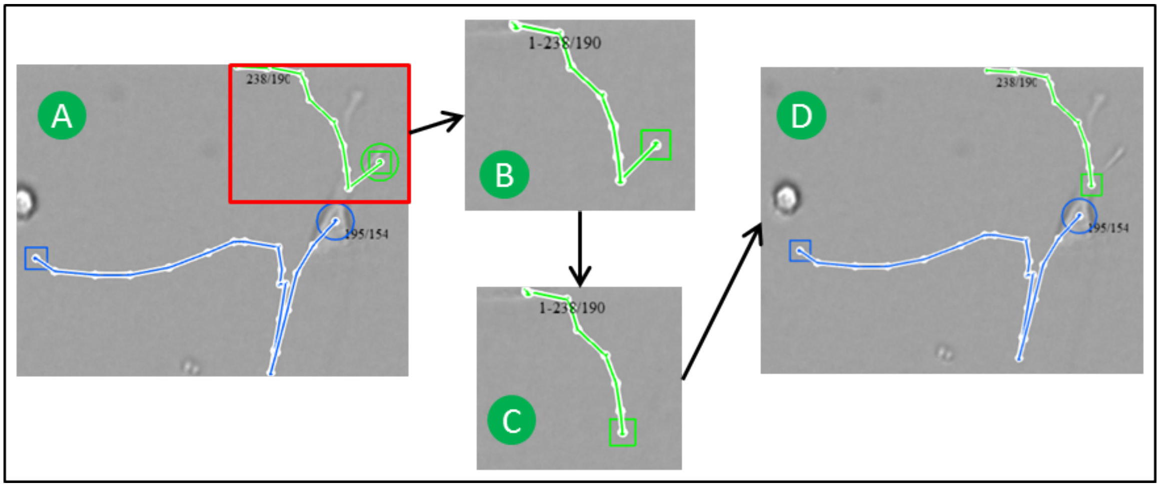


Figure 4.4: DELETE Operation of Track Editing. The last point of the green trajectory (red inset in 'A', separately shown in 'B'), is deleted in 'C', with the overall result being shown in 'D'.

JOIN trajectories

The JOIN operation allows for two or more trajectories to be merged together into a single, longer trajectory. Given that joining three or more trajectories is simply a case of performing multiple pairwise joins, we present the details of performing JOIN on a single pair of trajectories. We then examine the multiple join operation as an extension of the same, and enumerate any additional constraints. A meaningful JOIN operation requires that one trajectory start at an earlier point in time than the other. For ease of description, 'source' and 'target' are aliases for the earlier and later starting trajectories henceforth. The target may start at one of four possible points in time with respect to when the source ends:

- One or more time steps prior to source end,
- At the same time step as source end,
- One time step after source end, and

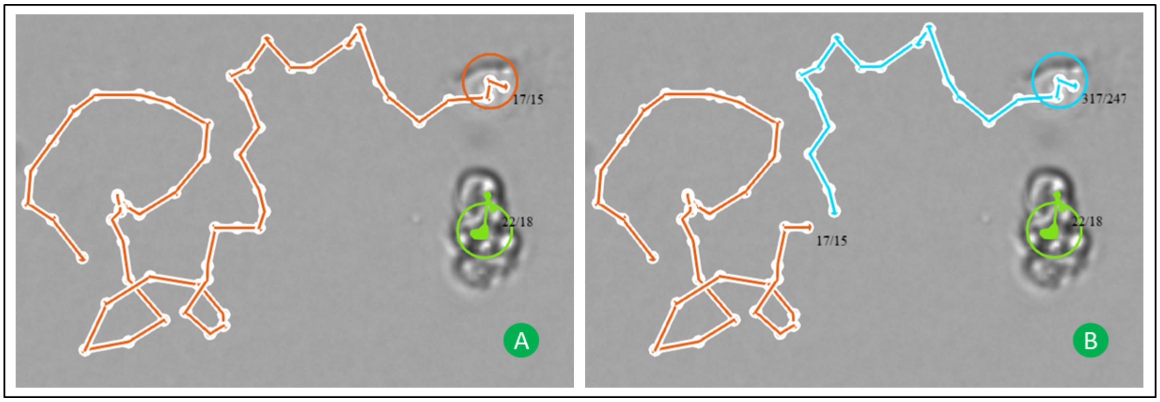


Figure 4.5: The ‘Split’ variant of the DELETE Operation. Note that the portion of the original trajectory preceding the DELETEd point in time has retained the original trajectory ID, while the trajectory fragment following the DELETEd point in time has been assigned the first free ID.

- Two or more time steps after source end.

Of the four cases, the third case is the base case: no trajectory points need be eliminated since each time step has a single associated point.

The fourth case involves trajectory interpolation, and the relevant details presented for ADD apply here as well. However, an additional step is involved: determining the trajectory to which the interpolated points should be added. KOLAM implements the simple rule that new points be added at the end of the source trajectory. Additionally, given any selected trajectory pair, KOLAM flexibly permits the user to designate either member of the pair as source or target. Thus, new points may be added (prepended or appended) to either trajectory. This decision may also be made via methods similar to those outlined for ADD.

Finally, the first and second cases involve elimination of one (the second case) or more (the first case) points from one (in certain situations, both) of the trajectories. It is important that both cases satisfy the following constraint: Regardless of the choice of trajectory for

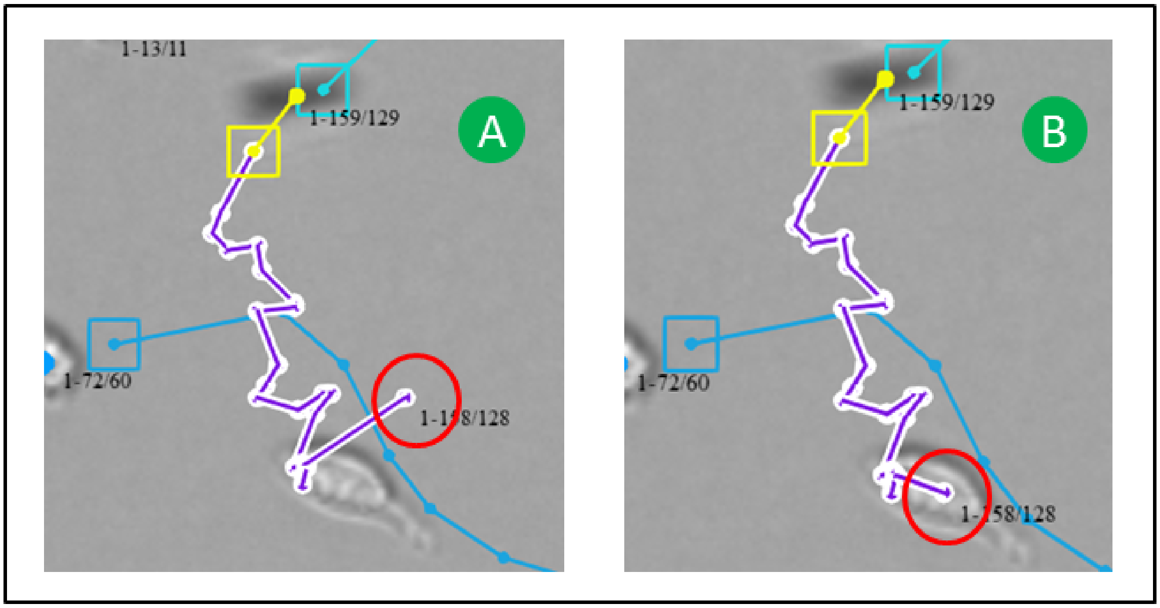


Figure 4.6: MOVE Operation of Track Editing. The end point (incorrect tracker detection) of the selected purple trajectory in ‘A’ is MOVEd by the user to its correct position in ‘B’.

point deletion, the resulting merged trajectory should continue to represent the motion of the object without loss of accuracy. Let us clarify this point further.

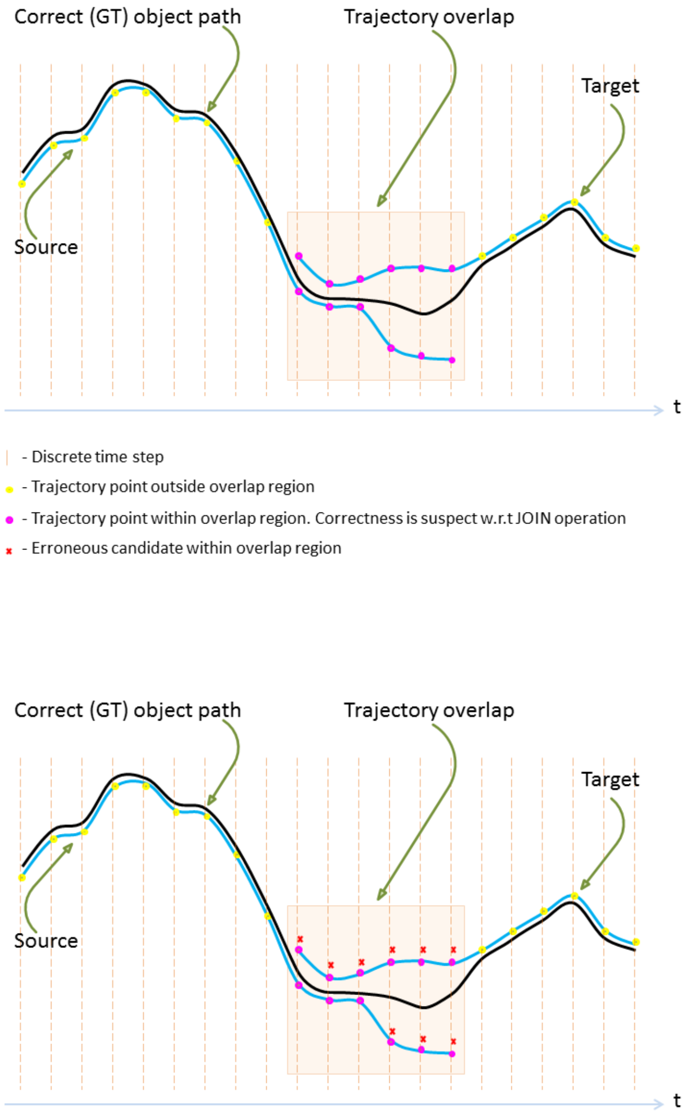
For the temporally overlapping portion of the two trajectories, two candidate points are present at each time step. In determining the candidate for deletion in each such pair, the optimal solution (that which minimizes the needed number of user corrections) is to discard that candidate which is spatially further from the actual position of the object. Depending on the nature of the specific dataset and the specific object being tracked, candidates for elimination might be at the end of the source, at the beginning of the target, or might be on both sides. This ’stitching’ problem is non-trivial, and whether the latency of the chosen algorithmic solution can be accommodated while retaining the real-time interactivity of the JOIN operation is uncertain. Given this difficulty, KOLAM currently provides a couple of simple solutions (which rely on user input rather than being automated): Given the case of erroneous points being on only one trajectory, KOLAM uses the rule that points be deleted from the target, and allows the user to designate either trajectory as source or target. If erroneous points are present in both trajectories, the user may use the DELETE and MOVE operations to partially correct the trajectories prior to using the JOIN operation. The partial overlap scenarios for the JOIN operation are illustrated in Figures 4.7, 4.8, 4.9 and 4.10.

Algorithm 21 outlines the procedure for joining a pair of trajectories. The trajectories may be chosen in any order, and the join is performed from the temporally precedent (with earlier start time) trajectory to the temporally antecedent (with later start time) trajectory. In the case of temporal overlap between the two trajectories, extra points in the temporal overlap zone are deleted from the temporally precedent trajectory.

4.4 Conclusions and Future Work

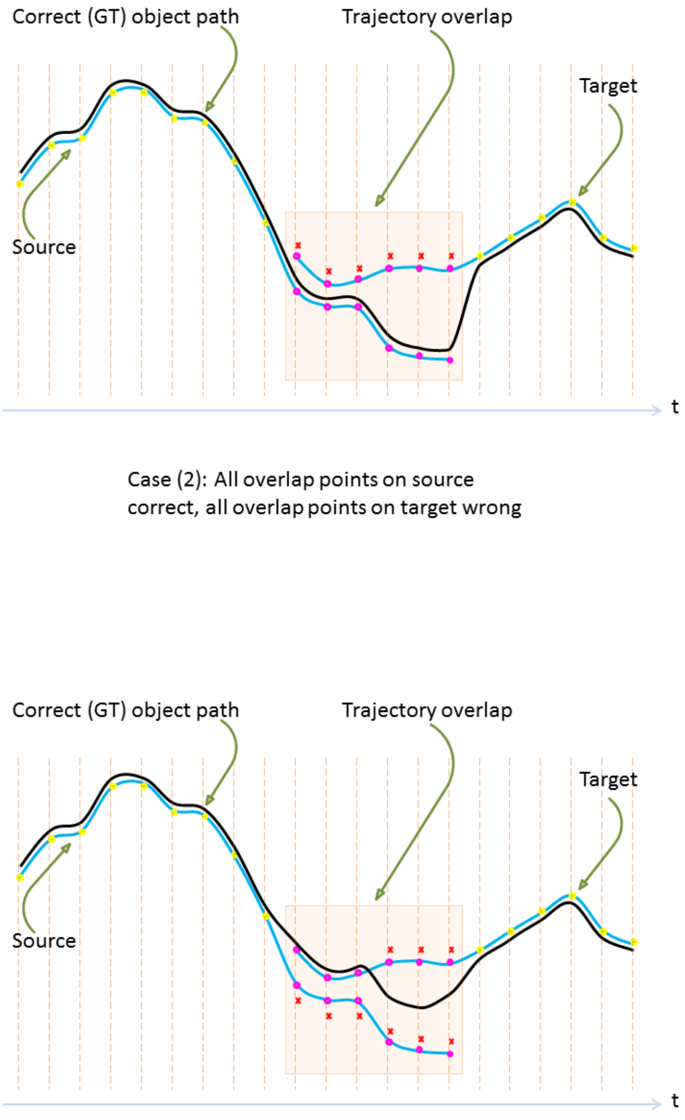
KOLAM is demonstrated to be a scalable and flexible tool for exploratory analysis of WAMI datasets that provides dense spatiotemporal coverage of wide-field urban regions and can be used for object detection and tracking. The current KOLAM environment supports an efficient tiled memory representation of very large time series of images using a spatial and temporal dual-caching mechanism. The multithreaded synchronization architecture ensures that tile access and tile display processes work smoothly with each other and can be readily scaled with the availability of additional processing cores and memory. KOLAM is particularly designed to facilitate user interaction and user-driven WAMI exploitation.

KOLAM is able to improve human effectiveness measured in terms of productivity for accurately tracking multiple targets and reviewing and validating the results of track-


Correct (GT) object path
Trajectory overlap
Target
Source
t
| - Discrete time step
● - Trajectory point outside overlap region
● - Trajectory point within overlap region. Correctness is suspect w.r.t JOIN operation
× - Erroneous candidate within overlap region
Correct (GT) object path
Trajectory overlap
Target
Source
t

Case (1): Some overlap points on source correct, all overlap points on target wrong

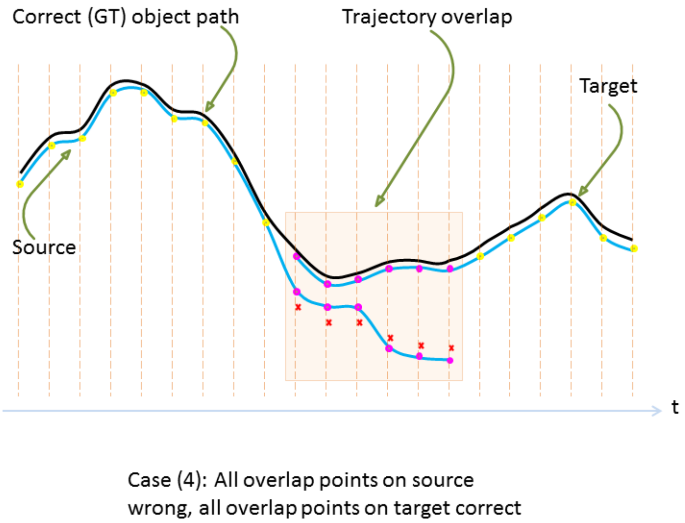
Figure 4.7: JOIN Operation with 'source' and 'target' trajectory overlap. Legend on TOP, Case (1) at BOTTOM



Case (2): All overlap points on source correct, all overlap points on target wrong

Case (3): All overlap points on source wrong, some overlap points on target correct

Figure 4.8: JOIN Operation with ‘source’ and ‘target’ trajectory overlap. Case (2) on TOP, Case (3) at BOTTOM


Correct (GT) object path
Trajectory overlap
Target
Source
t
Case (4): All overlap points on source wrong, all overlap points on target correct

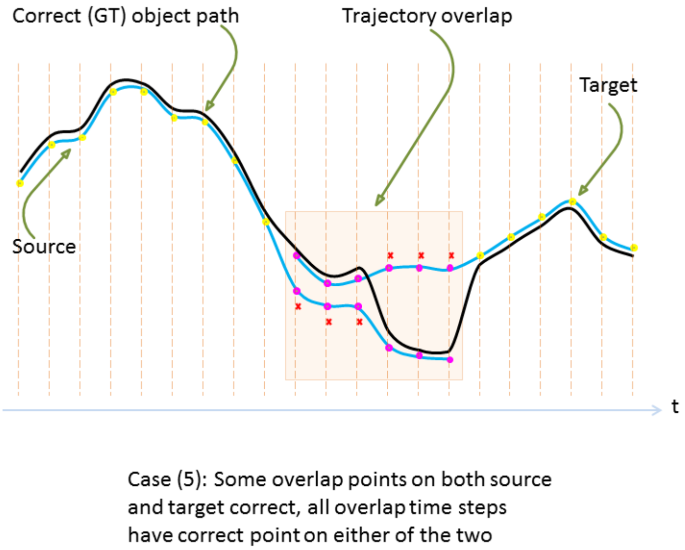
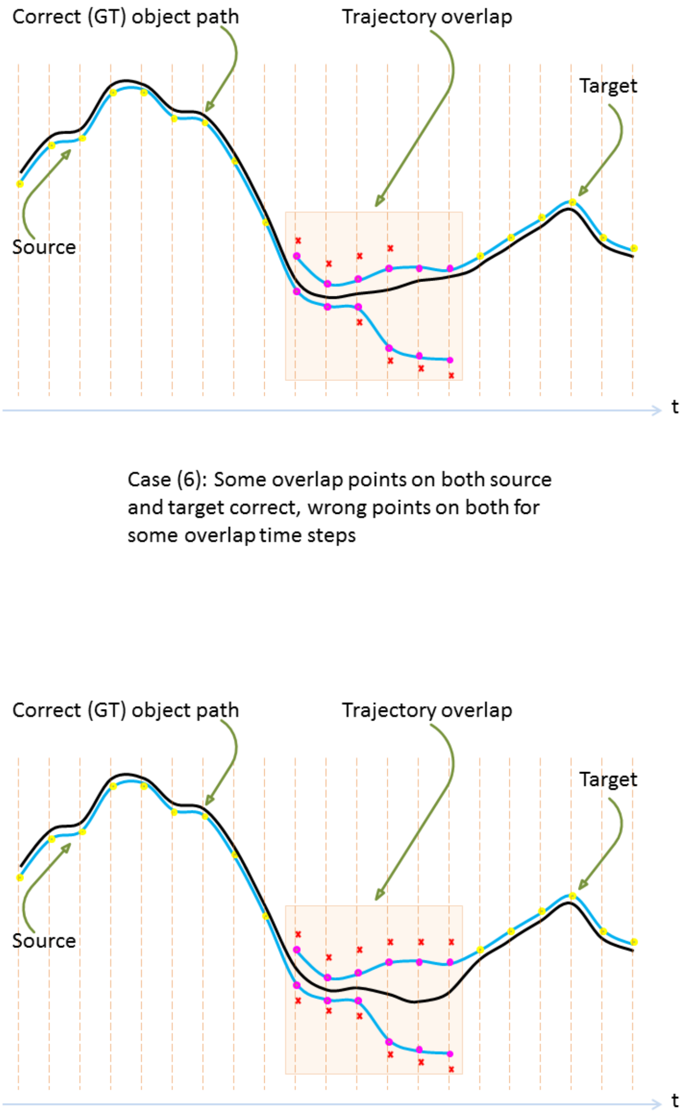

Correct (GT) object path
Trajectory overlap
Target
Source
t
Case (5): Some overlap points on both source and target correct, all overlap time steps have correct point on either of the two

Figure 4.9: JOIN Operation with ‘source’ and ‘target’ trajectory overlap. Case (4) on TOP, Case (5) at BOTTOM



Case (6): Some overlap points on both source and target correct, wrong points on both for some overlap time steps

Case (7): All overlap points on both source and target wrong

Figure 4.10: JOIN Operation with ‘source’ and ‘target’ trajectory overlap. Case (6) on TOP, Case (7) at BOTTOM

Algorithm 21: Trajectory JOIN algorithm

```
Input   : Image sequence $I$, task to join a pair of trajectories purely via user
          intervention based on visually perceived spatio-temporal parameters
Output: A pair of trajectories joined by the user; action and result interactively
          visualized by KOLAM; remaining trajectories have their IDs adjusted to
          account for eliminated IDs
Activate EditOverlay instance for Trajectory Editing;
Prepare SelectTrajs list<Trajectory >;
Prepare TrajectoryDS data structure;
while EditOverlay is active do
    // Need at least 2 trajectories to perform JOIN
    if TrajectoryDS.size $< 2$ then
        Create new Trajectory newTraj with newObj;
        // Automatically track additional object
        if User wants to automatically track newObj then
            Perform_Automatic_Tracking() with newTraj;
        end
        // OR, Visually track additional object
        else if User wants to manually track newObj then
            Perform_Visual_Tracking() with newTraj;
        end
        TrajectoryDS.append(newTraj);
    end
    // WHILE Loop; CONTINUED in Algorithm listing 22
end
Return to KOLAM's main UI Loop;
```

Algorithm 22: Trajectory JOIN algorithm: INNER WHILE

```
while Need to do JOIN && TrajectoryDS.size $> 1$ do
    // Select 2 trajectories to JOIN
    while pickCount $< 2$ do
        if pickCount $== 0$ then
            IDOne ← User selects SelectTrajs.currID();
            pickCount ← pickCount $+1$;
            continue;
        end
        IDTwo ← User selects SelectTrajs.currID();
        pickCount ← pickCount $+1$;
    end
    FStart ← TrajectoryDS [IDOne].startTime;
    FEnd ← TrajectoryDS [IDOne].endTime;
    SStart ← TrajectoryDS [IDTwo].startTime;
    SEnd ← TrajectoryDS [IDTwo].endTime;
    // If they temporally overlap, delete ending pts of temporally precedent trajectory, perform JOIN and update IDs
    if FStart $<$ SStart && FEnd $>$ SStart then
        while FEnd $>=$ SStart do
            TrajectoryDS [IDOne ].popBack();
        end
        tempTraj ← TrajectoryDS [IDTwo ];
        TrajectoryDS [IDOne ].append(tempTraj);
        delete tempTraj and update IDs;
    end
    else if SStart $<$ FStart && SEnd $>$ FStart then
        while FEnd $>=$ SStart do
            TrajectoryDS [IDTwo ].popBack();
        end
        tempTraj ← TrajectoryDS [IDOne ];
        TrajectoryDS [IDTwo ].append(tempTraj);
        delete tempTraj and update IDs;
    end
end
```

ing especially in the assisted tracking mode. As described in detail in this Chapter, the author's work in conceiving and implementing trajectory visualization and trajectory editing operations within the KOLAM system constitutes a crucial component of the solution to the problems and challenges involved in assisted object tracking, as well as automatic and manual object tracking. Visual analytics and visual synopsis will be useful directions to deal with the large data volume from WAMI sensors. Future improvements include support for IR and multispectral WAMI, a knowledge base for managing thousands of trajectories from automated and interactive analysis of events, developing techniques to visualize statistical trajectory flow information, distinguishing between normal and abnormal patterns of activity, and processing event-based queries using semantic models.

Chapter 5

CASE STUDY 3: Interactive Segmentation Relabeling for Histopathology Applications

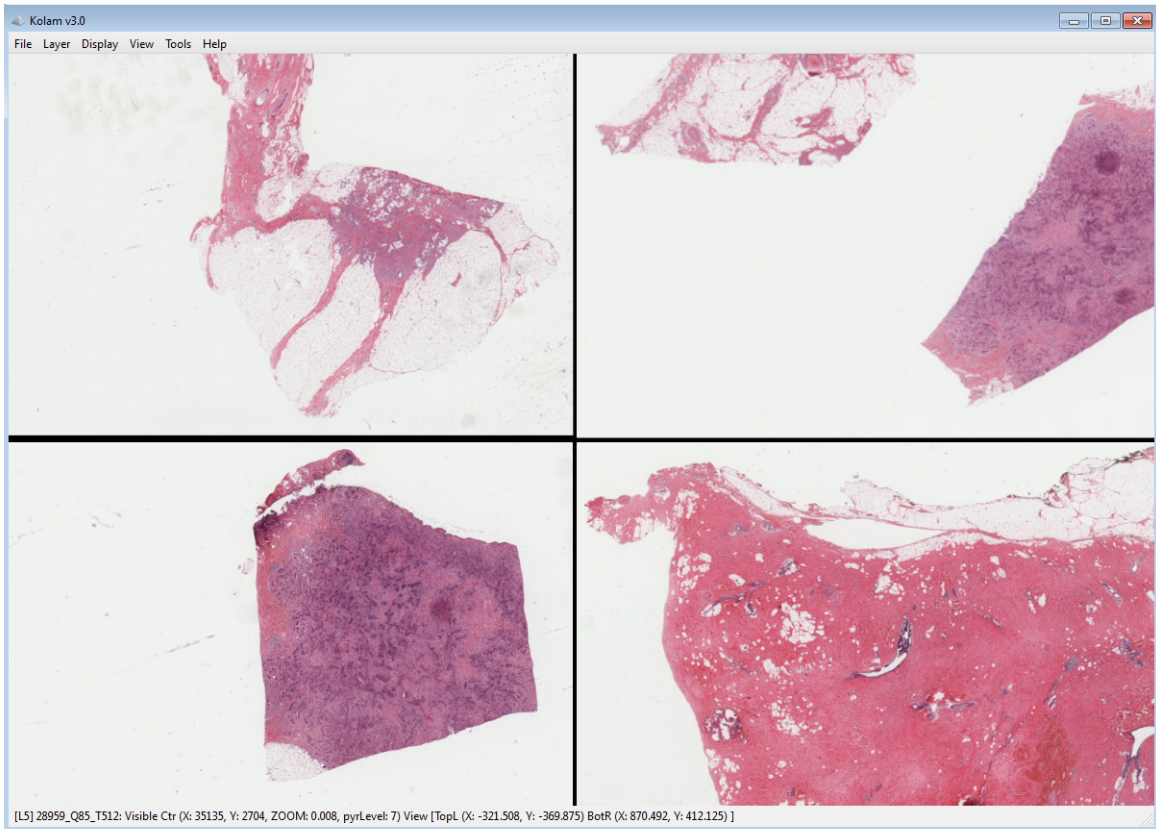
5.1 Motivation

Within the cancer research community, it is being increasingly appreciated that tumor stroma constitutes an integral part of cancer initiation, growth, and progression; therefore, it holds valuable prognostic and response-predictive information [64] [65] [66] [67] [68] [69] [70]. Stroma tissue surrounding cancer cells plays an important role in tumor development and behavior [67]. It is widely recognized that tumor progression and metastasis are intimately linked to tissue remodeling resulting from tumor cell interactions with the host tissue stroma [68]. On Haematoxylin-Eosin (H&E) stained histopathology images, the impact of stroma on patient outcome is mostly studied through tumor-stroma area ratio. In [67] Kruijf et al. showed that tumor-stroma ratio is a prognostic factor for relapse-free period. Dekker et. al. [69] observed that tumor-stroma ratio was associated with disease-free survival. For both studies, tumor-stroma ratio was determined by visual estimation. In a more comprehensive study [64], **automated analysis of H&E slides** has revealed that the stromal regions of breast tumors contain more prognostic information than the epithelial regions. The 'C-Path' system [64] that was developed for this purpose combined a large set (more than 6600) of image-derived morphometric features with image classification and machine learning techniques. In following the 'human-in-the-loop' paradigm, investigating whether a system that allows for the coupling of human expert intervention (via ground truth generation) with a *smaller* set of automatically derived features is capable of achieving similar or superior performance presents an interesting problem. The work described in this chapter attempts to lay a foundation to solving this problem.

Collecting ground-truth or gold standard annotations from expert pathologists for developing histopathology analytic algorithms and computer-aided diagnosis for cancer grading is an expensive and time consuming process. At this juncture, the term 'gold standard' must be defined, from a medical and statistical perspective. A **gold standard** test commonly denotes a diagnostic test or benchmark which is the best available under *reasonable conditions* [71]. It may also refer to the most accurate test possible *without any restrictions*. By a different definition, a hypothetical *ideal* gold standard test has a sensitivity of 100% and a specificity of 100%. However, such values are nearly impossible to obtain, and therefore in conjunction with the prior definitions, a gold standard test is one with the highest possible values for sensitivity and specificity under reasonable conditions. To conclude, the phrase 'gold standard' can be ambiguous, and therefore its relevant meaning must be deduced from the context of usage.

Efficient visualization and annotation tools are needed to enable ground-truthing large whole-slide imagery. As elaborated above, we thus have a new domain-specific motivation and problem statement, with the challenges involved being largely the same as encountered in previous chapters. In summary, the solution involves providing support for rapid interactive labeling and correction of automatic image classifier-based region labels of the tissue micro-environment by pathologists. In implementing the solution, KOLAM; which is already implemented as a scalable, cross-platform framework for interactive visualization of 2D, 2D+t and 3D imagery of high spatial, temporal and spectral resolution; has been extended to perform these tasks by the author [72]. Indeed, examples such as the one illustrated in Figure 5.1 demonstrate how KOLAM's extensible nature made visualization of the big-data histopathology image data a straightforward task that involved few changes to the existing architecture besides adding support for the requisite image file formats [72].

Besides annotating regions-of-interest (ROIs), KOLAM was extended to enable extraction of the corresponding large polygonal image sub-regions for input into automatic segmentation algorithms, single-click region label re-assignment and maintaining hierarchical image sub-regions [72]. Experience indicates that clinicians prefer simple-to-use interfaces that support rapid labeling of large image regions with minimal effort. The incorporation of easy-to-use tissue annotation features in KOLAM makes it an attractive candidate for integration within a **multi-stage histopathology image analysis pipeline** (consisting of *pre-processing*, *segmentation*, *feature extraction*, and *classification* modules [73]) supporting assisted segmentation and labeling to improve whole-slide imagery (WSI) analytics [72].



[L5] 28959_Q85_T512: Visible Ctr (X: 35135, Y: 2704, ZOOM: 0.008, pyrLevel: 7) View [TopL (X: -321.508, Y: -369.875) BotR (X: 870.492, Y: 412.125)]

Figure 5.1: KOLAM 3.0 diplaying several histopathology BigTIFF images interactively arranged in a grid. The dimensions and raw sizes of these big-data images follow. Top Left: 146878 x 79128 px, 46.5 GB. Top Right: 83519 x 57591 px, 19.2 GB. Bottom Left: 113279 x 38616 px, 17.5 GB. Bottom Right: 117119 x 84257 px, 39.5 GB. KOLAM 3.0 can fluidly translate and zoom into each image separately, or transform them as a group.

5.2 Introduction

Clinical decision support systems (CDSSs) play a crucial role in the clinical process, from diagnosis and investigation to treatment and long-term care [74]. CDSSs are currently facing a deluge of image data from multiple sources including radiology and pathology. Advances in imaging techniques, high-throughput technologies, pathology informatics, bioinformatics, need for exploratory analysis of WSI and personalized medicine data sets con-

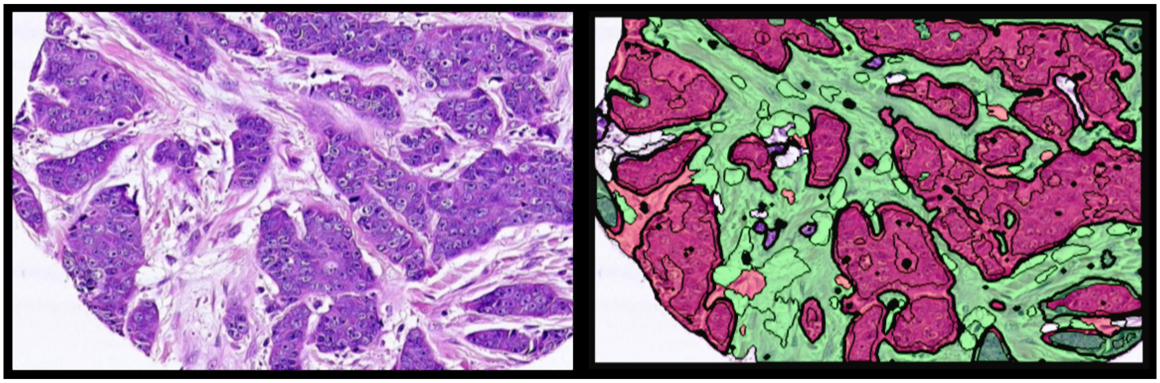


Figure 5.2: Breast Cancer slide section and segmentation results side-by-side.

tinue to generate big data volumes in need of immediate analysis. The web-based Cancer Digital Slide Archive [75] is one such repository, that integrates imaging, genomic and clinical datasets.

The approach of manually labeling patient biopsy slides for automatic algorithm development has its advantages: it leverages the ability of the pathologist expert to quickly intuit relevant meaning from slide appearance, as well as the robust invariance of his/her (human) visual system to irrelevant variations in image properties such as contrast and illumination. However, it is not scalable with the burgeoning volume of biopsy data, due to subjectivity of interpretation by pathologists, fatigue and likelihood of errors. The expert variability (in observer training, ability, experience, alertness and timing) in the visual inspection process is quite high, with up to a 20 percent discrepancy between different reviewers [76]. The important consequence of such variability is the potential danger of disagreement regarding course of treatment options among pathologists. In contrast, the image analysis component of CDSS is objectively versatile, consistent, inherently multiplexed, high in information content and far less labor-intensive. Furthermore, the image analysis component of CDSS is also capable of capturing small but significant biological characteristics that are undetectable by the human eye. By providing efficient methods for objective quantification of image features, CDSSs have emerged as useful decision-making tools involving biomedical imaging.

It is known that a foundational principle of cancer histopathology is the direct correlation between cancer cell biology and tissue morphology and the tumor microenvironment [77]. The C-Path system [64] is the first example of a multi-stage system that directly correlates tumor morphological patterns to outcomes in order to assist in cancer characterization, diagnosis and prognosis in a versatile and groundbreaking manner. In contributing to the goal of developing a system titled 'M-Path' which could potentially achieve similar goals, the author designed and implemented an improved human-computer interface for rapid annotation of the tissue microenvironment in histopathology WSI using a combination of automatic image classification output with manual expert region label correction, within the KOLAM visual analytics framework [72]. This KOLAM subsystem is focused on further investigating the correlation between microenvironment image features derived from stromal and epithelial regions and patient survivability beyond five years post-diagnosis, following the work done in [64]. In requiring a means to identify and distinguish between stromal and epithelial regions in hematoxylin and eosin (H&E) stained histopathology digital slides, we examined results derived from a number of approaches ([78] [79] [80] [81] [82]). We chose to integrate the KOLAM system with [73], which uses a hierarchical fuzzy c-means (HFCS) based segmentation system to partition the images into coherent partitions (super-pixels), from which regional color and texture features are extracted, followed by classification by a support vector machine (SVM) based classifier of the regions into stroma and epithelium (See Figure 5.2 for an example of the input data and the output as visualized in KOLAM). The benefits of this approach are two-fold: the partitions provide *larger support regions* for the features used in classification, and *ameliorate blending problems* encountered when the neighborhood around a pixel contains both epithelial and stromal regions [73]. The overall M-Path system [72] [73] is illustrated in Figure 5.3.

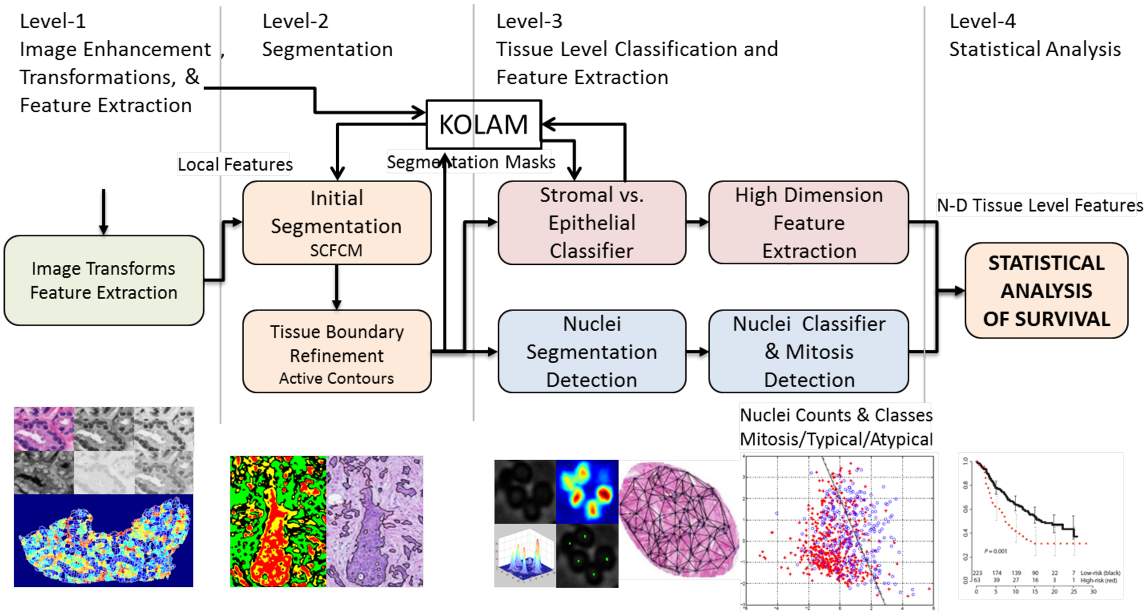


Figure 5.3: The M-Path pipeline.

For handling the type of problem like epithelia-stroma classification in imagery, two approaches have been widely employed : (a) perform the task with extensive (complete) manual intervention, or (b) develop scripts, macros and plugins which operate with well-known software packages, such as Metamorph, ImagePro Plus and MATLAB. The system we have developed is capable of performing in both roles: it has been developed as a dedicated system capable of solving this type of problem, and can additionally be enhanced via plugins and scripts. With the goal of positioning KOLAM [39] [40] [45] [9] as a cross-platform and scalable digital light table or virtual microscope framework, it has been extended to support visualization and fast, reproducible semi-supervised annotation of histopathology slides. KOLAM also provides interfaces for (semi-)automated labeling and manual correction or relabeling of stromal, epithelial and other regions of interest in the tissue micro-environment. The potential benefits of our system are two-fold. Firstly, utilizing automated classification modules permits the grading of patient cancer on the basis of morphological criteria to be performed objectively. Following automatic image segmentation, the discrepancies with the ground truth that invariably arise necessitate manual verification and correction of the segmentation. Leveraging KOLAM's interactive label annotation features, experts may dynamically alter patient cancer grades and iteratively improve the performance of the classification module. Secondly, by using KOLAM's features to visually analyze segmented and classified patient biopsy results before and after treatment, experts have more opportunities to potentially make new judgments and discoveries regarding how morphological features of interest have changed in response to said treatment over time.

The following section describes related work, including the state-of-the-art segmentation, classification, ROI extraction etc involving histopathology WSI data; as well as the leading proprietary and open-source software tools capable of handling and processing such imagery. Section 5.4 begins with a description of KOLAM's relevant features and handling of various WSI data aspects. The rest of the section describes KOLAM's interactive segmentation relabeling module and its use in ground-truth generation for creating training cases. These training cases are for histopathology image analysis as well as for interactively correcting any automated classification results. Section 7.5 presents details about our evaluation methodology; namely the factors we took into consideration while designing our experimental setup for evaluating different interactive relabeling software. Section 7.6 covers the experimental procedure, and in Section 7.7 we present the experimental results of quantitative and qualitative evaluation of KOLAM for the task of interactive histopathology segmentation/classification result relabeling. The final section of this chapter includes our concluding remarks and a brief outline of ongoing and planned future work on our system.

5.3 Related Work

There have been several algorithmic advancements pertaining to WSI data analysis in recent years. Beck et al. [64] have designed the 'C-Path' system which attempts to comprehensively analyze a large set of automatically quantified morphological features, thereby trying to identify characteristics of prognostic relevance and provide an accurate and reproducible means for assessing prognosis from microscopic image data. The set of features includes standard morphometric descriptors of image objects and higher-level contextual, relational and global image features.

Kothari et al. [83] present a framework which they use to demonstrate qualitative association between biologically interesting WSI regions and common histopathological image features, the usability of supervised analysis to help translate ROI-relevant histopathology domain knowledge into image features, and that portions of WSIs may be clustered into biologically relevant regions by sets of multiple image features. Gutierrez et al. [84] propose a means of emulating the pathologist's process of identifying diagnostically relevant ROIs by modeling the human visual recognition process. It involves simulating the effect of the V1 region of the visual cortex and its interaction with the V2 and V4 regions, thereby attempting to capture the groupings possessing similar histopathological meaning. Romo et al. [85] have formulated a semi-automatic method for extracting ROIs, which upon partitioning a Virtual Slide (VS) arbitrarily into sub-blocks and being assigned a (expert pathologist determined) distance measure, utilizes a non-linear combination of projection of the sub-blocks into low-level feature defined subspaces. A ranking score is then used to define several (not necessarily connex) levels of relevancy. Raghunath et al. [86] assessed pathologists' visual search performance by analyzing their mouse cursor and eye-gaze movement while they viewed WSI data of pathological specimens. They attempted to determine whether cursor movement data is a reliable indicator of physician attention and diagnostic behavior by examining the spatial coupling between eye-gaze and cursor positions.

We now outline a number of state-of-the-art segmentation and classification strategies that have been utilized on WSI data. As opposed to the explicit segmentation module whose results are passed to an SVM-based classifier module used in the pipeline that KOLAM has been integrated with, research efforts by other groups ([87], [88]) utilize pixel or block-based epithelium-stroma classifiers. DiFranco et al. [89] have devised a system that produces clinically relevant heat-maps of tumor presence probability, using sparsity of data derived from a limited number of training WSIs. Expert-annotated tiles from the training images are used to build an ensemble of classifiers, and individual classifier responses are used to compute cancer presence probability scores. Visualization is then performed via ensemble classification of the query image tiles. Roullier et al. [90] propose a graph-based multi-resolution segmentation strategy for breast cancer WSIs, that seeks to mimic pathologist interpretation under the microscope as focus of attention. Kong et al. [91] worked on multi-resolution hierarchical images to perform adaptive segmentation across the resolution levels based on confidence measure output from a classifier combination mechanism. Doyle et al. [92] utilize a boosted Bayesian multi-resolution classifier to initially identify suspicious regions at lower resolutions of the WSI, and analyze successive image resolutions in this fashion until a spatial map of the disease extent is obtained. Huang et al. [93] propose means to improve the efficiency of WSI handling in terms of speed and precision by incorporating a multi-scale dynamic sampling approach and GPU processing capabilities.

Another aspect of handling WSIs is the increasing number of custom data formats. Increasing clinical and commercial interest in digital (histo)pathology alongwith the current non-standardized proprietary large image formats has led to the Digital Imaging and Communications in Medicine (DICOM) Standards Committee proposing Supplement 145 for WSI, compatible with current enterprise-wide Picture Archiving and Communication System (PACS) software widely used by hospital information systems for radiology imagery [94]. The DICOM standard for histopathology will enable the electronic display, sharing, storage and management of large multi-resolution multi-planar WSI in a tiled pyramid image format. Each tile can be compressed using JPEG, JPEG2000 or other compression and coding methods. The OME Remote Objects (OMERO) [95] software platform is a collaborative effort towards building open-source data access and interoperability tools for WSI data.

WSI handling also requires capable visualization systems. Jeong et al. [96] present a novel visualization framework capable of smooth interaction with extremely large histopathology WSIs, which employs a local, adaptive data structure with demand-driven processing: this display-aware paradigm accesses image data via lazy, on-demand evaluation, thereby not having to manage fully processed global image pyramids unlike other existing methods. Imaris [97] is an extensive software suite primarily focusing on image processing operations for multi-channel 4D imagery. The industry-leading Visualization Toolkit (VTK) [98] is a comprehensive and mature C++ API for performing 3D computer graphics, visualization and image processing operations. ImageJ, the Java-based image processing program developed at the National Institutes of Health, has an open architecture extensible via Java plugins and recordable macros [99]. Fiji, a distribution of ImageJ focused on biological-image analysis, combines software libraries with scripting languages to enable rapid prototyping of image-processing algorithms [100]. The ePathology ImageScope [101] allows for panning and zooming, stain comparison, image analysis and more on digital slide images. ilastik [102] is an interactive segmentation and classification system that infers problem specific segmentations based on user-generated labels. The real-time feedback it provides allows for interactive refinement of the segmentation result, thus further fine-tuning the in-built random forest classifier. Cytomine [103] is a rich internet application integrating popular web and open-source technologies to enable remote visualization and collaborative annotation of digital slide imagery. Sedeen 5.0 [104] supports a number of WSI data formats, provides extensive annotation capability and other features including multi-modal overlays, image registration, cropping and resizing. GRAPHIE [105] is a visual analytics tool to explore, annotate and discover potential relationships in histology image collections with a biologically relevant context. By representing each image with informative features and subsequently visualizing the image collection as a graph, GRAPHIE allows users to effectively explore the image collection. Interactive feature selection for improved image collection visualization and annotation is possible.

5.4 KOLAM for Histopathology VisAnalysis

KOLAM is a large data / information visualization environment and visual analytics system with robust support for out-of-core multiresolution tiled imagery across application domains [39] [40] [45] [9]. In addition to the functionality of a virtual microscope [106] or light table, KOLAM has been extended by the author to support WSI data and is currently being used for generating ground-truth or supervised region labels for input to image classification training algorithms. The suitability of KOLAM for WSI (big data) handling stems from the following aspects of WSI. Firstly, WSI come with a wide variety of image and metadata formats, with a variety of format specifications and proprietary aspects; secondly, issues pertaining to the large images, of the order of tens of gigabytes; and finally, the need for an intuitive yet efficient human computer interface for interactive visualization, navigation, manipulation and labeling. We also note that the complexity of the ground-truth collection process (with respect to H&E stained histopathology imagery data) is typically governed by (a) the quality of the imaging system, (b) standardization of data access [95], (c) clinical diagnostic requirements, and (d) access to clinical patient data.

We now explain the solutions implemented in KOLAM that provide its capability to handle these aspects. Per Keim et al [107], visual analytics combines the tasks of visualization, human intervention and data analysis to make decisions about the data in an integrated fashion. Through our choice of segmentation and classification algorithms, we have selected superior candidates for our specific WSI analysis tasks, clearly defined the limits of (automated) operation of these algorithms on the data, consequently developing visualization and user interactive capabilities in KOLAM which complement our algorithms together in the best possible tightly integrated solution. The KOLAM interface environment aims to provide the pathologist with easy-to-use visualization and annotation capabilities

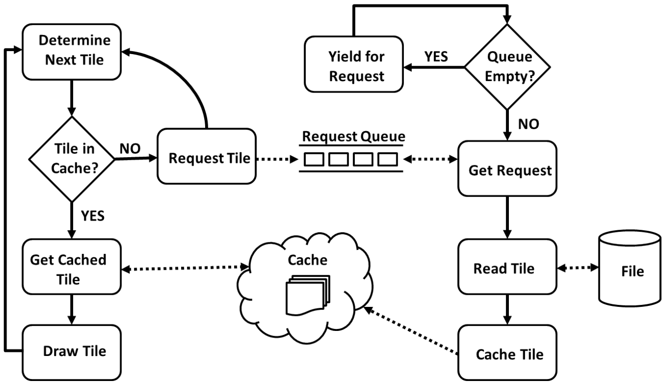


Figure 5.4: KOLAM's multi-threaded image rendering engine, showing the roles and interactions of the tile cache, tile request queue, display and reader threads for managing large multiscale tiled imagery with multiple layers of information including segmentation and classifier label annotations.

in a way that captures the expert's knowledge that is scalable [95]. Similar goals have been pursued and implemented in [102], [103] and [105]. Furthermore, these capabilities along with the intuitive feedback provided by KOLAM's interface allow the expert to interactively refine the results generated by segmentation, thereby further tweaking the input to the classifier module in a meaningful way. The KOLAM GUI facilitates and enhances opportunities for scientists and experts to make new inferences using the rich feature set of visanalysis tools, that would otherwise be difficult. The typical workflow for histopathological image analysis that has been observed in the literature comprises identifying regions of interest (ROI), feature extraction, feature selection and supervised classification.

5.4.1 Big Data Out-of-Core Visualization

As seen previously, KOLAM handles big data imagery by exploiting a multi-scale tiled memory efficient data structure [9] and out-of-core cache management strategies [39] as shown in Figure 5.4. KOLAM's multi-threaded image-rendering engine is capable of rapidly displaying multiple images and information layers as well as a temporal collection of multi-image (sequence) layers for display. The partitioning of each image into tiles allows fast, scalable random access to spatially coherent regions of the imagery across resolutions. This permits on-demand access to arbitrary regions and scales of the image needed for display, as well as application-level out-of-core management of memory [39]. Multiresolution (pyramidal) tiling allows for the interactive zooming of the image across scale [9]. Use of thread synchronization and CPU load balancing schemes allow KOLAM to manage out-of-core I/O and CPU demand patterns for visualizing large, tiled imagery. Furthermore, KOLAM's spatio-temporal dual cache system [39] permits the interactive visualization and playback of multiresolution image sequences. By using KOLAM features such as overlays, altering projection parameters, or interactively modifying the image resolution level the user can generate multiple views of the data by composing and organizing it in different ways. Additionally, custom view settings for data generated by one user may be stored and thus shared with other users of the WSI. All these features of KOLAM were covered in detail in Chapter 2.

5.4.2 Support for Multiple WSIs and Metadata Types

A number of image formats capable of supporting the huge dimensions and metadata requirements of WSIs have emerged recently. The Aperio '.svs' format is being supported in microscopy software from multiple vendors. The BigTIFF format can also handle the large sizes of WSI bio-images. In addition to handling these formats by supporting the OpenSlide [108] and LibTIFF libraries, KOLAM was also extended to provide access to a large single collection of popular microscopy image formats via its interfacing with the LOCI Bio-Formats [109] library. KOLAM provides a uniform interface for access to metadata and can query or update metadata information for all supported image file formats.

5.4.3 WSI Pre-processing

Prior to undergoing the imaging process which generates the WSIs, the tissue samples are processed, embedded, sectioned and stained. McCann et al. [110] and Rolls [111] both provide a description of these stages of the *pre-analysis* pipeline. They [110] also summarize the sources of image variability in the WSIs that arise due to different problems which occur during the pre-analysis stages. Consequently, the histopathology WSI dataset needs to be pre-processed, prior to undergoing either segmentation, classification, expert-driven ground truth generation or label re-assignment.

Regardless of whether or not the imagery has undergone some corrective processing by the image acquisition software suites, illumination often varies by up to 150% across the field of view. Such variations are caused either by uneven incoming illumination, sensor bias, lens and slide imperfections or operator error. Secondly, the staining stage is responsible for both color and contrast. The intensity of a given stain depends on: (a) original stain strength, (b) the staining procedure, (c) percentage fading since original processing of the sample, and (d) how much of the cellular substance of interest is present in the material [112]. Non-uniformity in the staining procedure is a critical source of variability. Together, such sources add noise at unacceptable levels to the imagery, obscure true quantitative differences and make all experiments dependent on fluorescence intensity measurements impractical [113]. The need for accuracy in the quantitative comparisons being performed between KOLAM and other software systems further underscores how critical illumination correction and noise removal are to this work.

To this end, a good WSI pre-processing module would incorporate techniques such as anisotropic diffusion filtering [114] for noise removal while preserving features, and contrast limited adaptive histogram equalization (CLAHE) [115] for illumination correction. The problem of staining variability is addressed by stain normalization of the WSI data. A number of approaches are described in the literature; for both when the stain colors are known [116], and when they are not [112] [117] [118] [119].

5.4.4 Interactive Region-of-Interest (ROI) Selection

The first step prior to ground-truth generation involves the clinicians interactively visualizing the WSI data, and annotating ROIs (ie. regions with abnormal tissue characteristics that may be cancerous) via KOLAM's overlay drawing planes. These annotations are saved in XML format and may be reloaded for later use or shared between users. These regions are sometimes very large themselves (*e.g.* a WSI image of 130K×100K pixels, with ROIs of sizes 30K×20K pixels). In order to focus more closely on select areas within these ROIs, and to keep the burden of computational complexity as small as possible, the ROIs are further partitioned into image chips for individual processing. While feasible for handling a single or a few WSI images, the approach of having experts annotate ROIs from scratch for each image in a large WSI dataset is extremely time-consuming. To this end, the approach implemented in [120]; ie. keeping track of the expert's viewport change history via viewport logs, learning from this history and attempting to predict similar ROIs in subsequent images by utilizing zoom level, navigation displacement and viewing duration information does indeed serve the need for increased efficiency, is part of the future work being implemented in KOLAM.

In the following steps, KOLAM is used first to assign new tissue labels for some of the segments generated by the applied segmentation algorithm. Subsequently, KOLAM is

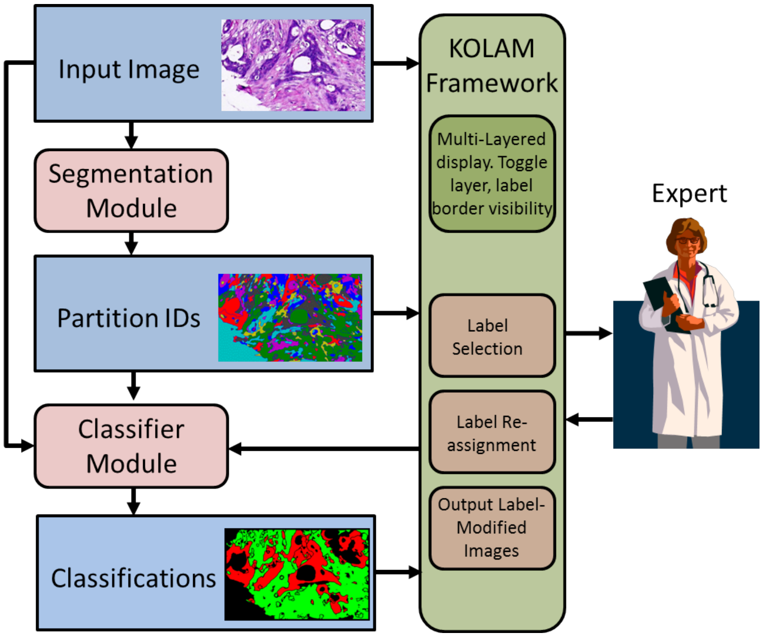


Figure 5.5: KOLAM's role in the training and testing steps of the classifier algorithm. The input image and corresponding partition IDs are provided to the classifier module, which the outputs new tissue microenvironment labels. The expert re-labels incorrectly labeled classes. These corrections are input to the classifier module for training.

used to relabel some of the tissue classes assigned by the classifier algorithm. The details of the training and testing steps are presented below.

5.4.5 Ground-Truth Creation and Relabeling

We now re-examine those stages of the multi-stage pipeline for epithelia-stroma classification (pre-processing, segmentation, feature extraction and classification) where KOLAM has the role in facilitating the creation of ground truth labels as well as their subsequent relabeling (as required). During the *pre-processing step*, image enhancement (morphological reconstruction, normalization, bilateral filtering etc.) operations are performed and pixel features are computed [73]. The computed pixel features include color transforms, texture features within a small region surrounding each pixel, including local binary patterns (LBP), first and second order derivatives, and derived features such as shape index and normalized curvature index [73]. During the *segmentation step*, the **hierarchical fuzzy c-means with spatial constraint (HFCS)** segmentation module incorporates spatial information into the objective function of the classical fuzzy c-means algorithm by including terms for spatial correlation and multi-resolution [73]:

$$J_{SCM}(U,V) = \sum_{i=1}^{C}\sum_{j=1}^{N} u_{ij}^{m} \| x_j - v_i \|^2 + \\ \frac{n}{2}\alpha \sum_{i=1}^{C}\sum_{j=1}^{N} u_{ij}^{m} e^{-\sum_{k\in\Omega} u_{ik}^{m}} + \beta \sum_{i=1}^{C}\sum_{j=1}^{N} u_{ij}^{m} f_i^{(n-1)}(x_j) \tag{5.1}$$

where $X = \{\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_N}\}$ denotes data (pixel feature vectors). $V = \{\mathbf{v_1}, \mathbf{v_2}, ..., \mathbf{v_C}\}$ represents the prototypes (clusters centers). Ω is a set of neighbors ($k \neq j$). $f_i^{(n-1)}(x_j)$ is the point x_j 's ancestor membership function to the i^{th} cluster in lower layer $(n-1)$. Parameters α and β control the influence of the associated terms. α is multiplied by scale factor $(\frac{n}{2})$ to reduce the effect of spatial constraint at lower levels. The HFCM objective function (5.1) contains three terms [73]. The first term is the same as in regular FCM. The second term is a spatial penalty that forces neighboring pixels to belong to the same class. It reaches a minimum when the membership value of neighbors for a particular cluster is large. The third term incorporates the relationship between classes of elements at different resolutions for more feature consistency. Optimization of (5.1) with respect to U is done in a classical way by a Lagrange multiplier technique [73]:

$$J_{SCM}(U,V) = \sum_{j=1}^{N} \lambda_j (1 - \sum_{i=1}^{C} u_{ij}) + \sum_{i=1}^{C} \sum_{j=1}^{N} u_{ij}^{m}$$
$$\left(\| x_j - v_i \|^2 + \frac{n}{2} \alpha e^{-\sum_{k \in \Omega} u_{ik}^{m}} + \beta f_i^{(n-1)}(x_j)\right) \tag{5.2}$$

After taking the derivative of Eq. 5.2 versus u_{ij}, solving for u_{ij}, and solving for λ_j with respect to the constraint eventually leads to the following membership update equation [73]:

$$u_{ij} = \frac{1}{\sum_{p=1}^{C} \left(\frac{\|x_j - v_i\|^2 + \frac{n}{2} \alpha e^{-\sum_{k \in \Omega} u_{ik}^{m}} + \beta f_i^{(n-1)}(x_j)}{\|x_j - v_p\|^2 + \frac{n}{2} \alpha e^{-\sum_{k \in \Omega} u_{pk}^{m}} + \beta f_p^{(n-1)}(x_j)} \right)^{\frac{1}{m-1}}} \tag{5.3}$$

As in (5.3), u_{ij}, the membership value of a point j to cluster i, depends on membership values of its neighbors and ancestor in the pyramidal representation. Regularization is controlled by weights α and β. The prototype update equation is the same as in standard FCM, since the second component of (5.1) does not depend on v_i. Centroids update obeys the equation:

$$v_i = (\sum_{j=1}^{N} u_{ij}^{m} x_j) / (\sum_{j=1}^{N} u_{ij}^{m}) \tag{5.4}$$

The process of segmentation relabeling for classification involves the two stages of training and testing. Designation of ground-truth tissue micro-environment labels is the focus of the training step in the processing pipeline. In Figure 5.5, the training step includes every module except the classifier module and its input, which is solely part of the subsequent testing step. Prior to tissue label reassignment by the expert, the image chips need to be segmented and assigned preliminary labels. Using the HFCM segmentation scheme described above, initial partitions of the input images were generated. For the first training subimage, the expert sees the partition boundaries in a single class color that is overlaid on top of the underlying image. The pathologist can then change the color of any partition from epithelia to stroma or vice versa with just a few user interaction events. Tha manual assignment of ground-truth labels continues until the pathologist is satisfied with the labels assigned to partitions without having to tediously draw any boundary contours in the image. The partitions are essentially an unlabeled segmentation.

These corrected tissue labels are then passed from KOLAM to the image region classifier module that can be applied to other test (sub)images with segmentations. Examples of classifier algorithms that are typically applied to histopathology WSI problems include *k*-nearest neighbor classifiers, support vector machines (SVM), random forest classifiers, multiple instance learning and classifier ensembles. The classifier module used in conjunction with KOLAM utilizes a two-class support vector machine (SVM) classifier (Figure 5.6). Prior to the testing step, the classifier module which received ground truth classes from KOLAM during the training step is trained on these classes in order to produce tissue labels for images outside of the original training image set. For each partition in the training set, class labels are extracted from corresponding expert labeled images. For reliable training, small partitions and partitions with mixed classes are removed from the training set. The complementary set of features used account for color/intensity and texture [73]. They can be grouped as: (1) color features, (2) first-order derivatives: gradient magnitude and gradient orientation, (3) second-order derivatives: from Hessian matrix-derived features such as Laplacian, eigen values, and eigen vectors; and (4) other texture features such as the local binary pattern (LBP) operator.

During the active learning testing phase (see Figure 5.5) the classifier generated tissue micro-environment labels on new (sub)images are used as input to the relabeling system.

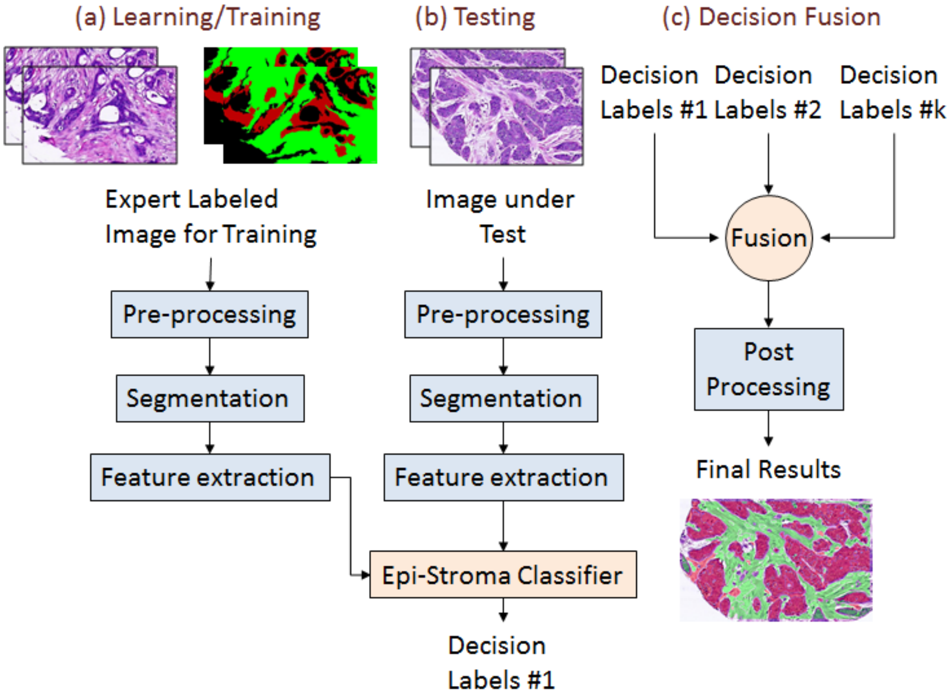


Figure 5.6: Stroma versus Epithelium identification pipeline.

The expert visualizes these modified results, and may correct the annotations (generate ground truth) as necessary. Figure 5.7 illustrates an example of reassignment of a label or label annotation correction during the testing step. This shows the benefits of various KOLAM interface components supporting ground-truth region label annotations. Thus armed with user-corrections to the classes it has already characterized the classifier now has additional verified input that it can use to further refine its next classification step.

The interactive segmentation and relabeling pipeline is described in listing **Algorithm 23**, with the listings for component **Algorithms 24 and 25** describing the label color changing and the image/partitions/labels/borders loading processes, respectively. Each image to be relabeled is comprised of a 3-tuple: the input image data, the class labels and the classifier output. The output of the pipeline is the set of corrected labels to be passed back to the classifier for re-training.

5.5 Conclusion

Pathologists have a strong and growing need for tools to lessen the burden involved in analyzing and annotating large numbers of whole-slide histopathology imagery for training CMSS/CAD systems, and for sharing results among experts using a web-based interface. Algorithm designers require a means to interactively visualize the results of their segmentation algorithms on the WSI data, a means to generate ground-truth for training classifiers and improving and fine-tuning the performance of other algorithmic modules. In addressing these problems and the inherent challenges, the author has devised the solutions and implemented them within the KOLAM system toward this thesis. Now, KOLAM provides capabilities to meet both computational and clinical needs, as well as to bridge the gap between pathologists and algorithm developers. Furthermore, KOLAM now provides a virtual microscope interface supporting a variety of WSI formats along with a novel method for rapidly labeling and correcting the tissue microenvironment stroma-epithelia region labels. To summarize, due to the author's unique contributions which have been described in this chapter, KOLAM now provides vital visanalysis software tools for medical and computational experts to develop the evolving field of *computational histopathology*.

Algorithm 23: Segmentation_Region_Relabeling Algorithm

```
while Relabeling File Mgmt UI Loop is Active do
    if User cancels without loading files or a project then
        break;
    end
    if User selects files or project for loading then
        Load image data;
        Obtain
    end
end
while Relabeling UI Loop is Active do
    if User Clears Dataset then
        break;
    end
    if User hits the ‘Escape’ key then
        break;
    end
    if User Loads Dataset then
        Load_Image_Partitions_Labels_Borders();
        while Data not Cleared do
            if User Clears Dataset then
                break;
            end
            if User Clicks on Image then
                Change_Label_Color();
            end
            if User Toggles All Borders then
                ToggleDraw_All_Borders();
                Save changed label image to disk;
            end
            if User Toggles Current Border then
                ToggleDraw_Current_Border();
                Save changed label image to disk;
            end
        end
    end
end
Return control to KOLAM’s main UI loop;
```

Algorithm 24: Change_Label_Color

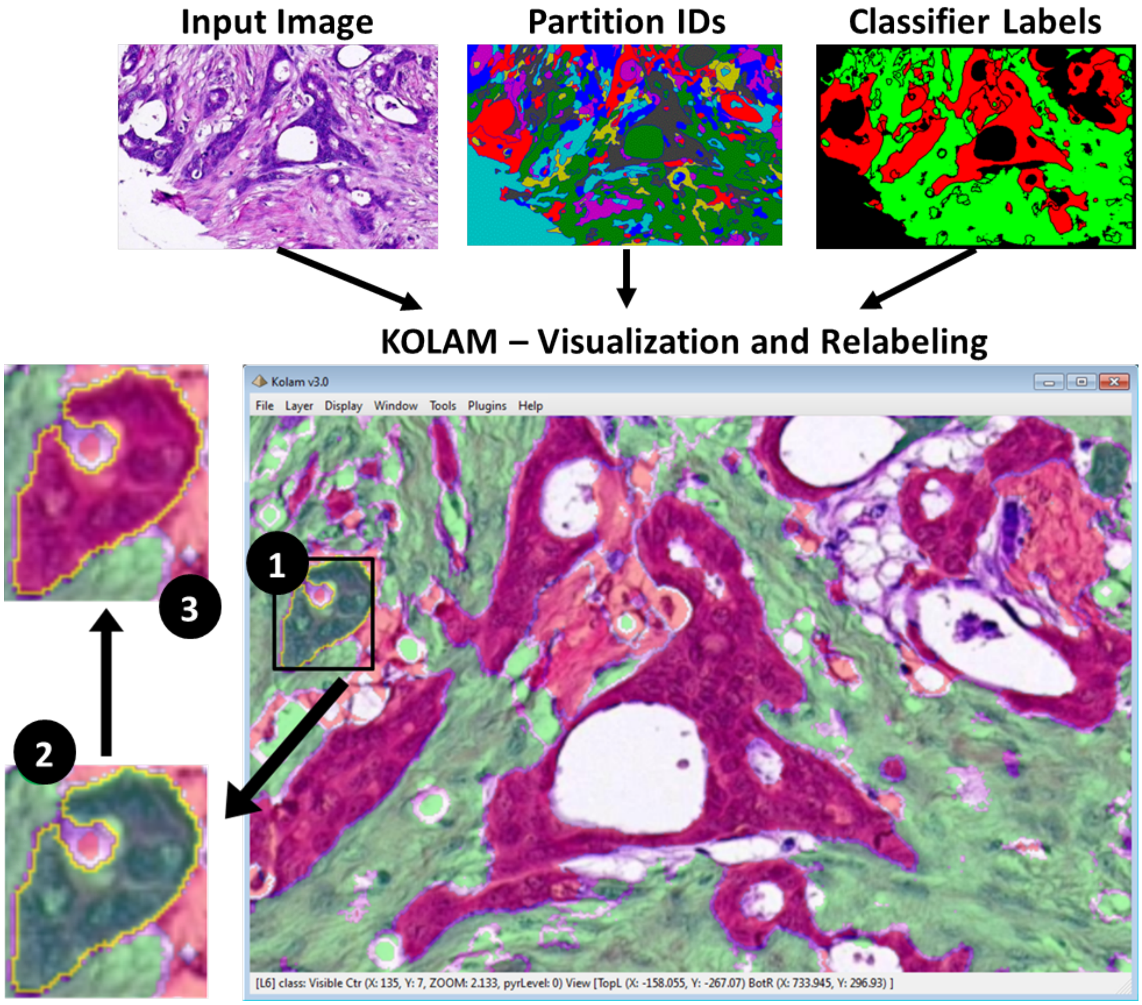
input : PartitionID table with IDs, classes, borders, labels

output: Label of user-selected class updated. Borders updated. Updated label image saved to disk.

```
// Update color of user-clicked label
pixel $(x, y)$ ← scrnToImg(UserClick $_{x,y}$);
ID $_{x,y}$ ← getID(pixel $(x, y)$);
color $_{x,y}$ ← segClassTbl [ID $_{x,y}$].color;
segClassTbl [ID $_{x,y}$].color ← getNextColor(color $_{x,y}$);
// Refresh and update borders if enabled
if drawAllBorders then
    for $i$ ← 0 to segClassTbl.$size() - 1$ do
        segClassTbl [$i$].border.draw();
    end
end
if drawCurrentBorder then
    segClassTbl [ID$_{x,y}$].border.draw();
end
// Save updated label image
$I_{\text{segClassTbl}}$.save();
```

Algorithm 25: Load_Image_Partitions_Labels_Borders

```
input  : Image $I[w \times h]$ with partitions and classifier labels
output: PartitionID table populated with IDs and associated classes, borders and
        labels
Clear any previously loaded data;
Initialize Segmentation Class list;
Load image $I$;
// Build Segm Class table from Partition IDs
for $i \leftarrow 0$ to $w - 1$ do
    for $j \leftarrow 0$ to $h - 1$ do
        ID $_{i,j}$ $\leftarrow$ getID(pixel $(i, j)$);
        segClassTbl [ID $_{i,j}$].append(pixel $(i, j)$);
    end
end
// Associate borders with labels
size $\leftarrow$ segClassTbl [borderIdx].class.size;
for $i \leftarrow 0$ to size $- 1$ do
    border_pixel $\leftarrow$ segClassTbl [borderIdx].class[$i$];
    for $j \leftarrow 0$ to $maxNeighbors(border\_pixel)$ do
        nb $\leftarrow$ neighbor(border_pixel);
        if $nb[j] \notin$ segClassTbl$[borderIdx].class$ then
            neighIdx $\leftarrow$ getID(*neighbor*$(border\_pixel)[j]$);
            thisIdx $\leftarrow$ segClassTbl [neighIdx];
            thisIdx.border.append(border_pixel);
        end
    end
end
Return to KOLAM's main UI Loop;
```


Input Image
Partition IDs
Classifier Labels
KOLAM – Visualization and Relabeling
Kolam v3.0
File Layer Display Window Tools Plugins Help
1
2
3

[L6] class: Visible Ctr (X: 135, Y: 7, ZOOM: 2.133, pyrLevel: 0) View [TopL (X: -158.055, Y: -267.07) BotR (X: 733.945, Y: 296.93)]

Figure 5.7: An example of interactive relabeling of classifier-assigned tissue labels within the KOLAM visualization framework. The inputs to KOLAM for the testing step are shown. The labels for the stroma and epithelium, colored green and red respectively, are overlaid with partial transparency on the underlying imagery which is an image subregion from a breast cancer WSI. The tissue label in the inset box at the left, labeled ’1’, is being relabeled by the user. The label is zoomed up (Inset 2) on the far left, which clearly displays the boundary of the label. Finally, with one click, the expert changes the label, from stroma (Inset 2) to epithelia (Inset 3) class label.

Chapter 6

Extension to Tiled Wall Displays

6.1 Motivation

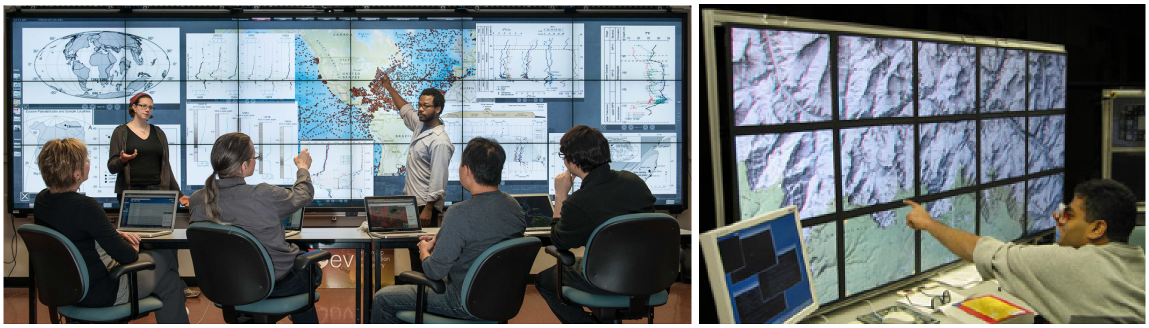
The demand for contiguous, high resolution display capability of the order of several hundred megapixels to gigapixels has been increasing over the past several years. The driving forces for this trend include the huge dimensions of image datasets needing to be analyzed, the decreasing costs of LCD (and competing technology) displays over time, and the collaborative computing and visual presentation needs of entities such as corporations, governments and research labs.

This chapter describes KOLAM's ability to utilize a high-performance cluster to display huge datasets across a tiled, multi-monitor 'Wall' display system. This feature of KOLAM makes it extremely useful and available for deployment to collaborators with interactive visualization and analysis requirements, needing to be demonstrated on multi-monitor wall display environments. An example of this was KOLAM's usage on a multi-monitor Wall display system at the Air Force Research Lab (AFRL) in Rome, NY. The single display / monitor paradigm needed to be extended to allow seamless image data display and interaction across monitor arrays of any available arrangements. The largest such environment comprised a system of 40 wall-mounted monitors, in an array configuration having 8 monitors along the length of the aggregate display area and 5 monitors along the height (ie. an 8 x 5 array).

6.2 Related Work

Gigapixel-sized imagery have become much more prevalent over the past decade in a variety of domains including medicine, space, satellite and defense. Consequently, the development of tools for gigapixel-sized image visualization has been an active area of research. A review of the most prominent literature allows for the rough classification of gigapixel visualization into single display-oriented and multiple display-oriented systems. As an important example of a single display-based system, Kopf et al. created a system for creating, processing and displaying ultra-high resolution images with high dynamic range and wide angle fields of view [11]. Means for systematically annotating and rendering both auditory and visual annotations in gigapixel imagery have also been devised [12]. Additionally, work has been done towards rapid post-processing of gigapixel imagery utilizing an efficient parallel programming methodology [13].

Based on the ability of the human visual system to rapidly identify patterns and discern differences, especially when the targeted data is extremely large, extensive research has been performed regarding the visualization of high-resolution image planes on tiled displays [14–24]. Architectures such as SAGE [16,20,25] (see Figure 6.1) and DIGI-Vis [24];



(a) (b)

Figure 6.1: Multi-display visualization system examples: (a) SAGE and (b) JuxtaView.

and applications such as JuxtaView [17] (see Figure 6.1) and Giga-Stack [23] seamlessly display a number of visualization applications over the entire tiled display.

6.3 Challenges and Solutions

When taking a multi-monitor "Wall" system into account, the fundamental differences between such a setup and a single display system include modification of display coordinate systems, changes to user interactions and associated event handling, and communication issues arising from ensuring that all content being displayed on all the physical monitors is accurately synchronized. Rendering performed by the different GPUs in the cluster (or different computers on a subnet) driving the overall display is done to each local display coordinate system by default. However, the display wall needs to present a single display coordinate system, spanning all rendering devices and associated displays, to the user. This is implemented in KOLAM as a virtual coordinate system that encompasses the individual display coordinate systems, which hides the transformation updates needing to be performed for each rendering device from the user. Tied closely to this are the modifications to the user interaction system, giving the user the feel of interacting with a single display. Finally, updates needed by the different local coordinate systems for display and interaction are transmitted to the different computers/cluster nodes, and are synchronized so that display and interaction feel no different to a user than using a single monitor system. KOLAM's broadcaster-receiver system works together with the changes to coordinate system handling to tackle these issues.

Multiple instances of KOLAM are instantiated on each of the servers driving the wall display. All instances are initialized as receivers, each only capable of receiving messages. Following this, one instance is made the broadcaster, thereby putting it solely in charge of sending messages to the receivers. The message contents may be either display refresh directives or coordinate system transformation information. The user may re-designate the role of broadcaster to a different receiver by interacting with a different display than the one the current broadcaster actively receives user input from. Consequently, the broadcaster and the selected receiver 'switch' roles. The interaction events generated by the user on the broadcaster instance are transmitted to the receivers along with the coordinate system information. Event queuing and delayed processing facilities provided by the Qt API are utilized in conjunction with broadcaster update frequency handling in KOLAM to provide the user a smooth interaction experience over the wall display. Event queuing and delayed event processing also takes care of display 'freezing' and 'swimming' issues that arise due to event overflow and de-synchronization.

Unlike the single monitor display mode, this mode of operation involves *multiple command-line invocations* of KOLAM (one to each node of the cluster), each of which takes *multiple parameters* that designate what to display on each tile (ie. monitor) of the multi-monitor system. **It is highly recommended** that the command-line instructions be saved in the form of *script files*, in order **to save initial setup time** and to **enable off-line editing** of the display parameters. Furthermore, it is imperative that the nodes of the cluster be on the same network subnet; the wall display mode will not function otherwise.

Additional **networking parameters** affecting the performance of the Wall system may be modified via the section of the Kolam-Preferences tool. An illustration of KOLAM displaying multiple layers at different zoom levels on a multi-monitor display is shown in Figure 6.2.

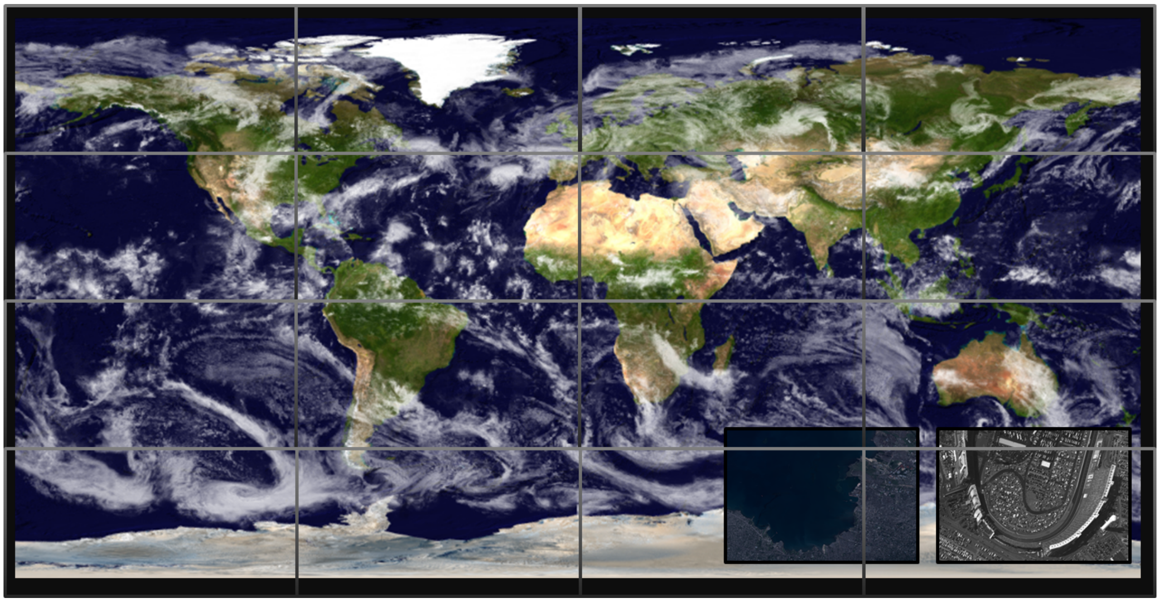


Figure 6.2: An illustration of KOLAM being used on a multi-monitor display.

6.3.1 Example Setup Scenario

Consider that the available multi-monitor setup is a **2x2 grid**, with each of the four monitors having a resolution of 2560 x 1600. Let the KOLAM instance that will be designated as the broadcaster display to the *lower-left* monitor. Three other computers on the same subnet, connected to the other three monitors on the grid, are designated as receivers (since they receive update events generated by the user from the broadcaster instance). The command-line instructions to load a PSS sequence of image files, *in order*, on the **lower-left**, **lower-right**, **upper-left** and **upper-right** monitors (these must be issued on the respective computers) are as follows:

```
kolam -geometry 2560x1600 ’full_path’/frame_#####_0.pss -x 2560 -y 0
kolam -geometry 2560x1600 ’full_path’/frame_#####_0.pss -x 0 -y 0
kolam -geometry 2560x1600 ’full_path’/frame_#####_0.pss -x 2560 -y 1600
kolam -geometry 2560x1600 ’full_path’/frame_#####_0.pss -x 0 -y 1600
```

6.3.2 Current System Limitations

User events are processed and propagated ONLY IF they occur within the monitor(s) to which the broadcaster instance of KOLAM is displaying. Events that occur on any receiver instance are ignored in favor of display synchronization updates that are dispatched from the broadcaster. Making another running instance of KOLAM a broadcaster instance is currently a manual process: the current broadcaster instance must have its broadcaster status removed and must be designated a receiver instance, followed by designating the instance of interest (currently a receiver) as the next broadcaster. The restriction of all KOLAM instances to the same subnet is yet another limitation of the system.

Chapter 7

Evaluation of KOLAM

7.1 Introduction

In this chapter, we present the evaluation methodologies, metrics, experiments that have been performed, as well as the results obtained, in order to evaluate the KOLAM software system. Both quantitative and qualitative evaluations have been performed. In the following sections, we first take a look at the evolution of the KOLAM system over time, looking at the addition of key features already described in previous chapters and the decisions and needs that led to them, and how these features were modified or replaced based on the feedback provided by expert domain users who used the software system for their specific needs during this time. Following this, we describe details of Usability Testing and provide a framework for the testing procedures that KOLAM was subjected to. The final sections describe the experimental setup, testing data, testing procedures, results and conclusions.

7.2 Evolution of KOLAM based on Iterative Expert Feedback

The involvement of experts over time in evaluating the newly added as well as the existing features of KOLAM can be considered a series of stakeholder walkthroughs; wherein end users, stakeholders and the designer collaborated to evaluate the earlier versions of KOLAM, generating recommendations for improvements and changes, as well as building empathy [121].

The author's initial contributions to KOLAM were made during August of 2009, when he created the initial design and implementation of the animation caching system and the animation playback interface (Kolam-Loop). Both initial implementations were sub-optimal, and were enhanced and thus evolved into the first stable version of the present day spatio-temporal caching and animation interface; with expert advice and feedback from the faculty adviser, the previous developer of KOLAM and other research group members. The first version of KOLAM's overlay system for annotation display and interaction was also developed at this time.

Around October of 2009, the initial version of the trajectory drawing module was developed. This system could process and render trajectory data solely from sets of flat files corresponding to each object trajectory (One file per time step, per trajectory). Furthermore, the drawing performance of the trajectory system was sub-par at this time, both for greater number of trajectories, as well as individually long trajectories. Concurrent with the development of the trajectory visualization system, the author implemented the first version of the Kolam-Tracker tool, which provided (a) the UI interface to the loose coupling between KOLAM and the first version of the CSURF object tracker, as well as (b) trajectory visualization management, which was also designed and implemented during this time.

The end of 2009 was spent iteratively refining the different UI aspects of the Kolam-Tracker tool with expert feedback from the faculty adviser and research group members following usage and testing, and addition of support for the first version of the LOFT tracker. The LOFT and SURF tracker programs require different number of input initializations to begin execution, and KOLAM's tracker input annotation component was modified to transparently handle both cases. Expert feedback regarding features was also received from a visiting domain expert collaborator from the Air Force Research Lab (AFRL). Following this, the author implemented a change to the trajectory visualization data structure which resulted in trajectory visualization efficiency improving by a polynomial factor.

During the Spring of 2010, a number of KOLAM's current features were created, modified and improved. Based on expert feedback from the faculty adviser, the menu systems were revamped to reduce the mouse travel time needed to perform related tasks. Tools such as the Kolam-Layer Editor were significantly updated during this period: for the first time, loaded layers could be interactively moved up or down the layer display stack, with the contents of the main display area being updated immediately. Interactive layer visibility toggling was also added to the Layer Editor during this period. Research and implementation ideas by the author led to several newly implemented components of KOLAM coming together: animation playback in layers, dynamically changing layer positions in the layer stack with immediate visual update, annotation overlays, trajectory drawing and management, per-layer vs whole-view transformation: to create one of the novel contributions: Coordinated visualization using stacked layers, side-by-side static/time-lapse image data layers with either annotations or trajectories. Partial inspiration for this idea was due to feature feedback provided by a domain expert collaborator from the Harvard Medical School visiting during this time. Finally, KOLAM's feature set was utilized in the conception of the first research paper on the LOFT object tracking system [45].

The Summer of 2010 saw several features added to KOLAM during the author's internship at the Army Research Lab (ARL) campus in Adelphi, MD. The features implemented during this time reflected both the object tracking and trajectory visualization needs of ARL, as well as the valuable expert feedback received from the highly experienced ARL analysts pertaining to relevant features for implementation and avoidable stumbling blocks they encountered in several other tools. The main features added during this time included the context-sensitive trajectory rewind, the Kolam-Locate tool displaying the Latitude, Longitude, image x-and y-coordinates of the user-clicked point in the data (with data navigation-driven interactive updates), the trajectory archival system for storing and sharing trajectory data across systems, and support for Octave executables in addition to Matlab. UI enhancements during this time (also driven by domain-expert user feedback) included improved animation scrubbing for image sequences, mouse wheel-based zomming for the data, and the addition of an online Help and imagery data browser.

The Fall of 2010 saw enhancements to KOLAM's trajectory drawing system. Based on expert user feedback, additional drawing primitives - bounding boxes and polygons - were added to supplement centroids as the primitive of choice for each time step along a trajectory. KOLAM started receiving interest from collaborators in cell biology during this time, and additions were made to trajectory drawing and visualization based on their requirements and feedback. 2011 saw greater interaction with collaborators from Cell Biology and incorporation of KOLAM with a harness system to test SURF and LOFT tracker performance [40].

Interaction between our group and Kitware Inc. began in Spring of 2012, with Kitware expressing interest in using KOLAM for ground-truthing with their in-house trajectory data format, KW18. Designing and implementing support for this semi-structured data storage format in KOLAM offered significant performance and storage benefits vis-a-vis the older flat text file system. KOLAM's trajectory input and data structure underwent changes to accommodate the KW18 format. The benefits were immediate: multiple trajectories could be stored in a single KW18, also allowing for a higher level of trajectory data organization involving data from multiple KW18s. The Summer of 2012 saw further work involving Kitware. Also, the AFRL's need at this time saw the development in KOLAM of the multi-monitor display Wall system. This work would carry on into the Fall of 2012.

The Spring of 2013 saw development begin on KOLAM's histopathology segmentation system. The author was part of a team from our research group who visited the pathologist expert at the University of Missouri Medical School. The expert was able to test the portion of the segmentation relabeling system that had already been completed, gave positive reviews of the implemented features, and provided valuable feedback regarding further feature addition and existing feature modification or enhancement. This resulted in a second visit during the Summer of 2013, when the process was repeated for another iteration of development, and acquisition of new data for segmentation and classification analysis alongwith KOLAM relabeling was discussed. Work on the interactive segmentation relabeling system continued through the Fall of 2013. The year 2014 was spent designing and implementing the initial form of KOLAM's trajectory editing system, followed by the iterative enhancements and modifications which resulted in its current form. The current feature set for the trajectory editing system was derived through extensive user feedback.

The remainder of the period between then and the current time was spent either enhancing or modifying existing features in KOLAM, based on expert feedback from various domain expert users. Notable activities during this time include addition of support of extra imagery data formats, in order to increase the target audience who could use KOLAM to visualize their data. Finally, optimizations and bug fixes were made throughout the KOLAM system in reponse to user feedback regarding prolonged usage scenarios for KOLAM. The following section outlines and provides some details regarding Usability testing.

7.3 Usability Testing

Usability testing is an evaluative method that allows for the observation of an individual's experience with a software system, while he/she performs the steps of a specified task (or task set). Per Gould et al. [122], usability testing focuses on people and their tasks, and seeks empirical evidence about how to improve the usability of an interface. The tests are designed based on tasks and scenarios that represent the end-user goals involving the software system and needs in question. The *tasks* should reflect the actual goals of the target users, and should be concrete and specific. The *scenarios* provide task context and thereby aid in task completion by providing additional information. Tasks and scenarios should be designed such that they do not influence testers to solve problems in a certain manner, nor seek to justify the software system requirements. Errors that need to be detected [123] include instances during which the tester: (1) *cannot complete the task* in a reasonable time period despite understanding it, (2) needs to *try different approaches* to complete the task despite understanding the goal, (3) *gives up* doing the task, (4) *completes a different task* from what was specified, (5) expresses *surprise*, (6) expresses *frustration or confusion*, (7) states that *something is wrong* or illogical, or (8) *suggests alternative(s)* for event flow or interface structure.

7.4 Surveys

Surveys enable collection of self-reported information from testers regarding their thoughts, feelings, perceptions, behaviors and attitudes as pertain to the task being performed and the software tool being used. They are an efficient method for collecting a large amount of data in a short time frame, typically from a large sample of testers. Survey data collection can be done either using questionnaires or structured interviews. The types of questions on the questionnaire are chosen to be (1) Closed, (2) Specific, (3) Factual, (4) Judgmental, (5) Comparative, and (6) Neutral.

7.5 Evaluation Methodology

As mentioned previously, KOLAM is part of the overall M-Path (MU Pathologist) system, a multi-stage pipeline which integrates computer vision and machine learning algorithms with a scalable big data visualization and analysis interface, thereby aiming to extend the duration of the patient survivability prediction performed by the C-Path system [64]. The portion of the pipeline that requires user intervention, include relabeling of the segmentation results and correcting classifier assigned labels. Furthermore, KOLAM's trajectory editing system is an individual component by itself, but also forms a part of KOLAM's overall ability to perform human expert assisted semi-automatic tracking of objects of interest in WAMI and relevant biomedical imagery datasets. These tasks need to be implemented in a manner that maximizes operational efficiency while retaining user attention and minimizing user fatigue. We now describe the evaluation methodology we have designed, and tabulate the results of experiments we have performed.

In order to effectively evaluate the different performance aspects of KOLAM, the following criteria need to be considered:

1. Accuracy: The role played by various features of the software system being evaluated in influencing the accuracy of the user in performing 1-Click Object Tracking of objects of interest, the interactive relabeling of segmentation results, or performing multiple ‘Move’ corrective operations on several points along an object’s trajectory;

2. Efficiency: How do the features provided by the software system being evaluated affect user efficiency? Is the increase in efficiency quantifiable, and if so, what metric is suitable to do the same?

3. Repeatability: For a given software system, can the user reproduce the the same set of object tracking results, relabeling corrections or trajectory editing operations with ease and accuracy?

We note that these criteria are highly interdependent: for example, user results are typically more accurate when users are given greater amount of time per task. Providing more time to perform the corrections may also reduce the possible number of user errors in the object tracking, relabeling or trajectory editing processes, thereby raising efficiency and repeatability. Repeatability (or reproducibility) describes the certainty by which different people under varying laboratory conditions can reproduce all the procedures and methodologies from start to finish, thereby describing the accuracy and reliability of a test method [124]. These criteria closely match the criteria outlined in the ISO/IEC standards for usability metrics, which are outlined below.

7.5.1 Usability Metrics in the ISO/IEC

Usability is defined as “the extent to which a product can be used by specified users to achieve specified goals with **effectiveness**, **efficiency** and **satisfaction** in a specified context of use; per the ISO 9241-11 standard [125]. The ISO/IEC 9126-4 metrics [126] recommend the same, stating that usability metrics should include:

1. **Effectiveness**: The accuracy and completeness with which users achieve specified goals (in this case, the accuracy of the user in performing either the trajectory editing or interactive relabeling of segmentation labels operations);

2. **Efficiency**: When the users achieve the specified goals, quantify the resources expended in relation to their accuracy and completeness.

3. **Satisfaction**: In achieving the specified goals, the comfort and acceptability of (using KOLAM by) the users.

The experiments that were performed using KOLAM aimed to obtain quantitative (effectiveness and efficiency) and qualitative (satisfaction) measures of these usability metrics.

7.6 Experimental Procedure

In this section, we detail the various aspects of the experiments that were performed towards KOLAM’s Usability testing. This includes information about experimental goals, participant involvement, a listing of the software systems and ground truth datasets, and the rules utilized for guiding the experiments. Criteria such as accuracy, reproducibility and efficiency were taken into account.

As errors in the ground truth have a direct impact on accuracy, any ground truth dataset used must be as accurate as possible, in order to measure accuracy effectively. The effective measurement of efficiency is primarily dependent on recording all the intermediate corrections that the users make in tracking objects, or editing the object trajectories, or relabeling the segmentation results; over time. It is important to note the relationship and tradeoff between accuracy and time: considering segmentation relabeling for example, users will have to prioritize what they relabel based on ’how wrong’ the individual segments they relabel are. Given that the purpose of the interactive relabeling facility in KOLAM and other software systems is to drastically reduce the amount of time in obtaining a correct segmentation of a histopathology image versus a purely manual labeling approach, users will need to correct the most inaccurate (based on object and / or boundary dissimilarity) segments prior to lesser inaccurate ones. A sufficient number of users / participants is needed in order to effectively measure reproducibility.

7.6.1 Goals

Currently used automated object detection algorithms [46] [127] do not perfectly produce object detections in the context of the relevant (WAMI) data. Obtaining correct object detections (viz trajectories) given the additional intervention of human experts implies that the time spent on trajectory editing operations is an important component of the overall time spent on the object detection task. The goals of the 1-Click Object Tracking and Trajectory Editing experiments are two-fold: (1) Assess KOLAM’s editing efficiency and effectiveness quantitatively, and (2) assess the satisfaction of using KOLAM qualitatively.

Beck et al. [64] developed the C-Path system with the goal of designing a data-driven approach without bias to discover prognostically significant morphological features in breast cancer imagery. This system is characterized by: (i) Comparatively small imagery ($<$ 10 MPixels each) and (ii) A large number of features (6642) per image. The epithelia-stroma classifier we are developing uses 700 features instead of 6642, while attaining the same predictive accuracy as the C-Path system (due to the greater predictive power per feature in our system). The goal of using the KOLAM system is to reduce the amount of time involved in the corrective relabeling process. It accomplishes this by (i) working in a tightly coupled fashion with the hierarchical segmentation and epithelia-stroma classifier modules and seamlessly exchanging information back and forth, and (ii) providing an interface in which each corrective step can be performed with minimum user interaction while maximizing the total corrected region and providing the best possible visual feedback for the same. KOLAM's whole-slide imagery visualization system, coupled together with its ROI extraction, sub-region partitioning interfaces make available to the segmentation and classification systems all the information they need to extract and generate powerful higher level features which could facilitate more robust predictive model performance for the classifier. The goals of the segmentation relabeling experiment are two-fold: (1) Assess KOLAM's relabeling efficiency and effectiveness quantitatively, and (2) assess the satisfaction of using KOLAM qualitatively.

7.6.2 Software

Users were able to test 1-click Object Tracking, Segmentation Relabeling and Trajectory Editing in KOLAM alone, due to time constraints and software and infrastructure deficiencies. The initial testing plan prior to these constraints is provided below for completeness.

Besides KOLAM, users would perform interactive relabeling of segmentation results by utilizing the iLastik, ePathology ImageScope and paint.Net software packages. Likewise, users were to perform trajectory editing in software that supported the feature, such as ViPER. 1-Click Object Tracking is a unique feature of KOLAM that does not exist in any other software system in the world, to the author's knowledge. Due to the disparity between interfaces, the users were to be given time sufficient to familiarize themselves with operating the different software features needed to perform the interactive segmentation relabeling task. The main set of experiments were not to be initiated until all users had achieved a similar level of comfort and expertise with the different software systems, thus none of the software were penalized due to factors such as past user familiarity, user preference based on subjective factors etc.

KOLAM

1-Click Object Tracking in KOLAM is performed by (i) Identifying the object of interest to track, (ii) Panning and/or zooming the image data until the object is in the user's field of view such that the user is comfortable with viewing and selecting the object for tracking, (iii) Annotate the object to track with a bounding box: the user is free to redraw the BBox until it's correct, (iv) Signal KOLAM that object selection is complete, wherein KOLAM invokes the external CSURF tracker to track the object, (v) Observe the output of the CSURF tracker in real time within the KOLAM display: the user can terminate the CSURF tracker if its output becomes erroneous, continue observing the object being tracked until the tracker is done, or begin the simultaneous tracking of the next object of interest.

Trajectory Editing in KOLAM is performed by (i) Selecting the trajectories of interest, (ii) Selecting the relevant trajectory from a drop-down, (iii) Selecting the desired editing operation (MOVE; in the case of the experiment), and (iv) Applying the operation to the desired points on the trajectory via right mouse click (and drag, in the case of MOVE). If the same editing operation is to be performed for multiple trajectories, Step (iii) needs to be performed only the first time. If users desire to visualize the entire length of the trajectory regardless of the current time step, they can do so via the ‘Whole Trajectory (Display)’ button. Furthermore, if users wish to avoid visual clutter from unselected trajectories as well as from overlapping selected trajectories, they may opt to select relevant trajectories one at a time and toggling the visibility of un-selected trajectories.

Segmentation relabeling in KOLAM is performed via a single right mouse click. Repeatedly clicking the right mouse button cycles through all used labels (currently, stroma, epithelia and undecided). The different labels are visually represented by transparent masks; each of which can be colored red, green or colorless. Every time the user performs the above relabeling interaction, the updated label is internally maintained, and one or more labels can be saved to an updated image label mask at the user’s discretion. The default mode for the same is to save the updated label mask every time the user makes a correction.

7.6.3 Data

For the purposes of the **1-Click Object Tracking** experiment, a WAMI image sequence from the Philadelphia, PA dataset (image sequence, each multi-resolution tiled image of dimensions 16384 pixels × 16384 pixels) collected by Ross McNutt (Persistent Surveillance Systems) on March 13th, 2008 was used as input imagery. The object tracker used for tracking the objects of interest was the CSURF tracker, developed in our research group.

For the purposes of the **Trajectory Editing** experiment, an ROI image sequence from the Edgewood, NM Walmart dataset (image sequence, each multi-resolution tiled image of dimensions 6600 pixels × 4400 pixels) collected by Dr. Steve Suddarth (Transparent Sky) on June 8th, 2017 was used as input imagery. The trajectories that were used for editing by the users were generated by a multi-target tracking algorithm developed in our research group.

For the purposes of the **Segmantation Relabeling** experiment, the input images were chosen from the Stanford Breast Cancer Microarray dataset. The segmentations and classifier labels used for relabeling by the users were generated by the segmentation and classification modules described in Section 5.4.

7.6.4 Tasks

For the purposes of the **1-Click Object Tracking** experiment, each user task comprised performing the 1-Click Object Tracking operation on 10 random objects of interest within a given image of the loaded image sequence dataset. Each user performed a total of 5 such tasks, for 5 different images in the loaded image sequence.

For the purposes of the **Trajectory Editing** experiment, each user task comprised performing the 'MOVE' edit operation on 10 random (object detection) points on a trajectory he/she selected. Each user performed a total of 5 such tasks, for 5 different trajectories.

For the purposes of the **Segmentation Relabeling** experiment, each user task comprised performing the relabeling operation on 10 random (classifier generated) labels for an image from the Stanford Breast Cancer microarray dataset. Each user performed a total of 5 such tasks, for 5 different images.

7.6.5 Conducting the Experiment

Based on the methodology described by Rubin et al [128], the different stages involved in conducting the experiment were carried out. These stages are described below.

(1) **Developing the Test Plan**: The test plan served as a test blueprint, defined the required resources, and provided the test milestone(s). It included the objectives of the test, the research questions, participant characteristics, test design, the list of tasks, the test environment and equipment, role of moderator, data to be collected and evaluation measures, and the final report.

(2) **Setting up the Testing Environment**: The testing procedure had no special observational requirements and no need for specialized equipment. KOLAM and data access were the only requirements. The former was installed on the relevant machines and access to the latter was provided via portable storage. The author's lab was chosen as the location for the tests.

(3) **Finding and Selecting Participants**: Obtained test results are valid only if test participants are typical users of image visualization and analysis software. Thus, the behavior, skill set and knowledge of a typical participant (a user profile) were taken into account when considering the prospective candidates for the experiment. The group of test users obtained in this way consisted of 5 graduate students. This number of participants was determined by the number of available candidates as well as time constraints. Of the 5 candidates, one used KOLAM every now and then (but not day-to-day), two rarely used KOLAM and the final two had never used KOLAM before. The participants were scheduled and confirmed via phone and email communications. The participants' privacy was protected in that no personal identifying information was collected from any of them. They are identified as 'User A', 'User B', 'User C', 'User D' and 'User E' in the test result tabulation.

(4) **Preparing the Test Materials**: These included the orientation script, data collection instruments, task scenarios and post-test questionnaire. The orientation / introduction script was read out to participants and described what would happen during the test session, the testing setup, and set that tone in the participants' minds. Finally, it put them at ease by informing them what they would be doing in neutral language, and that only KOLAM was being tested. It was made clear that the participants were not being tested themselves. The orientation script was read to each participant individually. In choosing a data collection method (timing information), the author chose the option of user-generated data collection. This choice was made due to the author's financial and time constraints. Data was collected manually; with participants recording times taken and number of operations in a table, and starting and stopping the timer device themselves. The task scenarios represent actual work that real-world users could perform using KOLAM. They comprised: (1) the result to be achieved by the participant, (2) motives for performing the task, (3) details about the data, (4) state of the system when the task was initiated, and (5) what the participant saw while performing the task. Upon completion of the tasks, the participants were given a post-test questionnaire. This was done to gather participant feedback, in order to gain insight into KOLAM's strengths and weaknesses from multiple user perspectives. The questionnaire used by the author was derived from the *System Usability Scale* (SUS), which is a Likert scale commonly used to test computer software and hardware systems.

(5) **Conducting the Experiment Sessions**: The author greeted the participants and read the orientation script to each participant individually. Following this, each participant was requested to move to the testing area. The author assumed the role of moderator for the sessions with each participant. After clearly communicating to the participants what tasks they were expected to perform, what data to collect and how to collect it, the moderator

stepped away from the vicinity of the participant, until the participant signaled that he/she had completed the assigned activity. Upon completion, the participant was given the post-test questionnaire to fill, following which the author debriefed the participant and closed the session. The author then collected the data observation sheets.

(6) **Participant Debriefing**: Following the post-test questionnaire, the author explored and reviewed the participant's actions during the test. The goal of this was to understand why any errors occurred, as well as to resolve any residual questions. The debriefing was held in the same room as the testing area.

(7) **Analyze data and observations**: Details are presented in the Experimental Outcome section (Section 7.7).

7.7 Experimental Outcome

7.7.1 Raw Data

The raw data gathered from the participant users for the three experiments (1-click object tracking, trajectory editing and segmentation relabeling) are presented in Table 7.1, Table 7.2 and Table 7.3.

The individual scores (1-5) for each participant for the System Usability Scale (SUS) questionnaire, which contained 10 questions, are given in Table 7.4.

7.7.2 Usability Metrics

The **effectiveness** of the application (viz. KOLAM) can be gauged via usability metrics such as *completion rate* (the *fundamental usability metric*, whether a test participant com-

	User A		User B		User C		User D		User E	
Run#	#Objs	T(s)	#Objs	T(s)	#Objs	T(s)	#Objs	T(s)	#Objs	T(s)
1	10	184	10	229	10	192	10	198	10	175
2	10	179	10	209	10	183	10	186	10	185
3	10	140	10	219	10	175	10	178	10	160
4	10	139	10	213	10	187	10	181	10	167
5	10	137	10	229	10	174	10	177	10	173
Avg.		156		220		182		184		172

Table 7.1: User testing data from five runs of the 1-Click Object Tracking operation for 5 users ‘A’, ‘B’, ‘C’, ‘D’ and ‘E’. Average times for each user have also been computed.

	User A		User B		User C		User D		User E	
Run#	#Movs	T(s)	#Movs	T(s)	#Movs	T(s)	#Movs	T(s)	#Movs	T(s)
1	10	21	10	150	10	54	10	52	10	45
2	10	23	10	53	10	37	10	42	10	38
3	10	24	10	95	10	30	10	43	10	40
4	10	21	10	105	10	33	10	47	10	41
5	10	18	10	66	10	29	10	49	10	37
Avg.		21.4		93.8		36.6		46.6		40.2

Table 7.2: User testing data from five runs of the Trajectory Editing operation - MOVE points for 5 users ‘A’, ‘B’, ‘C’, ‘D’ and ‘E’. Average times for each user have also been computed.

pletes a task or not) and *number of errors per task.* Completion rate is calculated as follows:

$$\text{Completion Rate} = \frac{\text{Number of tasks completed successfully}}{\text{Total number of tasks undertaken}} * 100 \qquad (7.1)$$

Application (KOLAM) **efficiency** is measured in terms of the time the participant takes to successfully complete a task. Task time is obtained by subtracting task starting time from the task ending time. Efficiency can be computed either in terms of *Time-based Efficiency* or *Overall Relative Efficiency*. Both are described below.

Time-based Efficiency is computed as follows:

	User A		User B		User C		User D		User E	
Run#	#Clks	T(s)	#Clks	T(s)	#Clks	T(s)	#Clks	T(s)	#Clks	T(s)
1	10	16	10	20	10	20	10	22	10	16
2	10	15	10	4	10	21	10	21	10	15
3	10	15	10	20	10	17	10	19	10	11
4	10	16	10	26	10	16	10	21	10	10
5	10	13	10	17	10	15	10	19	10	10
Avg.		15		17.4		17.8		20.4		12.4

Table 7.3: User testing data from five runs of the Segmentation Relabeling operation - CHANGE labels for 5 users 'A', 'B', 'C', 'D' and 'E'. Average times for each user have also been computed.

Question #	1	2	3	4	5	6	7	8	9	10
User A	5	3	4	4	4	2	4	1	4	1
User B	5	2	4	1	5	1	5	1	5	2
User C	5	1	5	1	5	1	5	1	5	1
User D	5	1	5	1	5	1	5	1	5	2
User E	5	1	5	1	4	2	5	1	5	1

Table 7.4: User satisfaction data from the post-experiment (SUS) questionnaire: 5 users 'A', 'B', 'C', 'D' and 'E'.

$$\text{Time Based Efficiency} = \frac{\sum_{j=1}^{R}\sum_{i=1}^{N}\frac{n_{ij}}{t_{ij}}}{NR} \tag{7.2}$$

where $N =$ The total number of tasks

$R =$ The number of users

$n_{ij} =$ The result of task i by user j :

if completed, then $n_{ij} = 1$, else $n_{ij} = 0$

$t_{ij} =$ Time user j spends to complete task i

The **Overall Relative Efficiency** is defined as the ratio of the time taken by the users who successfully completed the task relative to the total time taken by all the users, and is computed as follows. The same definitions for the terms (as above) apply:

$$\text{Overall Relative Efficiency} = \frac{\sum_{j=1}^{R}\sum_{i=1}^{N} n_{ij}t_{ij}}{\sum_{j=1}^{R}\sum_{i=1}^{N} t_{ij}} * 100\% \tag{7.3}$$

The **System Usability Scale (SUS)** [129] was chosen to qualitatively evaluate user satisfaction with KOLAM as it is a proven method for evaluating system usability against industry standards. Other features of SUS that factored into its choice for KOLAM's Usability Metrics evaluation include (i) its short length and not requiring much in the way of resources to administer and (ii) the availability of an existing SUS template questionnaire which was modified for KOLAM's specifics.

The SUS score was computed from the user selections on their questionnaires using these steps: (1) Subtract 1 from the score for odd-numbered questions, (2) Subtract the scores from 5 for even-numbered questions, and (3) Add the obtained values, and multiply the sum by 2.5. The score obtained is out of 100.

7.7.3 Results

Using Equation 7.1 with successful completion information from Table 7.1 gives KOLAM a **Completion Rate of 100%** for the 1-Click Object Tracking experiment. Additionally, using Equation 7.1 with the successful completion information from Tables 7.2 and 7.3 gives KOLAM a **Completion Rate of 100%** for both the Trajectory Editing and Segmentation Relabeling experiments.

Using Equation 7.2 with the timings from Table 7.1 gives KOLAM a **Time Based Efficiency of 0.003 Tasks/second** for the 1-Click Object Tracking experiment, where each Task consisted of 10 object tracking operations. Furthermore, using Equation 7.2 with the timings from Table 7.2 gives KOLAM a **Time Based Efficiency of 0.029 Tasks/second** for the Trajectory Editing experiment, where each Task consisted of 10 editing (MOVE) operations. Finally, using Equation 7.2 with the timings from Table 7.3 gives KOLAM a **Time Based Efficiency of 0.067 Tasks/second** for the Segmentation Relabeling experiment, where each Task consisted of 10 relabeling operations.

Using Equation 7.3 with the timings from Table 7.1 gives KOLAM an **Overall Relative Efficiency of 100%** for the 1-Click Object Tracking experiment. Furthermore, using Equation 7.3 with the timings from Table 7.2 gives KOLAM an **Overall Relative Efficiency of 100%** for the Trajectory Editing experiment. Finally, using Equation 7.3 with the timings from Table 7.3 gives KOLAM an **Overall Relative Efficiency of 100%** for the Segmentation Relabeling experiment.

Using the score data from Table 7.4, the SUS scores for each user were as follows:

$$\text{User } A = 75, \text{ User } B = 92.5, \text{ User } C = 100, \text{ User } D = 97.5, \text{ User } E = 95$$

This gave a **SUS score of 92** for the overall qualitative evaluation of KOLAM.

7.7.4 Findings

In the **1-Click Object Tracking experiment**, the test participants performed **0.003 object tracking operations/second** on average, per the results in subsection 7.7.3. Based on post-experiment feedback, several factors played a role in determining the obtained measurements. Most importantly, it was observed that of the three experiments, users took the longest time on average to complete individual tasks in this experiment. The primary contributing factor to these long times is that 1-Click Object Tracking is actually a collection of tasks, including: identifying the object to track in the imagery, panning and/or zooming to visually bring the object into the user's field of view, annotating the object and visually validating the correctness of this annotation, passing the object selection information to the external target tracking program and finally validating the tracking result. Each of these sub-tasks takes its own time to perform. User feedback also indicates that of all these sub-tasks, the task of panning and zooming the data to bring the object of interest into the users' field of view took the most time. Again, from user feedback, it was inferred that zooming in to examine objects in greater detail results in loss of global context of the data, whereby it was necessary to zoom out again to regain it, in order to focus on other potential objects of interest. Any approach to increase efficiency for this task must therefore involve retaining the global context of the data in some manner, while examining objects in greater detail (ie. in their local contexts) at different locations.

In the **segmentation relabeling experiment**, the test participants performed **0.67 relabeling operations/second** on average, per the results in subsection 7.7.3. Based on post-experiment feedback, several factors played a role in determining the obtained measurements. Primary among these was the mouse travel time from one segment to another, and the data navigation steps (panning and/or zooming) that were required. Longer mouse travel time and/or higher number of navigation steps directly translated into longer overall time to complete the task of relabeling 10 segmentation labels. The converse is true for shorter overall task completion time. This explains the 'anomalous' time of 4 seconds for the 2nd run for User 'B', who chose to relabel neighboring segments in an alternating manner during this run.

It can be inferred that user navigation of the data display in order to examine regions of interest (possible erroneous labeling), which involves a combination of one or more of panning in different directions and zooming in and out; plays the primary role in time taken to perform a multiple relabeling task (thereby its efficiency). Besides possible erroneous behavior of the software, task completion is affected by incorrect user actions, clarity of the visualization to the user, whether the user is relaxed or in a hurry to complete the task, the users' eyesight etc.

In the **trajectory editing experiment**, the test participants performed **0.29 MOVE operations/second** on average, per the results in subsection 7.7.3. Based on post-experiment feedback, several factors played a role in determining the obtained time measurements. Unlike segmentation relabeling, mouse travel time and data navigation time was not the only influencing factors. Overlapping trajectories and the interaction modes in KOLAM (for trajectory editing) also played a part in some of the recorded times. User 'C' experienced delays due to trajectory overlap, and had to spend extra time using KOLAM's features to display the trajectory of interest in an un-occluded manner prior to performing the MOVE operations on the trajectory points. This contributed to the longer time taken by User 'C' compared to User 'A'. User 'B' also experienced delays from trajectory overlap, and spent extra time toggling trajectory visibilities and scrubbing the image sequence back and forth in time to provide the clearest possible view of the trajectory point to edit. The combination of these two factors additionally contributed to User 'B' taking the longest time to complete the trajectory editing tasks.

Trajectory occlusion due to overlap is a defined problem to which a number of solutions are available - while these improve the display, they do not simultaneously provide user access to the actual data itself. KOLAM attempts to compromise by overlaying the actual trajectory data including positional overlaps, and providing additional view filtering options to the user to restrict the number of trajectories visible in the view of interest. While this does aid in task completion, users do need additional time to utilize the extra features. While the author acknowledges that an interaction and visualization scheme which raises operational efficiency might indeed be possible, the author has not conceived such a scheme at the present time. Another factor playing a role is the prior frequency of usage of these KOLAM features by the different users: User 'A' had spent significantly more time using the trajectory editing feature than either User 'B' or User 'C'. Yet another factor that may have played a role is possible user bias against KOLAM's features, due to having used similar features through different interaction sequences with other software.

7.8 Conclusions

User interfaces (UI) and the associated user experiences (UX) of a software system must be comprehensively evaluated, both to obtain standard benchmarks of their performance as well as to engender trust in their use. To this end, both quantitative and qualitative usability evaluations were performed for various operations of the KOLAM visual analytics system. The operations chosen for the evaluations: 1-Click Object Tracking, Trajectory Editing and Segmentation Relabeling are representative of KOLAM's capabilities across the spectra of domains as well as of task complexity. Per ISO/IEC specifications, metrics corresponding to the efficiency, effectiveness and satisfaction of users to complete the specified tasks were evaluated. The experimental goals, software, data and tasks were formally defined, following which the usability experiments were carried out per the methodology outlined in the literature.

Perfect task completion rate (effectiveness) was achieved for all 3 tasks for all participant users. Task efficiencies for each of the tasks was computed from on the experimental data for each experiment, and the reasons for these values for each experiment were elaborated on. In the cases of current efficiency bottlenecks, possible directions for tackling the same were also explored. To conclude, KOLAM has been proven to be perfectly effective in tackling real world scenarios of which the chosen experiments are representative. With regards to efficiency, there is room for improvement, which will be addressed in the future work. Overall qualitative user satisfaction with KOLAM is also high at present, and is expected to improve further when the efficiency concerns to be addressed are further explored and implemented.

Chapter 8

Conclusions and Future Work

8.1 Summary of Conclusions

This thesis began by elaborating on the subject of visual analytics, the need for it as well as its mission. In exploring its building blocks, the inherent process, the motivation for interactivity and the challenges involved, we see that visual analytics can serve as a highly interdisciplinary collaboratory for a number of disciplines such as visualization, data mining, data fusion, machine learning and statistics. The author's unique contributions to the KOLAM visualization system which transformed it into an interactive visual analytics system were presented. This was followed by an in-depth look at KOLAM's architecture, with emphasis on the new addition of spatio-temporal data access, handling and management. Following a general look at the requirements of integrating KDD and visual analytics systems, the changes to the KOLAM user interface were explored, in terms of the challenges, design choices and how they evolved for the UI components. In concluding this introductory portion of the thesis, KOLAM's source and executable availability details are provided, followed by a division of the remainder of the thesis into case studies, the usability testing of KOLAM, ending with this chapter, the bibliography and the appendix containing KOLAM's manual and documentation.

The first case study dealt with the utility of KOLAM for exploratory visualization and analysis of both static and time-varying imagery of different types and formats, which underscored the usefulness of the system for a variety of applications requiring accurate multiple target tracking that could scale to a large number of objects and image sizes. To this end, the requisite algorithms, data structures, data I/O formats and human computer interfaces towards performing automated, manual and human-in-the=loop assisted target tracking were conceived, designed and implemented in KOLAM by the author. For WAMI imagery, KOLAM's ability to interactively animate gigapixel-sized images as well as perform automatic or manual tracking and trajectory visualization and analysis translates into applications in both the civilian and defense sectors.

The second case study underscores the importance of the visualization and editing of object trajectories, as these play a vital role in the generation of ground truth trajectory generation for datasets of interest, and are essential components of a human in the loop (assisted) object tracking system. KOLAM is able to improve human effectiveness measured in terms of productivity for accurately tracking multiple targets and reviewing and validating the results of tracking, in the automatic and manual tracking modes, and especially in the assisted tracking mode. The author's work in conceiving and implementing trajectory visualization and trajectory editing operations within the KOLAM system constitutes a crucial component of the solution to the problems and challenges involved in assisted object tracking, as well as automatic and manual object tracking.

The third case study identifies the problems faced and needs of pathologists as well as algorithm designers: with the former having a strong and growing need for tools to lessen the burden involved in analyzing and annotating large numbers of whole-slide histopathology imagery for training CMSS/CAD systems, and for sharing results among experts using a web-based interface. The latter require a means to interactively visualize the results of their segmentation algorithms on the WSI data, a means to generate ground-truth for training classifiers and improving and fine-tuning the performance of other algorithmic modules. The case stody then elaborates on the author's unique contributions which resulted in KOLAM's capabilities to meet both computational and clinical needs, as well as to bridge the gap between pathologists and algorithm developers; by providing a virtual microscope interface supporting a variety of WSI formats along with a novel method for rapidly labeling and correcting the tissue microenvironment stroma-epithelia region labels. Thus, due to the author's unique contributions, KOLAM now provides vital visanalysis software tools for medical and computational experts to develop the evolving field of computational histopathology.

User interfaces (UI) and the associated user experiences (UX) of a software system must be comprehensively evaluated, both to obtain standard benchmarks of their performance as well as to engender trust in their use. To this end, both quantitative and qualitative usability evaluations were performed for various operations of the KOLAM visual analytics system. The operations chosen for the evaluations: 1-Click Object Tracking, Trajectory Editing and Segmentation Relabeling are representative of KOLAM's capabilities across the spectra of domains as well as of task complexity. Per ISO/IEC specifications, metrics corresponding to the efficiency, effectiveness and satisfaction of users to complete the specified tasks were evaluated. KOLAM has been proven to be perfectly effective in tackling real world scenarios of which the chosen experiments are representative. With regards to efficiency, there is room for improvement, which will be addressed in the future work. Overall qualitative user satisfaction with KOLAM is also high at present, and is expected to improve further when the efficiency concerns to be addressed are further explored and implemented.

8.2 Future Work

The problems, the respective domains, the associated challenges, and finally the possible solutions as identified and tackled in this thesis belong to dynamic, evolving fields of study. As such, while several problems and their solutions have been the subject of this thesis, many new and interesting directions of research yet remain to be explored as future work. The proposed future work to be completed has been briefly discussed at multiple points throughout the thesis. A summary listing of the discussed points follows.

(a) Multiple real-time updating display sub-windows, for multiple purposes. Currently intended to visually monitor and focus on particular tracked objects. (b) Porting to mobile platforms (Android, iOS etc.). Will require significant alteration to memory model, data handling (all remote, local data of very low priority), need to monitor power usage by different code portions so as to not inefficiently drain battery on target devices. (c) Superior data structure for scalable trajectory handling. (d) Add extensive data analytics modules. (e) Completion of assisted tracking system. (f) Modify KOLAM to be a full-fledged data acquisition and handling system, handling acquisition of image data from the source to real-time post-processing of the data. (g) Support for IR and multispectral WAMI. (h) A knowledge base for managing thousands of trajectories from automated and interactive analysis of events. (i) Developing techniques to visualize statistical trajectory flow information, distinguishing between normal and abnormal patterns of activity, and processing event-based queries using semantic models. (j) Tiled image processing operators. (k) Cloud-enabled big visual data workflows. (l) Support for parallel operators and GPU implementations for interactive big data analytics. (m) Interfacing with image processing operator libraries. (n) Provenance of workflows or image processing chains. (o) Rendering point clouds and meshes. (p) Support for storing and reading annotations from a database like FireFly and DragonFly tools. (q) Extension to AR and VR environments for big data visualization.

The following is a description of an idea that struck the author, which is very tentative and has not been discussed previously. It lists a few steps for fundamentally changing KOLAM's user interfaces. (a) Minimize eye loss of focus from the object of interest through possibly several steps involved in completing a task such as object tracking. Productivity is maximized if the user is allowed to focus solely on the object rather than having to shift eye focus towards spatially distant UI components in order to perform the next steps. (b) Need to minimize hand movement to reduce overall fatigue when performing a time-consuming operation such as generating ground-truth over several thousand image frames in a long image sequence. Also factors into prevention of loss of visual focus from object currently being operated on. A task like object tracking involves several steps. Let these steps be defined by the hypothetical sequence a,b,c,d,e. One group of users might find a,b',c,d,e to be more natural and productive. Another group might prefer a,b',c',d,e. Here, b' and c' are action options that are currently undefined in the system, yet seem more natural to specific user interaction. Current methods for handling such user behavior include limiting the available interaction options (either through interface non-responsiveness, warnings and error messages, disabling interface components) or presenting as many choices as possible through a hierarchy of interface components: menus and sub-menus, which in turn bring up dialogs, each of which can potentially possess one or more specialized sub-components. The latter approach models a tree of possible interaction choices, with the user having to navigate down the tree, from the 'root' (a top level menu item, for example), down to a leaf (altering the state of an end-of-the-line component like a check box, and clicking an 'OK' button). This approach, while both straightforward and comprehensive, suffers from forcing a learning curve of increasing complexity upon the user, as more choices of interaction need to be made available. (c) An interesting option is a 'plastic' interface: one that does not provide too many interaction options out-of-the-box, but rather attempts to learn from user actions and patterns of actions. Given a particular task like object tracking that involves a pipeline of actions on KOLAM's part in order to be accomplished, the system can attempt to learn user behavior by recording user actions that are undefined in the pipeline and mapping them to expected actions. Subsequent performance of the task (object tracking in this instance) will allow for the previously performed user action to either to go to the next step or to complete the overall task. This approach thus attempts to make user interaction with the system more natural, rewarding and less stressful, thereby increasing user productivity.

Appendix A

Manual for KOLAM

A.1 Location of Manual for KOLAM

In keeping with the need for extensive documentation of features and user interactions for a complex system such as KOLAM, the author has prepared a separate document - the ‘Kolam Manual’, which serves as a comprehensive guide for all of KOLAM’s usage scenarios.

The manual has been physically appended to the end of the thesis document, and can be found after the Bibliography.

Bibliography

[1] P. C. Wong and J. Thomas. Guest editors' introduction–visual analytics. *IEEE Computer Graphics and Applications, 24 (5): 20-21*, 24(PNNL-SA-41935), 2004.

[2] D. Keim, J. Kohlhammer, G. Ellis, and F. Mansmann. Mastering the information age solving problems with visual analytics. 2010.

[3] D. A. Keim, F. Mansmann, J. Schneidewind, and H. Ziegler. Challenges in visual data analysis. In *Tenth International Conference on Information Visualization, 2006. IV 2006.*, pages 9–16. IEEE, 2006.

[4] D. A. Keim, F. Mansmann, and J. Thomas. Visual analytics: how much visualization and how much analytics? *ACM SIGKDD Explorations Newsletter*, 11(2):5–8, 2010.

[5] B. Shneiderman. *Designing the user interface: strategies for effective human-computer interaction*. Pearson Education India, 2010.

[6] K. A. Cook and J. J. Thomas. *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. IEEE Computer Society, Los Alamitos, CA, United States(US)., 2005.

[7] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *IEEE Symposium on Visual Languages, 1996*, pages 336–343. IEEE, 1996.

[8] J. W. Tukey. Exploratory data analysis. *Addison-Wesley Series in Behavioral Science: Quantitative Methods*, 1977.

[9] K. Palaniappan and J. Fraser. Multiresolution tiling for interactive viewing of large datasets. In *17th Int. AMS Conf. Interactive Information and Processing Systems (IIPS) for Meteorology, Oceanography and Hydrology*, pages 338–342, 2001.

[10] K. Palaniappan, R. Rao, and G. Seetharaman. Wide-area persistent airborne video: Architecture and challenges. In B. Banhu, C. V. Ravishankar, A. K. Roy-Chowdhury, H. Aghajan, and D. Terzopoulos, editors, *Distributed Video Sensor Networks: Research Challenges and Future Directions*, chapter 24, pages 349–371. Springer, 2011.

[11] J. Kopf, M. Uyttendaele, O. Deussen, and M.F. Cohen. Capturing and viewing gigapixel images. volume 26, pages 93–102. ACM, 2007.

[12] Q. Luan, S.M. Drucker, J. Kopf, Y.Q. Xu, and M.F. Cohen. Annotating gigapixel images. In *Proc. 21st ACM Symp. on User Interface Software and Technology*, pages 33–36, 2008.

[13] D.R. Jones, E.R. Jurrus, B.D. Moon, and K.A. Perrine. Gigapixel-size real-time interactive image processing with parallel computers. In *Proc. Parallel and Distributed Processing Symposium*, page 7 pp., 2003.

[14] L.L. Smarr, A.A. Chien, T. DeFanti, J. Leigh, and P.M. Papadopoulos. The optiputer. *Communications of the ACM*, 46(11):58–67, 2003.

[15] N. Schwarz, S. Venkataraman, L. Renambot, N. Krishnaprasad, V. Vishwanath, J. Leigh, A. Johnson, G. Kent, and A. Nayak. Vol-a-tile a tool for interactive exploration of large volumetric data on scalable tiled displays. In *Proceedings of the conference on Visualization'04*, pages 598–19. IEEE Computer Society, 2004.

[16] R. Singh, B. Jeong, L. Renambot, A. Johnson, and J. Leigh. Teravision: a distributed, scalable, high resolution graphics streaming system. In *cluster*, pages 391–400. IEEE, 2004.

[17] NK Krishnaprasad, V. Vishwanath, S. Venkataraman, AG Rao, L. Renambot, J. Leigh, AE Johnson, and B. Davis. Juxtaview-a tool for interactive visualization of large imagery on scalable tiled displays. In *Proceedings of the 2004 IEEE International Conference on Cluster Computing*, pages 411–420. IEEE Computer Society, 2004.

[18] Xi Wang, V. Vishwanath, Byungil Jeong, R. Jagodic, E. He, L. Renambot, A. Johnson, and J. Leigh. Lambdabridge: A scalable architecture for future generation terabit applications. In *Broadband Communications, Networks and Systems, 2006. BROADNETS 2006. 3rd International Conference on*, pages 1 –10, oct. 2006.

[19] T. Ni, G.S. Schmidt, O.G. Staadt, M.A. Livingston, R. Ball, and R. May. A survey of large high-resolution display technologies, techniques, and applications. In *Virtual Reality Conference, 2006*, pages 223–236. IEEE, 2006.

[20] L. Renambot, B. Jeong, H. Hur, A. Johnson, and J. Leigh. Enabling high resolution collaborative visualization in display rich virtual organizations. *Future Generation Computer Systems*, 25(2):161–168, 2009.

[21] T.A. DeFanti, J. Leigh, L. Renambot, B. Jeong, A. Verlo, L. Long, M. Brown, D.J. Sandin, V. Vishwanath, Q. Liu, et al. The optiportal, a scalable visualization, storage, and computing interface device for the optiputer. *Future Generation Computer Systems*, 25(2):114–123, 2009.

[22] B. Jeong, J. Leigh, A. Johnson, L. Renambot, M. Brown, R. Jagodic, S. Nam, and H. Hur. Ultrascale collaborative visualization using a display-rich global cyberinfrastructure. *IEEE Computer Graphics and Applications*, 30(3):71–83, 2010.

[23] K. Ponto, K. Doerr, and F. Kuester. Giga-stack: A method for visualizing giga-pixel layered imagery on massively tiled displays. *Future Generation Computer Systems*, 26(5):693–700, 2010.

[24] K. Ponto and F. Kuester. DIGI-Vis: Distributed interactive geospatial information visualization. In *IEEE Aerospace Conference*, pages 1–7, 2010.

[25] L. Renambot, A. Rao, R. Singh, B. Jeong, N. Krishnaprasad, V. Vishwanath, V. Chandrasekhar, N. Schwarz, A. Spale, C. Zhang, et al. Sage: the scalable adaptive graphics environment. In *Proceedings of WACE*, volume 2004, 2004.

[26] R. C. Gentleman, V. J. Carey, D. M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry, et al. Bioconductor: open software development for computational biology and bioinformatics. *Genome biology*, 5(10):R80, 2004.

[27] R. Theron. Visual analytics of paleoceanographic conditions. In *IEEE VAST*, pages 19–26. IEEE, 2006.

[28] TIBCO Spotfire. Spotfire 5. Online at: http://spotfire.tibco.com, 2012.

[29] S. Wolfram. *The Mathematica*. Cambridge university press Cambridge, 1999.

[30] SAS Institute. *SAS user's guide: statistics*, volume 2. Sas Inst, 1985.

[31] J. Ellson, E. Gansner, L. Koutsofios, S. C. North, and G. Woodhull. Graphvizopen source graph drawing tools. In *International Symposium on Graph Drawing*, pages 483–484. Springer, 2001.

[32] M. R. Berthold, N. Cebron, F. Dill, T. R. Gabriel, T. Kötter, T. Meinl, P. Ohl, K. Thiel, and B. Wiswedel. Knime-the konstanz information miner: version 2.0 and beyond. *AcM SIGKDD explorations Newsletter*, 11(1):26–31, 2009.

[33] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[34] R. Klinkenberg. *RapidMiner: Data mining use cases and business analytics applications*. Chapman and Hall/CRC, 2013.

[35] J. Lin, E. Keogh, S. Lonardi, J. P. Lankford, and D. M. Nystrom. Viztree: a tool for visually mining and monitoring massive time series databases. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 1269–1272. VLDB Endowment, 2004.

[36] J. Seo and B. Shneiderman. A knowledge integration framework for information visualization. In *From Integrated Publication and Information Systems to Information and Knowledge Environments*, pages 207–220. Springer, 2005.

[37] R. Santamaría, R. Therón, and L. Quintales. Bicoverlapper: a tool for bicluster visualization. *Bioinformatics*, 24(9):1212–1213, 2008.

[38] I.J. Roth. Real-time visualization of massive imagery and volumetric datasets. Master's thesis, University of Missouri-Columbia, 2006.

[39] J. Fraser, A. Haridas, G. Seetharaman, R. Rao, and K. Palaniappan. KOLAM: A cross-platform architecture for scalable visualization and tracking in wide-area motion imagery. In *Proc. SPIE Conf. Geospatial InfoFusion III*, volume 8747, page 87470N, 2013.

[40] A. Haridas, R. Pelapur, J. Fraser, F. Bunyak, and K. Palaniappan. Visualization of automated and manual trajectories in wide-area motion imagery. In *15th Int. Conf. Information Visualization*, pages 288–293, 2011.

[41] K. Palaniappan, A.F. Hasler, J.B. Fraser, and M. Manyin. Network-based visualization using the distributed image spreadsheet (DISS). In *17th Int. Conf. on Interactive Information and Processing Systems (IIPS) for Meteorology, Oceanography and Hydrology*, pages 399–403, 2001.

[42] K. Palaniappan and J.B. Fraser. Multiresolution tiling for interactive viewing of large datasets. In *17th International Conference on Interactive Information and Processing Systems*, 2001.

[43] G. J. Hunter and M. F. Goodchild. Managing uncertainty in spatial databases: Putting theory into practice. In *Papers from the Annual Conference-Urban and Regional Information Systems Association*, pages 15–15. URISA URBAN AND REGIONAL INFORMATION SYSTEMS, 1993.

[44] K. Palaniappan, A.F. Hasler, J.B. Fraser, and M. Manyin. Network-based visualization using the distributed image spreadsheet (DISS). In *17th Int. Conf. Interactive Information and Processing Systems*, 2001.

[45] K. Palaniappan, F. Bunyak, P. Kumar, I. Ersoy, S. Jaeger, K. Ganguli, A. Haridas, J. Fraser, R. Rao, and G. Seetharaman. Efficient feature extraction and likelihood fusion for vehicle tracking in low frame rate airborne video. In *13th Int. Conf. Information Fusion*, 2010.

[46] I. Ersoy, K. Palaniappan, and G. Seetharaman. Tracking in persistent wide-area motion imagery. In M.F. Pellechia and Sorensen R., editors, *SPIE Proc. Geospatial InfoFusion Systems and Solutions for Defense and Security Applications*, volume 8053, 2011.

[47] M. B. Rosson and J. M. Carroll. Scenario based design. *Human-computer interaction. boca raton, FL*, pages 145–162, 2009.

[48] P. Ehn. Scandinavian design: On participation and skill. *Participatory design: Principles and practices*, 41:77, 1993.

[49] E. B-N Sanders, E. Brandt, and T. Binder. A framework for organizing the tools and techniques of participatory design. In *Proceedings of the 11th biennial participatory design conference*, pages 195–198. ACM, 2010.

[50] D. Doermann and D. Mihalcik. Tools and techniques for video performance evaluation. In *15th Int. Conf. Pattern Recognition*, volume 4, pages 167–170. IEEE, 2000.

[51] R. Vezzani and R. Cucchiara. Annotation collection and online performance evaluation for video surveillance: The ViSOR project. In *IEEE 5th Int. Conf. Advanced Video and Signal Based Surveillance*, pages 227–234, 2008.

[52] S.K. Ralph, J. Irvine, M.R. Stevens, M. Snorrason, and D. Gwilt. Assessing the performance of an automated video ground truthing application. *Applied Image Pattern Recognition Workshop,*, 0:202–207, 2004.

[53] Home Office Scientific Development Branch. Imagery Library for Intelligent Detection Systems (i-LIDS). pages 445–448, 2006.

[54] A.T. Nghiem, F. Bremond, M. Thonnat, and V. Valentin. ETISEO: Performance evaluation for video surveillance systems. In *IEEE 5th Int. Conf. Advanced Video and Signal Based Surveillance*, 2007.

[55] C. Liu, W.T. Freeman, E.H. Adelson, and Y. Weiss. Human-assisted motion annotation. In *IEEE Conf. Computer Vision and Pattern Recognition*. IEEE, 2008.

[56] C.J. Carrano. Ultra-scale vehicle tracking in low spatial resolution and low framerate overhead video. In *Proceedings of SPIE*, volume 7445, 2009.

[57] R. Porter, A.M. Fraser, and D. Hush. Wide-area motion imagery. *IEEE Signal Processing Magazine*, 27(5):56–65, 2010.

[58] K. Palaniappan, F. Bunyak, P. Kumar, I. Ersoy, S. Jaeger, K. Ganguli, A. Haridas, J. Fraser, R. Rao, and G. Seetharaman. Efficient feature extraction and likelihood

fusion for vehicle tracking in low frame rate airborne video. In *13th Int. Conf. Information Fusion*, 2010.

[59] N.P. Cuntoor, A. Basharat, A.G.A. Perera, and A. Hoogs. Track initialization in low frame rate and low resolution videos. In *Int. Conf. Pattern Recognition*, pages 3640–3644. IEEE, 2010.

[60] V. Reilly, H. Idrees, and M. Shah. Detection and tracking of large number of targets in wide area surveillance. In *11th European Conf. Computer Vision*, pages 186–199. Springer-Verlag, 2010.

[61] X. Jiangjian, C. Hui, H. Sawhney, and H. Feng. Vehicle detection and tracking in wide field-of-view aerial video. In *IEEE Conf. Computer Vision and Pattern Recognition*, pages 679 – 684, 2010.

[62] Air Force Research Laboratory. Columbus Large Image Format (CLIF) 2007 dataset.

[63] A. L. Chan. A description on the second dataset of the U.S. Army Research Laboratory Force Protection Surveillance System. Technical Report ARL-MR-0670, Army Research Laboratory, Adelphi, MD, 2007.

[64] A. H. Beck, A. R. Sangoi, S. Leung, R. J. Marinelli, T. O. Nielsen, M. J. van de Vijver, R. B. West, M. van de Rijn, and D. Koller. Systematic analysis of breast cancer morphology uncovers stromal features associated with survival. *Science Translational Medicine*, 3(108):108–113, 2011.

[65] G. Finak, N. Bertos, F. Pepin, S. Sadekova, M. Souleimanova, H. Zhao, H. Chen, G. Omeroglu, S. Meterissian, A. Omeroglu, M. Hallett, and M. Park. Stromal gene

expression predicts clinical outcome in breast cancer. *Nature Medicine*, 14(5):518–527, 2008.

[66] K. Pietras and A. Östman. Hallmarks of cancer: Interactions with the tumor stroma. *Exp. Cell Res.*, 316(8):1324–1331, 2010.

[67] E. de Kruijf, J. van Nes, C. van de Velde, H. Putter, V. Smit, G. Liefers, P. Kuppen, R. Tollenaar, and W. Mesker. Tumor-Stroma Ratio within the Primary Tumor Is a Prognostic Factor in Early Breast Cancer Patients, Especially in Triple-Negative-Carcinoma Patients. *Cancer Res.*, 69(24 Supplement):3049–3049, 2009.

[68] A. Planche, M. Bacac, P. Provero, C. Fusco, M. Delorenzi, J.C. Stehle, and I. Stamenkovic. Identification of prognostic molecular features in the reactive stroma of human breast and prostate cancer. *PLoS One*, 6(5), 2011.

[69] T. J. Dekker, C. J. H. Van De Velde, G. W. Van Pelt, J. R. Kroep, J. P. Julien, V. Smit, R. Tollenaar, and W. E. Mesker. Prognostic significance of the tumor-stroma ratio: Validation study in node-negative premenopausal breast cancer patients from the EORTC perioperative chemotherapy (POP) trial (10854). *Breast Cancer Res. Treat.*, 139(2):371–379, 2013.

[70] J.M. Chen, A.P. Qu, L.W. Wang, J.P. Yuan, F. Yang, Q.M. Xiang, N. Maskey, G.F. Yang, J. Liu, and Y. Li. New breast cancer prognostic factors identified by computer-aided image analysis of he stained histopathology images. *Scientific Reports*, 5, 2015.

[71] E Versi. ”Gold standard” is an appropriate term. *BMJ: British Medical Journal*, 305(6846):187, 1992.

[72] A. Haridas, F. Bunyak, and K. Palaniappan. Interactive segmentation relabeling for classification of whole-slide histopathology imagery. In *IEEE Int. Symposium on Computer-Based Medical Systems (CBMS)*, pages 84–87, Jun 2015.

[73] F. Bunyak, A. Hafiane, Z. Al-Milagi, I. Ersoy, A. Haridas, and K. Palaniappan. A segmentation-based multi-scale framework for the classification of epithelial and stromal tissues in h & e images. In *Proc. IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 450–453, 2015.

[74] The OpenClinical Project. Decision Support Systems. Online at: http://www.openclinical.org/dss.html, 2001.

[75] D. A. Gutman, J. Cobb, D. Somanna, Y. Park, F. Wang, T. Kurc, J. H. Saltz, D. J. Brat, L. Cooper, and J. Kong. Cancer Digital Slide Archive: an informatics resource to support integrated in silico analysis of TCGA pathology data. *J. American Medical Informatics Association*, 20(6):1091–1098, 2013.

[76] L. A. Teot, R. Sposto, A. Khayat, S. Qualman, G. Reaman, and D. Parham. The problems and promise of central pathology review: development of a standardized procedure for the children's oncology group. *Ped. and Develop. Pathology*, 10(3):199–207, 2007.

[77] B. Sabata. Digital Pathology Imaging-The Next Frontier in Medical Imaging. In *2012 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, pages 1–6. IEEE, 2012.

[78] V. B. S. Prasath, F. Bunyak, P. Dale, S. R. Frazier, and K. Palaniappan. Segmentation of breast cancer tissue microarrays for computer-aided diagnosis in pathology. In *IEEE Healthcare Innovation Conference*, 2012.

[79] F. Bunyak, A. Hafiane, and K. Palaniappan. Histopathology tissue segmentation by combining fuzzy clustering with multiphase vector level sets. In *Software Tools and Algorithms for Biological Systems*, pages 413–424. Springer, 2011.

[80] A. Hafiane, F. Bunyak, and K. Palaniappan. Evaluation of level set-based histology image segmentation using geometric region criteria. In *IEEE Int. Symp. Biomedical Imaging*, 2009.

[81] A. Hafiane, F. Bunyak, and K. Palaniappan. Fuzzy clustering and active contours for histopathology image segmentation and nuclei detection. *Lecture Notes in Computer Science (ACIVS)*, 5259:903–914, 2008.

[82] A. Hafiane, F. Bunyak, and K. Palaniappan. Clustering initiated multiphase active contours and robust separation of nuclei groups for tissue segmentation. In *IEEE Int. Conf. Pattern Recognition*, page Online, 2008.

[83] S. Kothari, J. H. Phan, A. O. Osunkoya, and M. D. Wang. Biological Interpretation of Morphological Patterns in Histopathological Whole-Slide Images. In *Proceedings of the ACM Conference on Bioinformatics, Computational Biology and Biomedicine*, pages 218–225. ACM, 2012.

[84] R. Gutiérrez, F. Gómez, L. Roa-Peña, and E. Romero. A supervised visual model for finding regions of interest in basal cell carcinoma images. *Diag. Pathology*, 6(1):14, 2011.

[85] D. Romo, E. Romero, and F. González. Learning regions of interest from low level maps in virtual microscopy. *Diag. Pathology*, 6(Suppl 1):8, 2011.

[86] V. Raghunath, M. O. Braxton, S. A. Gagnon, T. T. Brunyé, K. H. Allison, L. M. Reisch, D. L. Weaver, J. G. Elmore, and L. G. Shapiro. Mouse cursor movement and eye tracking data as an indicator of pathologists attention when viewing digital whole slide images. *Journal of pathology informatics*, 3, 2012.

[87] A. Qu, J. Chen, L. Wang, J. Yuan, F. Yang, Q. Xiang, N. Maskey, G. Yang, J. Liu, and Y. Li. Two-step segmentation of hematoxylin-eosin stained histopathological images for prognosis of breast cancer. In *IEEE International Conference on Bioinformatics and Biomedicine*, pages 218–223, 2014.

[88] N. Linder, J. Konsti, R. Turkki, E. Rahtu, M. Lundin, S. Nordling, C. Haglund, T. Ahonen, M. Pietikäinen, and J. Lundin. Identification of tumor epithelium and stroma in tissue microarrays using texture analysis. *Diagnostic Pathology*, 7(1):22, 2012.

[89] M. D. DiFranco, G. OHurley, E. W. Kay, R. W. G. Watson, and P. Cunningham. Ensemble based system for whole-slide prostate cancer probability mapping using color texture features. *Computerized Medical Imaging and Graphics*, 35(7):629–645, 2011.

[90] V. Roullier, O. Lézoray, V-T. Ta, and A. Elmoataz. Multi-resolution graph-based analysis of histopathological whole slide images: Application to mitotic cell extraction and visualization. *Comp. Med. Imaging and Graphics*, 35(7):603–615, 2011.

[91] J. Kong, O. Sertel, H. Shimada, K. L. Boyer, J. H. Saltz, and M. N. Gurcan. Computer-aided evaluation of neuroblastoma on whole-slide histology images: Classifying grade of neuroblastic differentiation. *Pattern Recognition*, 42(6):1080–1092, 2009.

[92] S. Doyle, M. Feldman, J. Tomaszewski, and A. Madabhushi. A boosted Bayesian multiresolution classifier for prostate cancer detection from digitized needle biopsies. *IEEE Trans. Biomedical Engineering*, 59(5):1205–1218, 2012.

[93] C. Huang, A. Veillard, L. Roux, N. Loménie, and D. Racoceanu. Time-efficient sparse analysis of histopathological whole slide images. *Computerized Medical Imaging and Graphics*, 35(7):579–591, 2011.

[94] R. Singh, L. Chubb, L. Pantanowitz, A. Parwani, et al. Standardization in digital pathology: Supplement 145 of the DICOM standards. *Journal of Pathology Informatics*, 2(1):23, 2011.

[95] C. Allan, J. Burel, J. Moore, C. Blackburn, M. Linkert, S. Loynton, D. MacDonald, W. J. Moore, C. Neves, A. Patterson, et al. OMERO: flexible, model-driven data management for experimental biology. *Nature Methods*, 9(3):245–253, 2012.

[96] W. Jeong, J. Schneider, S. G. Turney, B. E. Faulkner-Jones, D. Meyer, R. Westermann, R. C. Reid, J. Lichtman, and H. Pfister. Interactive histology of large-scale biomedical image stacks. *IEEE Trans. Visualization and Computer Graphics*, 16(6):1386–1395, 2010.

[97] Bitplane Inc. Imaris v 8.0.1. Online at: http://bitplane.com, 2014.

[98] W. Schroeder, K. Martin, L. S. Avila, and C. C. Law. The Visualization Toolkit User's Guide, Version 4.0. *Kitware Publishing*, 4, 2001.

[99] C. A. Schneider, W. S. Rasband, and K. W. Eliceiri. NIH Image to ImageJ: 25 years of image analysis. *Nature methods*, 9(7):671–675, 2012.

[100] J. Schindelin, I. Arganda-Carreras, E. Frise, V. Kaynig, M. Longair, T. Pietzsch, S. Preibisch, C. Rueden, S. Saalfeld, B. Schmid, et al. Fiji: an open-source platform for biological-image analysis. *Nature Methods*, 9(7):676–682, 2012.

[101] Leica Biosystems. ePathology Imagescope. Online at: http://www.leicabiosystems.com/pathology-imaging/aperio-epathology/integrate/imagescope/, 2015.

[102] C. Sommer, C. Straehle, U. Koethe, F. Hamprecht, et al. ilastik: Interactive learning and segmentation toolkit. In *2011 IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pages 230–233. IEEE, 2011.

[103] R. Marée, B. Stévens, L. Rollus, N. Rocks, X. M. Lopez, I. Salmon, D. Cataldo, and L. Wehenkel. A rich internet application for remote visualization and collaborative annotation of digital slides in histology and cytology. *Diagnostic Pathology*, 8(Suppl 1):1–4, 2013.

[104] PathCore Inc. Sedeen 5. Online at: http://pathcore.ca/sedeen, 2015.

[105] H. Ding, C. Wang, K. Huang, and R. Machiraju. GRAPHIE: graph based histology image explorer. *BMC bioinformatics*, 16(Suppl 11):S10, 2015.

[106] U. Catalyurek, M. D. Beynon, C. Chang, T. Kurc, A. Sussman, and J. Saltz. The virtual microscope. *IEEE Trans. Information Technology in Biomedicine*, 7(4):230–248, 2003.

[107] D. Keim, G. Andrienko, J. Fekete, C. Görg, J. Kohlhammer, and G. Melançon. Visual analytics: Definition, process, and challenges. *Information Visualization*, pages 154–175, 2008.

[108] A. Goode, B. Gilbert, J. Harkes, D. Jukic, M. Satyanarayanan, et al. OpenSlide: A vendor-neutral software foundation for digital pathology. *Journal of Pathology Informatics*, 4(1):27, 2013.

[109] M. Linkert, C. T. Rueden, C. Allan, J. Burel, W. Moore, A. Patterson, B. Loranger, J. Moore, C. Neves, D. MacDonald, et al. Metadata matters: access to image data in the real world. *J. Cell Biology*, 189(5):777–782, 2010.

[110] M. T. McCann, J. A. Ozolek, C. A. Castro, B. Parvin, and J. Kovacevic. Automated histology analysis: Opportunities for signal processing. *IEEE Signal Processing Magazine*, 32(1):78–87, 2015.

[111] G. Rolls. An Introduction to Specimen Preparation. Online at: http://www.leicabiosystems.com/pathologyleaders/an-introduction-to-specimen-preparation/, 2011.

[112] M. Macenko, M. Niethammer, J. S. Marron, D. Borland, J. T. Woosley, X. Guan, C. Schmitt, and N. E. Thomas. A method for normalizing histology slides for quantitative analysis. In *Proceedings of the Sixth IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pages 1107–1110. IEEE, 2009.

[113] A. E. Carpenter, T. R. Jones, M. R. Lamprecht, C. Clarke, I. H. Kang, O. Friman, D. A. Guertin, J. H. Chang, R. A. Lindquist, J. Moffat, et al. Cellprofiler: image analysis software for identifying and quantifying cell phenotypes. *Genome biology*, 7(10):R100, 2006.

[114] J. Weickert. *Anisotropic diffusion in image processing*, volume 1. Teubner Stuttgart, 1998.

[115] Karel Zuiderveld. In *Graphics Gems IV*, pages 474–485. Academic Press Professional, Inc., San Diego, CA, USA, 1994.

[116] A. C. Ruifrok and D. A. Johnston. Quantification of histochemical staining by color deconvolution. *Analytical and quantitative cytology and histology/the International Academy of Cytology [and] American Society of Cytology*, 23(4):291–299, 2001.

[117] A. M. Khan, N. Rajpoot, D. Treanor, and D. Magee. A nonlinear mapping approach to stain normalization in digital histopathology images using image-specific color deconvolution. *IEEE Transactions on Biomedical Engineering*, 61(6):1729–1738, 2014.

[118] M. T. McCann, J. Majumdar, C. Peng, C. Castro, J. Kovacevic, et al. Algorithm and benchmark dataset for stain separation in histology images. In *IEEE International Conference on Image Processing (ICIP)*, pages 3953–3957. IEEE, 2014.

[119] M. Gavrilovic, J. C. Azar, J. Lindblad, C. Wahlby, E. Bengtsson, C. Busch, and I. B. Carlbom. Blind color decomposition of histological images. *IEEE Transactions on Medical Imaging*, 32(6):983–994, 2013.

[120] E. Mercan, S. Aksoy, L. G. Shapiro, D. L. Weaver, T. Brunye, and J. G. Elmore. Localization of diagnostically relevant regions of interest in whole slide images. In *IEEE 22nd International Conference on Pattern Recognition (ICPR), 2014*, pages 1179–1184. IEEE, 2014.

[121] R. G. Bias. The pluralistic usability walkthrough: coordinated empathies. In *Usability inspection methods*, pages 63–76. John Wiley & Sons, Inc., 1994.

[122] J. D. Gould and C. Lewis. Designing for usability: key principles and what designers think. *Communications of the ACM*, 28(3):300–311, 1985.

[123] Niels Ebbe Jacobsen, Morten Hertzum, and Bonnie E John. The evaluator effect in usability studies: Problem detection and severity judgments. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 42, pages 1336–1340. SAGE Publications Sage CA: Los Angeles, CA, 1998.

[124] A. Pospischil and G. Folkers. How much reproducibility do we need in human and veterinary pathology? *Experimental and Toxicologic Pathology*, 67(2):77–80, 2015.

[125] W Iso. 9241-11. ergonomic requirements for office work with visual display terminals (vdts). *The international organization for standardization*, 45, 1998.

[126] N. Bevan. Quality in use: Meeting user needs for quality. *Journal of systems and software*, 49(1):89–96, 1999.

[127] R. Pelapur, F. Bunyak, K. Palaniappan, and G. Seetharaman. Vehicle detection and orientation estimation using the radon transform. In *Proc. SPIE Conf. Geospatial InfoFusion III (Defense, Security and Sensing: Sensor Data and Information Exploitation)*, volume 8747, page 87470I, 2013.

[128] J. Rubin and D. Chisnell. *Handbook of usability testing: how to plan, design and conduct effective tests*. John Wiley & Sons, 2008.

[129] John Brooke et al. SUS-A quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.

KOLAM 3.0 (v. 199)
The Complete Reference

Anoop Haridas
(Author)

Kannappan Palaniappan
(Faculty Supervisor)

Department of Computer Science
University of Missouri
Columbia, MO 65201, USA

`ahkrc@mail.missouri.edu`
`palaniappank@missouri.edu`

Rev. 2: Apr. 20, 2016.
Rev. 1: Nov. 19, 2014.
Rev. 0: Apr. 08, 2013.

CONTENTS

ACKNOWLEDGEMENTS

KOLAM is a software package for interactive visualization of massive geospatial datasets and 2D biomedical imagery on standard desktop and mobile computers developed at the University of Missouri. KOLAM uses multi-resolution tiled pyramid data structures and an efficient cache memory hierarchy to deliver interactivity independent of dataset size. KOLAM supports embedded datasets at multiple resolutions that may be hundreds of gigabytes to hundreds of terabytes in size. In addition to rapid roam & zoom operations, KOLAM supports an arbitrary number of simultaneously visible embedded layers, on-the-fly mosaics, colormap lookup tables and histogram enhancement, projection of images onto a spherical surface and elevation maps or terrain rendering. KOLAM 3.0 enables animating pyramid files (2D+t) plus other supported file types, supports interactive object tracking using either a MATLAB or OCTAVE interface, interactive ground truth generation and annotation, multi-object user configurable trajectory display and interactive segmentation relabeling.

KOLAM 3.0 is developed by Anoop Haridas and Kannappan Palaniappan. Past developers are Joshua Fraser, Ian Roth and Jared Hoberock. Rengarajan Pelapur contributed full feature testing, performance and error feedback, and building and porting for multiple target platforms. Ilker Ersoy contributed testing and feedback. ©2002 - 2016, All rights reserved.

The development of KOLAM and its associated out-of-core 2D and 3D rendering technologies using memory hierarchies was partially supported by grants from NASA Goddard Space Flight Center NAG 5-3900, NASA Stennis Space Center NAG-13-99014, the U.S. Air Force Research Laboratory (AFRL) under agreements AFRL FA8750-09-C-0226, FA8750-11-C-0091, FA8750-11-1-0073 and Leonard Wood Institute (LWI 181223) in cooperation with the U.S. Army Research Laboratory (ARL) under Cooperative Agreement Number W911NF-07-2-0062. Approved for public release. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied of AFRL, LWI, ARL, NASA or the U.S. Government.

This manual is authored by Anoop Haridas ©2013 - 2016. This revision corresponds to KOLAM 3.0 v. 199, built against Qt 4.7.4 and Qt 5.3.2 (©The Qt Company).

Disclaimer

This program is provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

CHAPTER 1

INTRODUCTION

Exploratory visualization and analytics tools are powerful methods to support multiple data-streams and navigation through very large datasets. Such tools offer ways to mitigate the data deluge being faced by users and analysts and are useful for providing insight into complex patterns in scientific, geospatial, satellite, and surveillance applications. KOLAM is a scalable and extensible framework for high-resolution, high throughput image data visualization with applications in a variety of image analysis domains (including WAMI). It is platform and operating system independent and supports embedded datasets scalable from hundreds of gigabytes to petabytes in size on architectures ranging from clusters to netbooks.

KOLAM uses a scalable data structure and cache management strategies to support large datasets along with an extensive set of image processing and analysis features. In order to efficiently manage the display of large time-varying imagery, each image is partitioned or tiled into sub-images of the larger dataset through the process of a quad-tree like regular tiling. These tiles allow random access to spatially coherent regions of the image and thus allow for on-demand access to those parts of the image needed for display. In order to provide interactive zooming of the image across scale, multiresolution tiling is used. A multiresolution pyramid is created by scaling the image in half along each dimension like a quad-tree until a minimum size (*i*.e. single tile) is reached. Following successive scaling, each resolution is divided into fixed-sized tiles. For our system, we've chosen fixed-resolution (compressed) tiles across all resolutions of the image. This tiled multiresolution data organization creates a pyramidal structure for each frame in motion imagery as shown in Figure 1.1.

KOLAM supports a robust animation and novel tracking interface that enables smooth animation of WAMI sequences and supports one-click tracking of multiple objects. KOLAM is capable of interfacing with different tracking algorithms and visualizing the resulting trajectories that may be composed of multiple segments. The coupling between the visualization environment and the open architecture

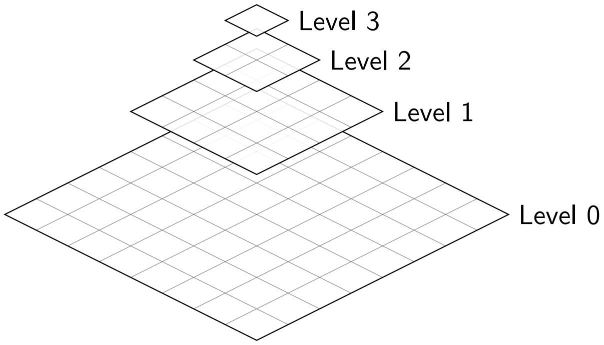


Figure 1.1: Multiresolution tiled images are used by KOLAM for efficient display.

supporting a collection of tracking algorithms. A feature-rich tracking interface enables simultaneous visualization of multiple tracks, context-sensitive track operations, track archival and retrieval, moving object ground truth generation, track editing and annotation.

The need for user-driven models in the analysis of WAMI data to address challenging or unsolved problems in WAMI exploitation include: automated search tools, a complementary visual analytics tool for analysis, human-computer interaction that is capable of capturing user domain knowledge to improve productivity and search tools applied to multitarget tracking results. The following chapters provide a detailed description and usage details about all the components of KOLAM.

Please cite our work

If you use KOLAM in your projects, please cite our publications as described here.

When using KOLAM for all visualization and analysis needs, please cite our InfoVis 2011 paper [1], our SPIE 2013 paper [2], our Fusion 2010 paper [3], as well as our IIPS 2001 papers [4] [5].

When using KOLAM's interactive segmentation relabeling module, please also cite our CBMS 2015 paper [6] and our BIBM 2015 paper [7].

CHAPTER 2

BUILD, INSTALL & RUN

This chapter details the building and installation steps for KOLAM for multiple target platforms. External library dependencies are also explained here.

IMPORTANT: For each OS, there are **two** scenarios: **first**; using the respective pre-built binaries, and **second**; building KOLAM from a source distribution. For the **latter** use case, we **strongly recommend** that the user possess the necessary OS-specific knowledge and skill set regarding building an executable from a large source tree, dealing with multiple external dependencies etc.

1. Windows OSes (XP, Vista, 7, 8, 8.1, 10)

Using the pre-built executable

Using the pre-built KOLAM Windows executable is the easiest of all the OS options - with the exception of external programs such as the MATLAB Compiler Runtime, object trackers, algorithm plugins etc. the user **does not need** to perform any additional setup, besides downloading the KOLAM compressed archive (a .zip file), uncompressing the files into a desired location, and running the executable. In case the user's copy of Windows has not been patched with the latest software updates, attempting to run the KOLAM executable might produce some seemingly meaningless errors. These arise due to one or more versions of the *Microsoft Visual C++ Redistributable* missing from the user's system. Should this occur, please **download and install every version** of the Microsoft Visual C++ Redistributable starting from the 2005 version onward. This includes versions 2005, 2008, 2010, 2012 (and 2013, if available). This also includes the SP1 versions. Unless KOLAM is specifically built as 64-bit, which is not the case by default, the x64 versions of these redistributables are not needed.

Building with MinGW

MinGW stands for Minimalist GNU for Windows. MinGW thus provides access to GNU G++ and GCC, and is being used for developing the Windows flavored GNU version of KOLAM. All Qt versions have pre-built MinGW versions of the API, which may be freely downloaded from the Qt website. Under Windows, building with MinGW is the alternative to building KOLAM with Visual Studio (which is explained in the next subsection). Follow these steps:

(1) Download and install the desired MinGW version of the Qt API (version 4.8.4 for Qt 4, version 5.5.1 for Qt 5; or the closest available higher numbered versions in each case). During the installation procedure, if installing Qt Creator is provided as an unchecked option, we recommend enabling it. We will use Qt Creator to build KOLAM. KOLAM may also be built via a Qt - configured DOS command prompt, if such an approach is preferable.

2. Configure the system environment variables. This includes the 'HOME', 'INCLUDE', PATH', 'QMAKESPEC' and 'QTDIR' user variables, as well as the 'INCLUDE' and 'Path' system variables. Details can be found at:

`http://doc.qt.io/qt-5/windows-building.html`.

3. Set up Qt Creator. This IDE which is provided along with the Qt API is highly versatile and may be used for developing non-Qt C++ projects as well. The steps for configuring the different aspects of any Qt project within Qt Creator can be found at:

`http://doc.qt.io/qtcreator/creator-configuring.html`,

and should be followed for configuring the KOLAM source project.

4. Build the KOLAM project, after finishing setting up all external dependencies (explained later in this chapter). If the build procedure is unsuccessful, please make sure that all steps were followed. If a step was missed, perform a clean ('make clean') of the project (within the same top-level menu as the build option of the project), and perform all the build steps again. If the problem persists, please contact the authors for support.

Building with Visual Studio

The KOLAM source tree can also be built with Microsoft Visual Studio, from version 2010 onward. Follow these steps:

(1) Download and install the desired Visual Studio version of the Qt API (version 4.8.4 for Qt 4, version 5.5.1 for Qt 5; or the closest available higher numbered versions in each case). During the installation procedure, if installing Qt Creator is provided as an unchecked option, we recommend enabling it. We will use Qt Creator to build KOLAM. KOLAM may also be built via a Qt - configured DOS command prompt, if such an approach is preferable.

2. Configure the system environment variables. This includes the 'HOME', 'INCLUDE', PATH', 'QMAKESPEC' and 'QTDIR' user variables and the 'INCLUDE' and 'Path' system variables. Details can be found at:

`http://doc.qt.io/qt-5/windows-building.html`.

3. Set up Qt Creator. This IDE which is provided along with the Qt API is highly versatile and may be used for developing non-Qt C++ projects as well. The steps for configuring the different aspects of any Qt project within Qt Creator can be found at:

`http://doc.qt.io/qtcreator/creator-configuring.html`,

and should be followed for configuring the KOLAM source project. Alternatively, Visual Studio can be directly used to build KOLAM. Download and install the Visual Studio plugin from the Qt website. Complete setup and configuration steps can be found at:

`http://doc.qt.io/vs-addin/`.

4. Build the KOLAM project, after finishing setting up all external dependencies (explained later in this chapter). If the build procedure is unsuccessful, please make sure that all steps were followed. If a step was missed, perform a clean of the project (within the same top-level menu as the build option of the project), and perform all the build steps again. If the problem persists, please contact the authors for support.

2. Linux OSes (CentOS, Arch, Fedora)

Using the pre-built executable

Under most circumstances, usage under Linux will involve building from source. However, a pre-built executable is also available for the sake of completeness. We offer no guarantees regarding its ability to operate, given the wide range of variability of the different aspects of a random Linux installation. Open a terminal, navigate to the KOLAM directory, and run the shell script file (file with extension '.sh') that is in the same directory as the actual executable file (which typically will either not have an extension, or the extension '.bin'). The script takes care of issues such as temporarily adding to the system's 'LD_LIBRARY_PATH' variable and changes to any other Linux environmental variables.

Building from source

The steps involved in building KOLAM under Linux include downloading and installing the desired version of Qt, getting the desired KOLAM version from the svn or git (as applicable) repository, ensuring that all of KOLAM's external dependencies are taken care of, and finally building KOLAM. More details follow:

(1) Download and install the desired Qt version for Linux. All details can be found at:

`http://doc.qt.io/qt-5/linux.html`.

(2) Get the desired version of KOLAM from the svn or git repository. Links to these will be provided soon. In the interim, please contact the authors for the source tree.

(3) With the exception of the 'libtiff' library, all of KOLAM's external dependencies should come installed with Linux by default. If not, simply use your package managers to download and install these libraries. Please look at 'External Dependencies' later in this chapter for the list of libraries.

(4) In the final build step, run the following commands within KOLAM's 'src' directory:

```
qmake -spec linux-g++
make
```

This assumes that the path to 'qmake' has been corectly set in the environment variables. If not, do so, and run the commands again. If the build procedure is unsuccessful, please contact the authors for support.

3. Mac OSX (10.7, 10.8, 10.9, 10.10)

Using the pre-built executable

KOLAM can be run without issues from the pre-built app bundle. A download link for the same will be provided soon. In the interim, please contact the authors.

Building from source

The steps involved in building KOLAM under Mac OSX include downloading and installing the desired version of Qt, getting the desired KOLAM version from the svn or git (as applicable) repository, ensuring that all of KOLAM's external dependencies are taken care of, and finally building KOLAM. More details follow:

(1) Download and install the desired Qt version for Mac OSX. All details can be found at:

`http://doc.qt.io/qt-5/osx.html`.

(2) Get the desired version of KOLAM from the svn or git repository. Links to these will be provided soon. In the interim, please contact the authors for the source tree.

(3) With the exception of the 'libtiff' library, all of KOLAM's external dependencies should come installed with Mac OSX by default. If not, simply use your package managers to download and install these libraries. Please look at 'External Dependencies' later in this chapter for the list of libraries.

(4) In the final build step, run the following commands within KOLAM's 'src' directory:

```
qmake -spec macx-g++
```

make

This assumes that the path to 'qmake' has been corectly set in the environment variables. If not, do so, and run the commands again. If the build procedure is unsuccessful, please contact the authors for support.

4. External dependencies

KOLAM depends on a number of 3rd party libraries to provide efficient and crucial functionality. These libraries along with their download and install instructions are listed below. When installing any external library, it is crucial to ensure that all necessary system paths are properly set. On *NIX platforms, the shell being used (bash, tcsh etc.) needs to be taken into account when running the installation commands.

(a) The POSIX threads (***pthreads***) library: The POSIX threads library is a default part of *NIX OSes, which include Linux and MacOS systems. For Windows, different DLL versions are available for download at:

`https://www.sourceware.org/pthreads-win32/`,

depending on whether development is being done under MinGW/Cygwin (pthreadGC2.dll) or Visual Studio (pthreadVC2.dll) environments. The .dll and .lib files need to be placed in the '/extern/win32' directory.

(b) The Joint Photographic Experts Group (***JPEG***) library: The JPEG library is used by KOLAM for fast tile decompression for certain supported image file formats. The library needs to be built from source on all platforms, if not installed at a prior time. Users need to download version '6b' of the library from:

`http://jpegclub.org/support/`.

GCC and Visual Studio specific Makefiles are available within the source tree so that the library may be built using compilers of choice under all major OSes. For Windows and MacOS, generated library files must be placed within the OS specific directory under '/extern'.

(c) The ***Zlib*** library: This compression library is included as part of the libtiff library, and details will be explained in that subsection.

(d) The Tagged Image File Format (***TIFF***) library: The libtiff library is required by KOLAM as it adds support for the TIFF and BigTIFF image file formats. The library source may be downloaded at:

`http://www.remotesensing.org/libtiff/`.

Users must remember to obtain a version greater than 4.0.0 of the library, in order to get BigTIFF support. Follow similar build steps as for the previously listed libraries. The libtiff library has the zlib library as a dependency of its own, so download and install this library from source as well if it hasn't already been installed. Under Windows and Mac, place all compiled library files under the OS specific subdirectories of the '/extern' directory.

(e) The ***Bio-Formats LOCI*** library: The links for downloading as well as installing the Bio-Formats library can be found at:

`http://loci.wisc.edu/software/bio-formats`.

Please note that this library is currently supported for Windows ONLY. Support for other platforms is being added.

The MATLAB Compiler Runtime (MCR)

The MCR needs to be installed **ONLY IF** the user desires to perform automatic tracking of objects (usually vehicles in Wide-Area Motion Imagery). MCR installation has been most tested on the Windows platform. Links to download Windows, Linux and Mac versions of the MCR are available on the MATLAB website. In the event that our MATLAB trackers use a version of the MCR that is older than the versions of the MCR publicly available for download, please contact Dr. Palaniappan via email in order to get in contact with the developers of the tracker programs, so that they may provide the user with the correct version of the MCR to install. Once correctly installed, the MCR typically does not require additional configuration. Invoking the MATLAB tracker program involves the sole step of setting the path to the tracker executable within the 'KOLAM Preferences' tool.

5. Running KOLAM

KOLAM may be executed in one of two ways: first; by double-clicking an icon representing a shortcut to the KOLAM executable, and second; from a command prompt or terminal. While the former method is suitable for most of KOLAM's operation, certain execution conditions (such as running KOLAM with command-line parameters, or running KOLAM in a multi-monitor wall display environment) require the latter method. Command-line execution of KOLAM, alongwith the possible parameters, is explained below.

Command-line execution

The main command for executing KOLAM with options from the command line is of the form:

kolam.exe [*option(s)*] [*imageName* [-cm *colormapName*]] ...

'kolam.exe', '*imageName*' and '*colormapName*' are the full paths (or pre-defined aliases) to the KOLAM executable, the name of the image/image sequence, and the name of the colormap file. Multiple image or image sequence paths may be supplied in this command, separated by single spaces. Each image or image sequence may have one or more associated colormaps, each of which is preceded by '-cm'. '*option(s)*' refers to one or more options that may be supplied in the above command, and these are listed in the following table.

KOLAM Options & their Descriptions	
-h / –help	Command-line usage help
-a	Toggle the use of the alpha channel.
-x *offset*	Horizontal offset. *offset* is in pixels.
-y *offset*	Vertical offset. *offset* is in pixels.
-z *zoomFactor*	Specify zoom. *zoomFactor* is a floating point number.
-k *offset*	Specify trajectory file. *filename* is the full path to the kw18 trajectory file.
-cm *filename*	*File name*, with full path, of colormap file.

CHAPTER 3

DESCRIPTION OF MENUS

This chapter lists the actions performed when the items on KOLAM's different menus are clicked. The inner workings of all the dialogs and widgets that pop up are explored in different chapters, and are conveniently cross-referenced from here.

FILE Menu

Open Image(s) ...

Pops open a file dialog which lets the user **select one or more image file(s) for display**. Selecting and loading a *single image* will cause it to be loaded in a single new layer and be displayed on top of the currently displayed content. Selecting and loading *multiple images* will cause each of them to be loaded in its own new layer. The layers thus created are successively overlaid, and displayed on top of the currently displayed content.

Open Sequence ...

Pops open a file dialog which lets the user **select an image sequence for display**. Once the user has navigated to the directory containing the desired image sequence, *only the first image in the sequence need be selected and loaded*; KOLAM automatically loads the whole sequence. For proper display, **KOLAM requires** that **ALL images in the directory be of the same dimensions** (width and height). KOLAM also allows **simultaneous animation** of *multiple sequences.*

Open Colormap ...

Pops open a file dialog which lets the user load a colormap (.cm) file. Colormaps are typically used to specify color *Lookup Tables* (LUT) for images that have **less than 8 bits per pixel** (bpp).

Discover PSS ...

Pops open a tool which **discovers all PSS datasets on all drives** located in the directory 'pss_pss', in turn located in the drives' root directory. Currently *implemented exclusively for the Windows platform.* This tool is the precursor to a general (platform independent) file-type based dataset discovery utility.

Preferences ...

Pops open the ***Kolam-Preferences*** tool, described in detail in Chapter 4.

Close Layer

Closes the currently active layer. Currently, **works for single image layers only**.

Close All Layers

Closes all loaded image layers. Currently, **works for single image layers only**.

Quit

Exits KOLAM. Certain user-specified preferences are *saved prior to program exit.* This procedure allows these **settings to be maintained across sessions** until modified by the user.

LAYER Menu

Grid

Toggles the display of the tile outlines on the multiple levels of a multi-resolution image.

Layer Info ...

Displays the second page of the Layer Editor tool, containing all relevant information about the current layer. The Layer Editor is described in detail in Chapter 4.

Layer Editor ...

Pops open the ***Layer Editor*** tool, which is used for layer management and manipulation in KOLAM. It is described in detail in Chapter 4.

Colormap Editor...

Pops open the ***Colormap Editor*** tool, which is used for colormap management and manipulation in KOLAM. It is described in detail in Chapter 4.

Synthetic Pyramid ...

Pops open a tool for creation of a blank pyramid image, with all the tiles but no data.

DISPLAY Menu

This menu includes **different modes** of visualizing data - when the data is *physically mapped differently* (onto a rectangular or spherical surface), when the data is *rendered differently* (scanline drawing versus texture mapping), when the camera processes a *different projection* (orthographic versus spherical) and so on.

2D Raster

Changes the current viewing mode, if different, to an *orthographic view* in which the **data is rendered in scanline fashion**, ie. data from the tiles that have been loaded into memory is rendered in a row-by-row manner for visualization. When zooming in, one notices *aliasing artifacts all over the image*, which are a defining characteristic (as well as the most important disadvantage) of scanline rendering.

2D Texture

Changes the current viewing mode, if different, to an *orthographic view* in which the **data is rendered by texture mapping**, ie. KOLAM uses the texturing facilities in the *OpenGL API* to load and render the tiles in the image. In-built as well as explicitly issued OpenGL texture interpolation directives ensure that aliasing is by-and-large eliminated by smoothing. Anti-aliasing capabilities are also enabled to further mitigate any remaining aliasing artifacts. This is KOLAM's **default viewing mode**.

3D Sphere

Changes the current viewing mode, if different, to a *3D spherical view* in which the **data is rendered by texture mapping to spherical coordinates**. Unlike the other viewing modes which utilize 2D transformations, this mode utilizes 3D translation, rotation and scaling. **KOLAM avoids Gimbal Lock** which can arise in this viewing mode by using *quaternions* in all transformations.

WINDOW Menu

Overview

Pops open the ***Overview*** tool, which enables the user to view the whole image as well as the portion of it being currently displayed. It also permits instant navigation to parts of the image that are not in close proximity to each other. The tool is further described in Chapter 4.

Cache Glyph

Pops open the ***Cache Glyph*** tool. This is a diagnostic utility which illustrates which tiles of the different levels of the loaded multi-resolution image are currently loaded in memory. It is further described in detail in Chapter 4.

TOOLS Menu

Loop

Pops open the ***Kolam-Loop*** tool, described in Chapter 5.

Track Viz

Pops open the ***Kolam-Tracker*** tool, described in Chapter 8.

Track Files

The sub-menu entries are described below:

(a) **Load a File**: Pops a File Open dialog to select a KW-18 track file for loading.

(b) **Create new File**: Creates a new KW-18 track file. Newly created tracks are added to this file.

(c) **Save to KW-18**: Pops a File Save dialog for saving the currently loaded KW-18 file as needed.

(d) **Save to CSV**: Pops a File Save dialog for saving the currently loaded KW-18 file in the CSV file format. **To save user-selected tracks in CSV**, first save them from KOLAM-Tracker as a KW-18, then load the saved KW-18 and save it as a CSV.

(e) **Delete Last**: Deletes the most recently loaded KW-18 file. All associated trajectories will no longer be displayed.

(f) **Delete All**: Deletes all KW-18 files that have been loaded. ALL trajectories will no longer be displayed.

(g) **Display Names**: Displays the names of the currently loaded KW-18 files. The default display color is green. If 'Group by Color' (below) is checked, each file name will be displayed in its group color.

(h) **Group by Color**: Displays ALL trajectories in a KW-18 file in a single color. This is useful in a number of scenarios; for example, when a comparison between ground-truth trajectories and tracker-generated trajectories (each set in separate KW-18 files) needs to be performed.

(i) **Track Toggling File**: Provides an alternative way for selecting and toggling visibilities of user selected versus unselected tracks. Allows the user to load a track toggling file (must be in .txt format, with each line containing a single, unique Object-ID), provided a KW-18 file has already been loaded.

Screen Capture

Pops open the ***Screen Capturing*** tool, described in Chapter 11.

ROI Selection

Pops open the ***Region-Of-Interest*** (ROI) Selection tool, described in Chapter 11.

Coordinate Position

Pops open the ***Coordinate Position*** tool, described in Chapter 8.

PLUGINS Menu

This section lists the plugins currently installed and active in KOLAM. The detailed description of KOLAM's plugin architecture is provided in Chapter 12.

Computer Vision

The Segmentation Relabeling plugin (under the Computer Vision menu) is currently available to users as an example of KOLAM's plugin capabilities. The detailed description of the same is provided in Chapter 12.

HELP Menu

Kolam Shortcuts

This menu item pops up a web page that lists the various hotkeys in KOLAM. A consolidated listing of all hotkeys is available in Chapter 14.

About Kolam ...

Pops open a dialog with several pieces of information about KOLAM. Key among these is the *Build number*, listed on the first line. Users must **provide the build number** when placing **requests for bug fixes**. Among the other information provided, users **must also provide the Qt version numbers** (*compile-time* and *run-time*) when requesting bug fixes.

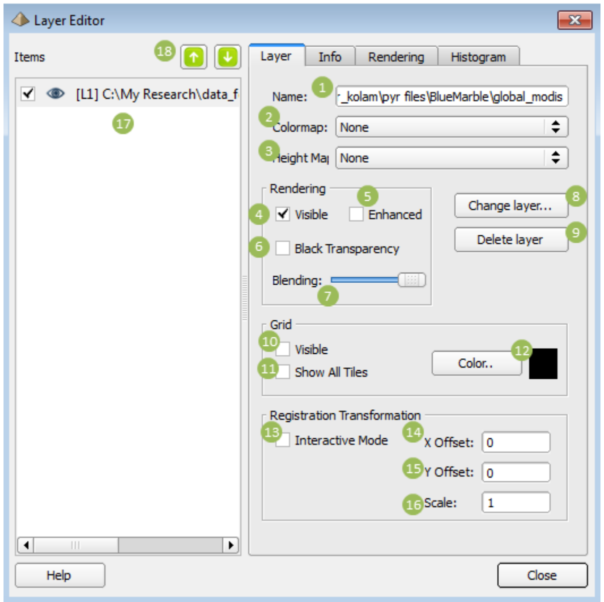
CHAPTER 4

Tools and Widgets

In this chapter, we describe KOLAM's Preferences, Discover PSS, Layer Editor, Colormap Editor, Overview and Cache Glyph tools.

Layer Editor

This tool allows the user to **manage layer properties** and **set up certain viewing and navigation options** for all loaded data layers.



Layer Editor

Items

☑ [L1] C:\My Research\data_f

Name: '_kolam\pyr files\BlueMarble\global_modis

Colormap: None

Height Ma| None

Rendering

☑ Visible ☐ Enhanced

☐ Black Transparency

Blending:

Grid

☐ Visible

☐ Show All Tiles

Registration Transformation

☐ Interactive Mode

X Offset: 0

Y Offset: 0

Scale: 1

Figure 4.1: 'Layer' page of KOLAM's Layer Editor

The following is a listing of the available functionality on the 'Layer' page of the Layer Editor (Figure 4.1, page 24).

1. Path to Dataset

Displays the full file path to the dataset loaded in the CURRENTLY ACTIVE Layer.

2. Colormap Selector

Sets the colormap for the currently active Layer.

3. Heightmap Selector

Sets the heightmap for the currently active Layer. If set, the result is most apparent in a 3D viewing mode, such as the Sphere Rendering Mode.

4. Layer Visibility

Toggles the visibility of the currently active Layer ON or OFF.

5. Layer Enhancement

Currently non-functional.

6. Layer Black Transparency

Sets the visibility of the fully black pixels in the dataset of the currently active Layer to zero, ie. those pixels are rendered fully transparent. DOES NOT WORK with image datasets (single image OR sequence) that have an inherent alpha channel (in other words, it works with RGB, but not with RGBA imagery).

7. Alpha Blending

Blends the currently active Layer against a fully black background. In other words, it progressively renders the active Layer transparent and dark.

8. Change Layer

Loads a new Layer, and makes it the currently active Layer. The display is instantly updated to reflect the change. Currently functional only if the loaded layers have image(s) of the same file type (Different file types causes display corruption).

9. Delete Layer

Deletes the currently active Layer. The display is instantly updated to reflect the change. Currently functional only for single image layers.

10. Grid Visibility

Given that the currently active Layer is a multi-resolution pyramidal image, this check box allows the user to toggle ON or OFF the grid displaying tile boundaries at the current zoom level. If the active Layer is a non-tiled image, it has only ONE tile boundary (regardless of the current zoom level), and the check box simply toggles the visibility of this single tile boundary.

11. Grid Show All Tiles

Currently non-functional.

12. Grid Color

Allows the user to set the color of the tile Grid. The latest color selection made by the user is also visible in the color pixmap to the right.

13. Interactive Mode

When checked, allows the user to apply transformations (viz. translation and scaling) to the currently active Layer ALONE: when unchecked (KOLAM's default mode of operation), ALL layers are transformed together.

14. X Offset

Sets the X-Offset of the currently active layer on the display. This setting affects only ONE layer, even if multiple layers have been loaded.

15. Y Offset

Sets the Y-Offset of the currently active layer on the display. This setting affects only ONE layer, even if multiple layers have been loaded.

16. Scale

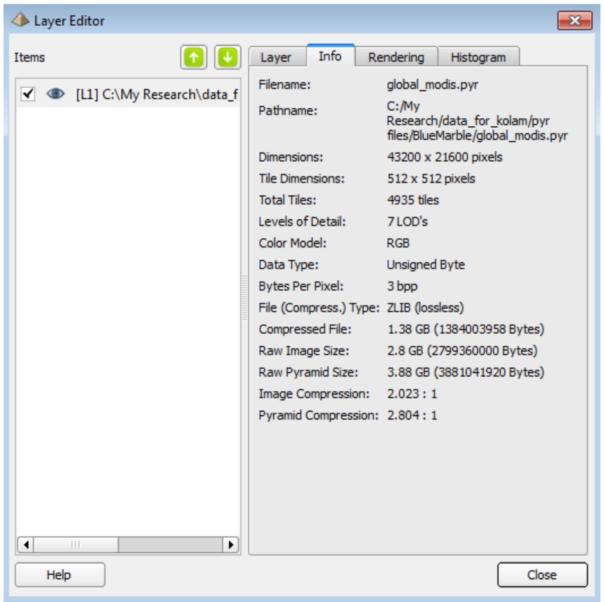
Sets the zoom level of the currently active layer on the display. This setting affects only ONE layer, even if multiple layers have been loaded.

17. Layer List

Lists the currently loaded layers. The check box provided at the beginning of each Layer name may be used to toggle the visibility of the associated Layer ON or OFF. The status of Layer visibility is also communicated to the user via the EYE icon, next to the check box (A Black icon stands for a Visible Layer, and a Grey Icon stands for an Invisible Layer).

18. Layer Move

These two buttons allow the user to MOVE the currently active Layer UP or DOWN the list of Loaded Layers. If multiple Layers are simultaneously visible, this has the effect of moving the active Layer TO THE FRONT or TO THE BACK of the 'visibility stack'.



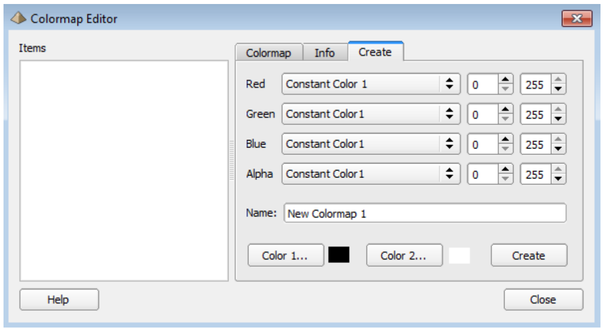
Filename:	global_modis.pyr
Pathname:	C:/My Research/data_for_kolam/pyr files/BlueMarble/global_modis.pyr
Dimensions:	43200 x 21600 pixels
Tile Dimensions:	512 x 512 pixels
Total Tiles:	4935 tiles
Levels of Detail:	7 LOD's
Color Model:	RGB
Data Type:	Unsigned Byte
Bytes Per Pixel:	3 bpp
File (Compress.) Type:	ZLIB (lossless)
Compressed File:	1.38 GB (1384003958 Bytes)
Raw Image Size:	2.8 GB (2799360000 Bytes)
Raw Pyramid Size:	3.88 GB (3881041920 Bytes)
Image Compression:	2.023 : 1
Pyramid Compression:	2.804 : 1

Figure 4.2: 'Info' page of KOLAM's Layer Editor

The 'Info' page of the Layer Editor (Figure 4.2, page 27) lists various properties of the currently active layer. These are self-explanatory and are hence not covered.

Colormap Editor

This tool allows the user to **load, create, manage and save** colormaps.



Red	Constant Color 1	0	255
Green	Constant Color 1	0	255
Blue	Constant Color 1	0	255
Alpha	Constant Color 1	0	255

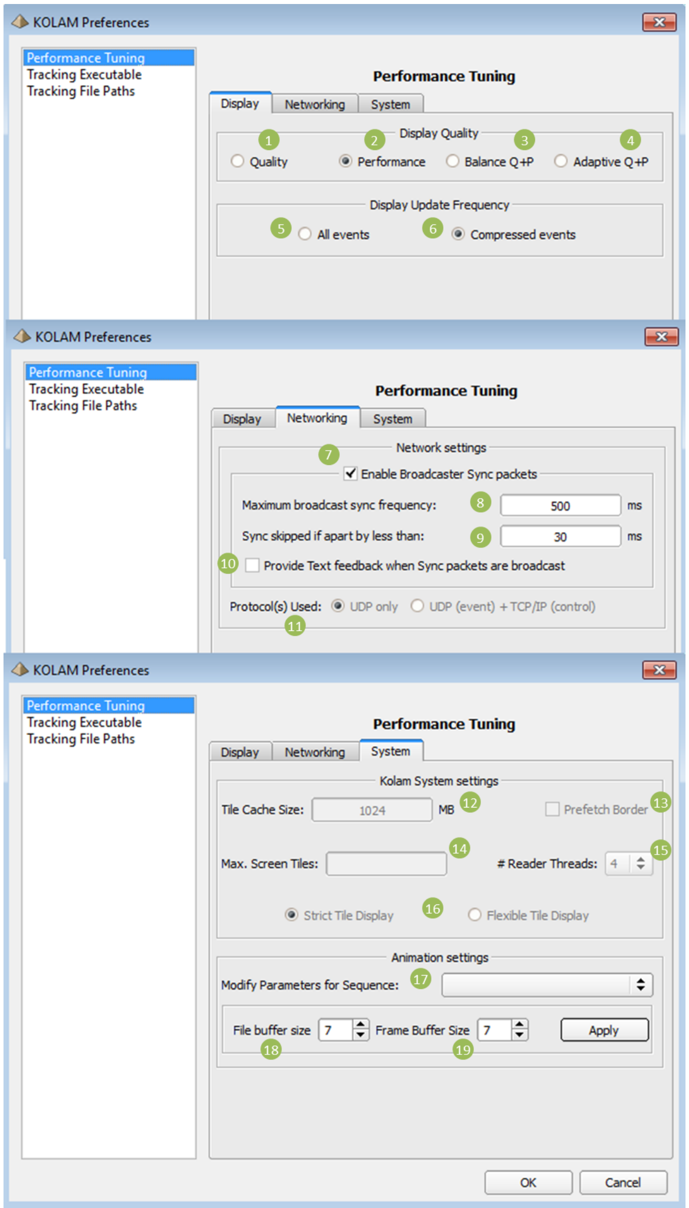
Name: New Colormap 1

Figure 4.3: KOLAM's Colormap Editor

Figure 4.3 (page 27) shows the 'Create' page of the tool. The other two pages (not shown) are the 'Colormap' page (for loading and closing colormaps) and the 'Info' page (displays information about the currently active colormap).

Preferences - *Performance Tuning*

This tool allows for user management of various system-wide KOLAM settings. The various pages of the ’**Performance Tuning**’ section are illustrated in Figure 4.4.



KOLAM Preferences

Performance Tuning
Tracking Executable
Tracking File Paths

Performance Tuning

Display | Networking | System

Display Quality

○ Quality ● Performance ○ Balance Q+P ○ Adaptive Q+P

Display Update Frequency

○ All events ● Compressed events

KOLAM Preferences

Performance Tuning
Tracking Executable
Tracking File Paths

Performance Tuning

Display | Networking | System

Network settings

☑ Enable Broadcaster Sync packets

Maximum broadcast sync frequency: 500 ms

Sync skipped if apart by less than: 30 ms

□ Provide Text feedback when Sync packets are broadcast

Protocol(s) Used: ● UDP only ○ UDP (event) + TCP/IP (control)

KOLAM Preferences

Performance Tuning
Tracking Executable
Tracking File Paths

Performance Tuning

Display | Networking | System

Kolam System settings

Tile Cache Size: 1024 MB □ Prefetch Border

Max. Screen Tiles: # Reader Threads: 4

● Strict Tile Display ○ Flexible Tile Display

Animation settings

Modify Parameters for Sequence:

File buffer size 7 Frame Buffer Size 7 Apply

OK Cancel

Figure 4.4: The Performance Tuning section of Kolam-Preferences.

1. Quality

Displays data at the best possible quality for a given zoom level. KOLAM does this by displaying the next higher level of detail while transitioning between the CURRENT zoom level and the next HIGHER zoom level. Prioritizes display quality over display interactivity; actions such as panning and animating might cause display lag on computers with older hardware. Computers with modern hardware on the other hand, should avail of the crisp display quality made possible by this mode.

2. Performance

KOLAM prioritizes user interactivity over display quality in this display mode. Done by displaying the next higher level of detail ONLY when the next HIGHER zoom level is reached, and not before. This mode of display is better suited for computers with older hardware. Additionally, this mode might be more viable over 'Quality' if the data is being animated at a high frame rate AND the user is rapidly interacting with the display.

3. Balanced Q+P

In this display mode, KOLAM attempts to create a balance between the two prior display modes, by transitioning between display qualities in a manner that averages the quality transitioning behavior of the prior two modes.

4. Adaptive Q+P

This is arguably the best choice among all the display modes; in this mode, KOLAM attempts to adapt the current display quality to the frequency of user interactivity with the display. In other words: when the frequency of user interaction increases, display quality is progressively lowered to maintain interactivity; and when this frequency drops, display quality is progressively raised. Currently non-functional.

5. All Events

KOLAM processes all user events on the display as soon as they arrive. For diagnostic purposes only; the default (better) mode is for KOLAM to process compressed events. Choosing this mode will cause KOLAM to lag frequently due to event flooding.

6. Compressed Events

The default mode of user event handling (on the display) by KOLAM. Prevents event flooding by compressing multiple events that occur in rapid succession into a single event, which is then handled by KOLAM. The best mode for event handling, and should not be changed by the user under normal circumstances.

7. Enable Broadcaster Sync Packets

[WALL MODE] Checking this box will cause the KOLAM broadcaster instance to regularly transmit synchronization packets to all KOLAM receiver instances. In other words, this forces synchronization between the broadcaster and all receivers.

8. Maximum Sync Frequency

[WALL MODE] Sets the frequency at which the KOLAM broadcaster instance sends out synchronization packets to all receivers. If set to a value less than that set in the field below, the KOLAM broadcaster instance will skip sending the synchronization packets.

9. Sync skipped if Apart by Less Than

[WALL MODE] Sets the fastest frequency at which the KOLAM broadcaster instance can synchronize all KOLAM receiver instances. Determines whether the value set in the field above will be used by KOLAM or not.

10. Provide Text Feedback

[WALL MODE] This check box toggles ON or OFF whether the user receives text feedback from KOLAM when synchronization packets are broadcast.

11. Protocol(s) Used

Currently hard-set to the UDP protocol.

12. Tile Cache Size

Currently hard-set to 1024 MB.

13. Prefetch Border

Currently disabled.

14. Max. Screen Tiles

Currently disabled.

15. # Reader Threads

Currently hard-set to 4 Threads.

16. Strict/Flexible Tile Display

Currently disabled.

17. Modify Parameters for Sequence

Selects a sequence, from ALL loaded sequences, for which the File Buffer Size and Frame Buffer Size parameters may be modified.

18. File Buffer Size

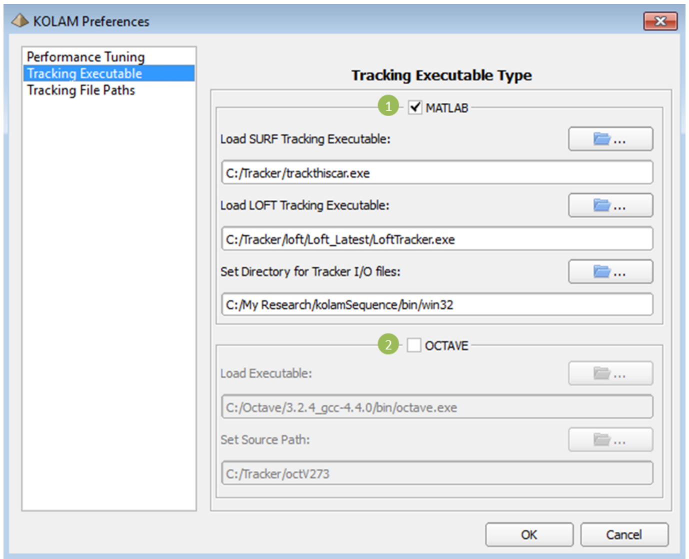
Sets the size of the file buffer for the currently selected sequence. The size of the file buffer must always be less than the number of images in the sequence, and must always be greater than the size of the lookahead buffer, the size of which is set below.

19. Frame Buffer Size

Sets the size of the lookahead (frame) buffer for the currently selected sequence. If set to a value greater than the size of the file buffer (set above), the file buffer is resized to be of the same size as the lookahead (frame) buffer.

Preferences - *Tracking Executable*

The '**Tracking Executable**' section of Kolam-Preferences is illustrated in Figure 4.5.



1 ☑ MATLAB

Load SURF Tracking Executable: C:/Tracker/trackthiscar.exe

Load LOFT Tracking Executable: C:/Tracker/loft/Loft_Latest/LoftTracker.exe

Set Directory for Tracker I/O files: C:/My Research/kolamSequence/bin/win32

2 □ OCTAVE

Load Executable: C:/Octave/3.2.4_gcc-4.4.0/bin/octave.exe

Set Source Path: C:/Tracker/octV273

Figure 4.5: The Tracking Executable section of Kolam-Preferences.

1. MATLAB

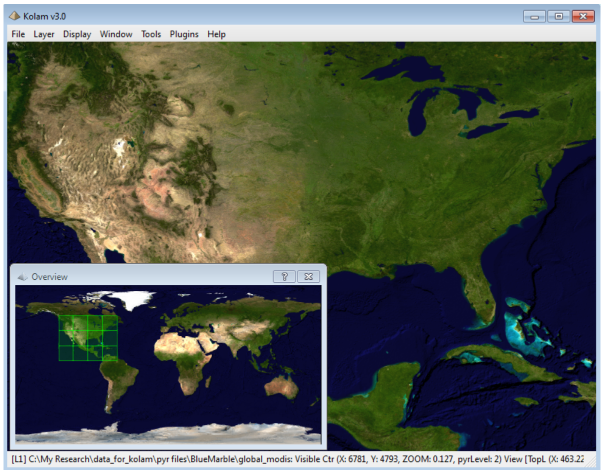
Checking this box sets MATLAB-based tracker(s), (or any non-OCTAVE based executables), as the tracker programs that KOLAM can invoke as external processes corresponding to the CSURF and LOFT trackers. The section allows the user to set the paths to the CSURF and LOFT executables, as well as the path (for *Flat Files ONLY*) to the output trajectory files.

2. OCTAVE

Checking this box sets an OCTAVE-based tracker as the external program invokable by KOLAM. Currently allows for setting the path for only *ONE tracker program AT A TIME*. The OCTAVE tool chain is not as robust as that for MATLAB; hence the user is allowed to set paths to the source code files directly, rather than to a stand-alone executable.

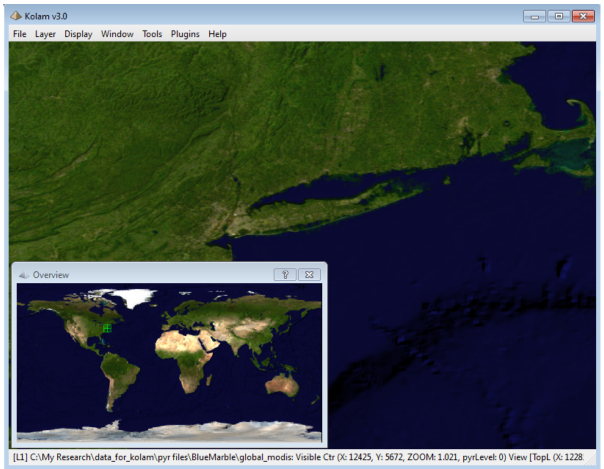
Overview

This tool provides the user with a **miniature view of the whole dataset** and allows rapid navigation to non-adjacent parts of the data. The green, semi-transparent grid indicates the sub-region of the imagery (at the current zoom level) that is actually loaded in memory and being displayed on-screen.



```
[L1] C:\My Research\data_for_kolam\pyr files\BlueMarble\global_modis: Visible Ctr (X: 6781, Y: 4793, ZOOM: 0.127, pyrLevel: 2) View [TopL (X: 463.22
```

Figure 4.6: KOLAM's overview tool. The green semi-transparent grid shows the loaded tiles for the view of the continental United States.

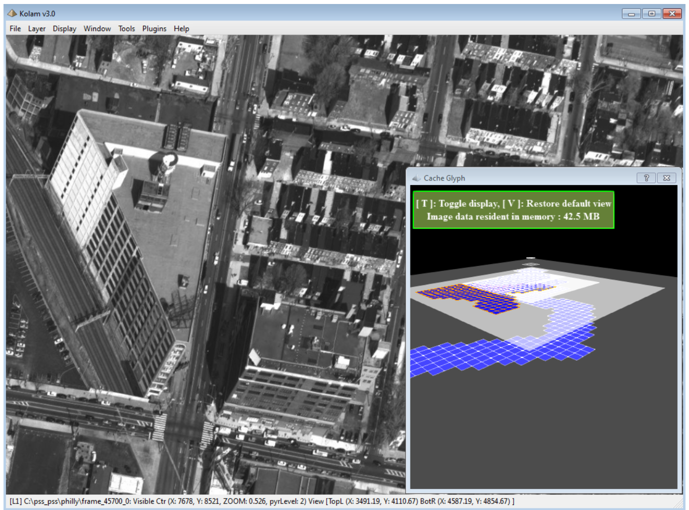


```
[L1] C:\My Research\data_for_kolam\pyr files\BlueMarble\global_modis: Visible Ctr (X: 12425, Y: 5672, ZOOM: 1.021, pyrLevel: 0) View [TopL (X: 1228
```

Figure 4.7: KOLAM's overview tool. The grid, now much smaller than in Figure 4.6, shows the loaded tiles for the zoomed-in view of New York City, Long Island and the surrounding area.

Cache Glyph

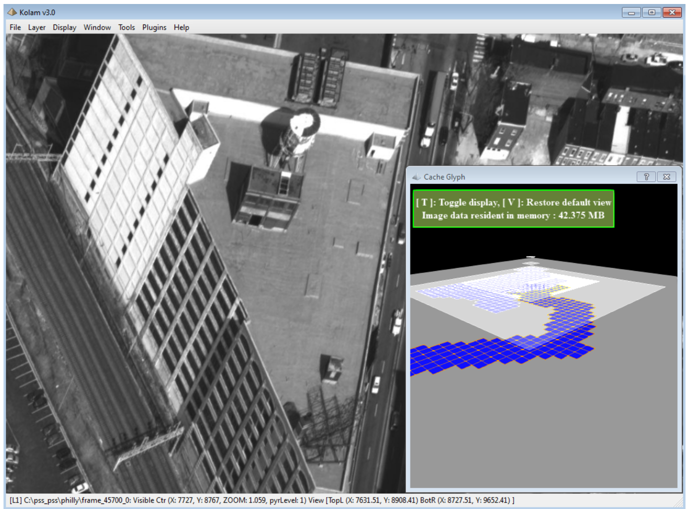
This tool interactively displays the **various residency states of image tiles in memory**. Navigating on the image instantly updates the cache glyph view: panning causes more tiles to be displayed in blue on the same plane of the multiresolution image pyramid, and zooming in or out changes the plane with the active, memory resident tiles. This is clearly demostrated in the given illustrations. In Figure 4.8, the image is zoomed to one level above the native resolution, and is then zoomed to the native resolution itself in Figure 4.9.



[T]: Toggle display, [V]: Restore default view
Image data resident in memory : 42.5 MB

```
[L1] C:\pss_pss\philly\frame_45700_0: Visible Ctr (X: 7678, Y: 8521, ZOOM: 0.526, pyrLevel: 2) View [TopL (X: 3491.19, Y: 4110.67) BotR (X: 4587.19, Y: 4854.67) ]
```

Figure 4.8: KOLAM’s cache glyph tool, with a zoomed out view.



[T]: Toggle display, [V]: Restore default view
Image data resident in memory : 42.375 MB

```
[L1] C:\pss_pss\philly\frame_45700_0: Visible Ctr (X: 7727, Y: 8767, ZOOM: 1.059, pyrLevel: 1) View [TopL (X: 7631.51, Y: 8908.41) BotR (X: 8727.51, Y: 9652.41) ]
```

Figure 4.9: KOLAM’s cache glyph tool, with a zoomed in view.

CHAPTER 5

ANIMATION

This chapter details the animation capabilities currently available in KOLAM. The user may interact with KOLAM's animation subsystem via the **Kolam-Loop** dialog, depicted in Figure 5.1. The fully expanded tool with all widgets displayed is depicted in Figure 5.2. The slightly different appearance is due to a change in stylesheets that was necessitated for maintaining compatibility with Qt 4 while continuing development with Qt 5.

Widget Description

A description of the numbered component widgets of **Kolam-Loop** and their functionality is given below.

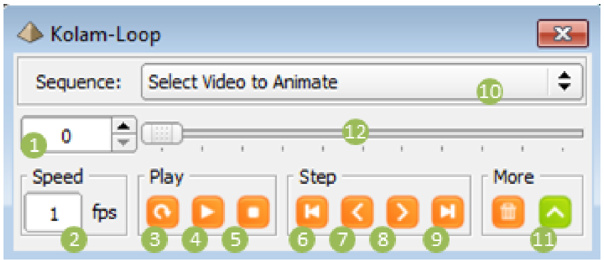
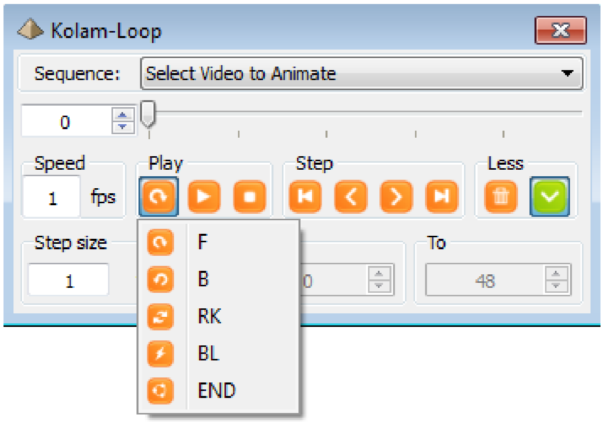


Figure 5.1: Kolam-Loop

1. Frame Counter

Indicates the current frame number of the sequence being animated. May be edited by the user (within the valid frame range) to instantaneously jump to an arbitrary frame. Editing instantly updates the Animation Scrub Bar (5).



Sequence: Select Video to Animate

0

Speed 1 fps

Step size 1

To 48

Figure 5.2: Kolam-Loop expanded to show all widgets

2. Playing speed

Animation speed in frames per second. May be edited by the user to *either speed up or slow down* the animation playing speed.

3. Play Type

This button allows the user to *choose from 5 different modes/types* of playing the animation, provided as a drop down button menu (visible only when the user clicks this button). See Figure 5.2. The buttons (from top to bottom) are as follows:

a) Loop forward: The image frames will be played *forward in time*, in an unending loop.

b) Loop backward: The image frames will be played *backward in time* in an unending loop.

c) Rock: The image frames will be '*rocked*' (ie., played forward until the end, and then backward until the beginning) in an unending loop.

d) Blink:
The animation will be continually '*blinked*' between the current frame, and either the previous frame (if the sequence was playing backward prior to blink mode being activated) or the following frame (if the sequence was playing forward prior to blink mode being activated).

e) Play without looping: The animation *will be played once and stopped*, either at the beginning of the sequence (if it was playing backward prior) or at the end of the sequence (if it was playing forward prior).

4. Play

Plays an image sequence in the mode selected by the '**Play Type**' button. This button serves to toggle between *playing* (the button is *depressed* and displays the 'Pause' icon) and *pausing* (the button is *raised* and displays the 'Play' icon) the image sequence.

5. Stop

Stops the currently playing animation. Regardless of whether the animation is playing or paused, this button *rewinds the animation* to the first frame of the sequence.

6. Skip to Start

Pauses the animation (if playing when pressed) and *skips to the first frame* of the image sequence.

7. Step Backward

Pauses the animation (if playing when pressed) and *skips to the previous frame* in the image sequence. When KOLAM's main window has focus, this functionality may also be accessed via the **Left Arrow** key on the keyboard.

8. Step Forward

Pauses the animation (if playing when pressed) and *skips to the next frame* in the image sequence. When KOLAM's main window has focus, this functionality may also be accessed via the **Right Arrow** key on the keyboard.

9. Skip to End

Pauses the animation (if playing when pressed) and *skips to the last frame* of the image sequence.

10. Dataset Name

This drop down *displays* the names of the currently loaded image sequences. **Need not be used** as KOLAM's Layer Editor provides the same functionality.

11. More Options

This button *toggles between displaying and hiding* the lower portion of the Kolam-Loop tool (normally hidden from view). The frames per step widget (first on the left) is currently non-functional. The second and third widgets (non-interactive) display the *starting and ending frame number* of the image sequence, thus providing the total number of images in the sequence at a glance.

12. Animation Scrub Bar

Dragging this slider allows the user to *navigate* to any frame in the image sequence. Instantly updates the **Frame Counter** (page 34).

CHAPTER 6

Layer Display Modalities

KOLAM's flexibility of visualization allows for multiple types of visualizations to be created, especially when multiple layers and overlays have been loaded. This chapter describes these different visualization types.

1. Display Layers overlaid one on top of the other

This is KOLAM's default display mode: newly loaded layers are overlaid on top of the layer(s) already loaded and being displayed. Each layer loaded after the first is aligned with the top left corner of the first loaded layer. An example of this type of visualization combined with other display modalities is illustrated in Figure 6.1. At present, loading a new layer causes all layers already being displayed to be realigned with the top left corner of the first layer, regardless of how the user has already moved them about the display or zoomed them. Since this is undesirable behavior, a solution is being implemented at present and will be part of the next minor software update to KOLAM.

2. Layers arranged side-by-side

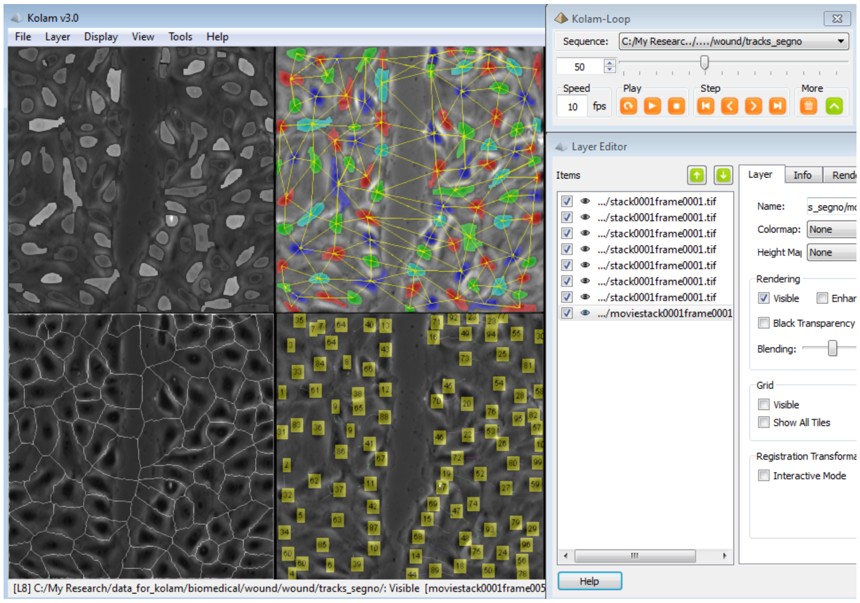
By using either KOLAM's interactive translation capability in the Layer Editor (the Interactive Mode checkbox within the Registration Transformation section, see Figure 4.1), via predefined settings in a script file, or by user mouse interaction; individual layers may be arranged in any desired non-overlapping manner in the display area. The most common mode of arrangement is a non-overlapping grid of layers. See Figure 6.1 for an example. Other possible arrangements include partial overlap between layers to correspond to a desired mosaic pattern, and arrangement (of smaller regions of interest) along the edges of the display surrounding a larger, central display of one particular layer.

3. Trajectory overlay placement

KOLAM allows for yet more complex and information-rich visualizations by allowing for multiple overlays containing trajectories or any other supported annotations to be overlaid on the loaded layers. See Figure 6.1. All overlays are currently associated with the active layer only. If multiple sequence layers are loaded, overlay placement may be transferred to the desired layer by making that layer active; which may be done by clicking on the respective layer name in the layer listing on the left hand side of the Layer Editor (see Figure 4.1, page 24).

4. A combination of all of the above

This case represents the most general and versatile case of KOLAM's 2D+t visualization and animation capabilities. Multiple layers may be loaded, of which one or more layers may have their transparencies altered. These layers may overlap to varying degrees with each other and may be zoomed in on at different resolution levels. In addition to this, one or more overlays with trajectory data or any other type of annotation may be overlaid on the layers. Figure 6.1 is an example of such a visualization.



Sequence: C:/My Researc../..../wound/tracks_segno

50

Speed 10 fps

☑ .../stack0001frame0001.tif
☑ .../stack0001frame0001.tif
☑ .../stack0001frame0001.tif
☑ .../stack0001frame0001.tif
☑ .../stack0001frame0001.tif
☑ .../stack0001frame0001.tif
☑ .../stack0001frame0001.tif
☑ .../moviestack0001frame0001

Name: s_segno/mo
Colormap: None
Height Ma| None

Rendering
☑ Visible □ Enhar
□ Black Transparency
Blending:

Grid
□ Visible
□ Show All Tiles

Registration Transforma
□ Interactive Mode

[L8] C:/My Research/data_for_kolam/biomedical/wound/wound/tracks_segno/: Visible [moviestack0001frame005

Figure 6.1: KOLAM's screen capturing feature, with multiple layers and overlays. Eight sequences (4 copies of the wound healing dataset & 4 mask layers showing different results, from top left to bottom right: segmentation masks, 4-colored graph-based segmentation results, Voronoi diagram, and cell motility trajectories with ID labels; transparently overlaid on the dataset copies) have been organized in a grid pattern for comparative visualization. KOLAM can also capture this grid visualization animation and output the captures utilizing user specified location, format and naming scheme.

CHAPTER 7

SPHERE & TERRAIN RENDERING

Sphere Rendering

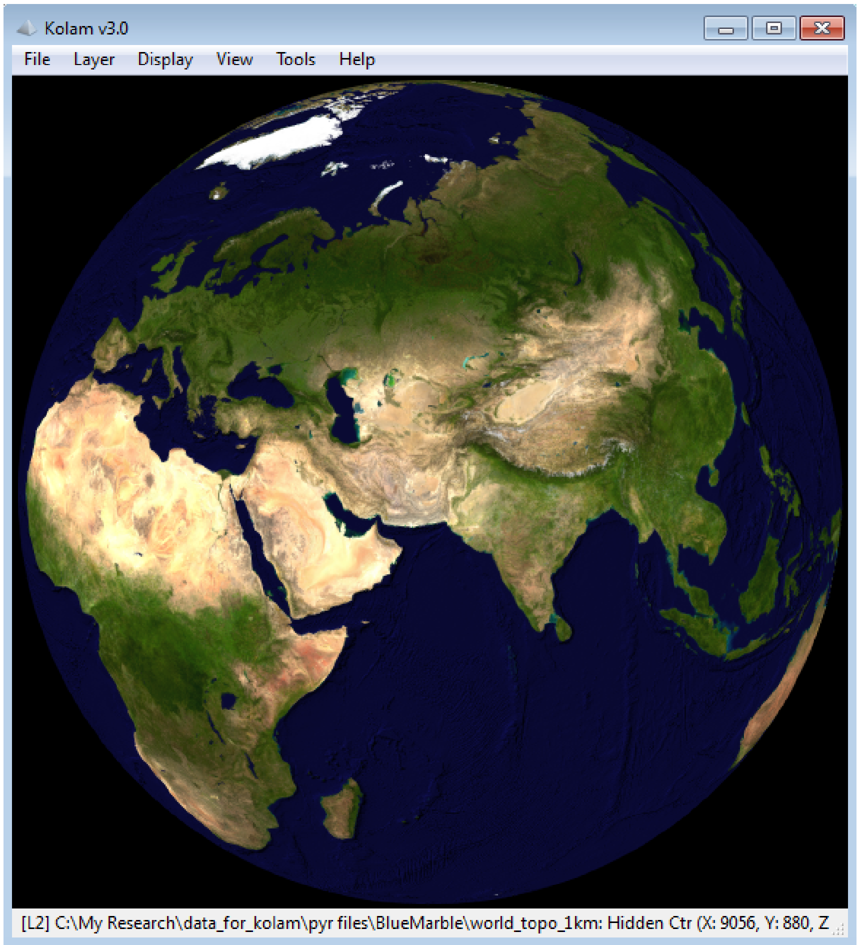
KOLAM's sphere viewing and rendering system (Eg., Figure 7.1) projects the loaded 2D image data onto a spherical shape which is approximated by a triangular mesh. Vertices on the sphere's surface are computed for each tile using that tile's (x,y) image plane offsets, as well the tile's (u,v) latitude-longitude pair. The prohibitive time and space requirements for performing the required trigonometric computations are ameliorated by precomputing the individual components and combining them in the final formula.

The data used for both the sphere and terrain rendering visualizations (Figure 7.1, Figure 7.2 and Figure 7.3) is from NASA's MODIS satellite imagery dataset.

Terrain Rendering

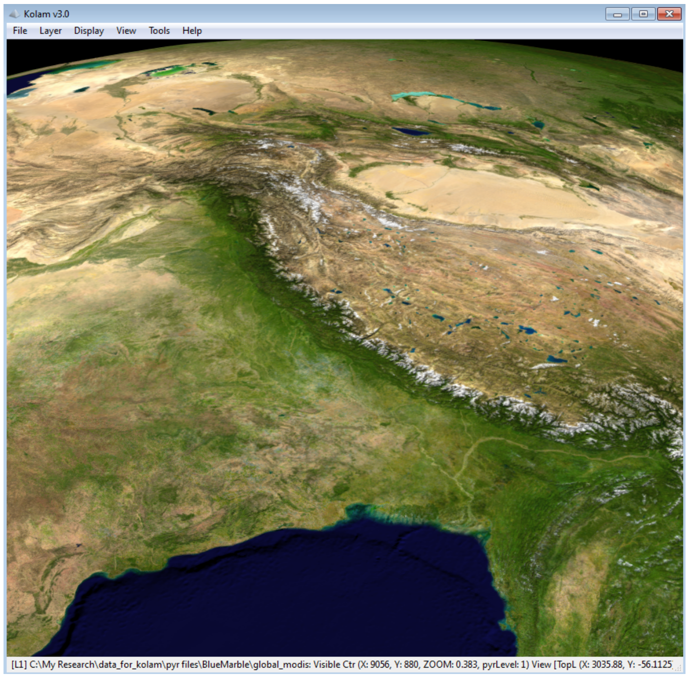
KOLAM's terrain rendering system is an extension of its texture rendering functionality, and generates additional geometry to represent the height values associated with the textured image loaded in the layer. Height map (or elevation map) images are typically 8-bit or 16-bit grayscale images. Acceptable frame rates when rendering these height maps are achieved by simplifying the generated geometry, which KOLAM currently performs by uniformly subsampling the input height samples.

An example visualization of the terrain rendering in action in provided below, as a pair of illustrations. Figure 7.2 shows a zoomed in view of the Himalayas without terrain rendering, and Figure 7.3 shows exactly the same view, but with terrain rendering enabled.



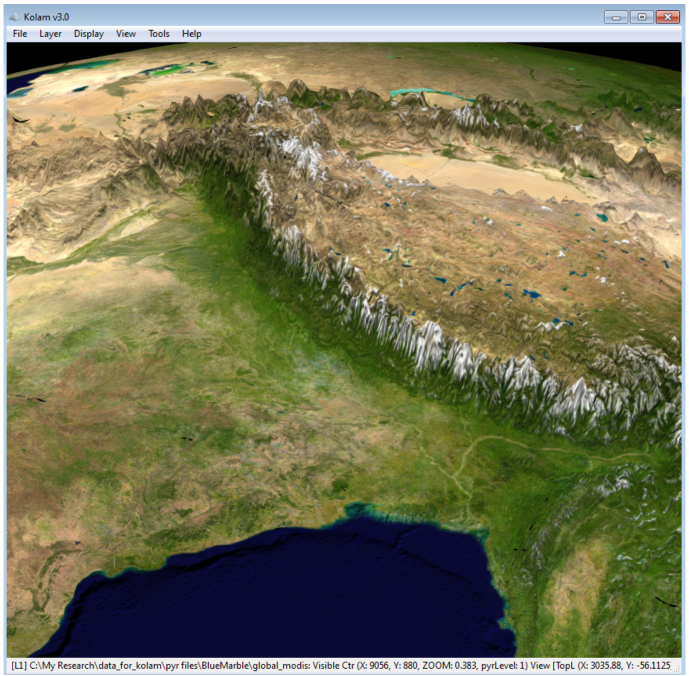
[L2] C:\My Research\data_for_kolam\pyr files\BlueMarble\world_topo_1km: Hidden Ctr (X: 9056, Y: 880, Z

Figure 7.1: Sphere viewing and rendering in KOLAM.



[L1] C:\My Research\data_for_kolam\pyr files\BlueMarble\global_modis: Visible Ctr (X: 9056, Y: 880, ZOOM: 0.383, pyrLevel: 1) View [TopL (X: 3035.88, Y: -56.1125

Figure 7.2: A zoomed-in view of the Himalayas; Terrain Rendering - turned OFF.



[L1] C:\My Research\data_for_kolam\pyr files\BlueMarble\global_modis: Visible Ctr (X: 9056, Y: 880, ZOOM: 0.383, pyrLevel: 1) View [TopL (X: 3035.88, Y: -56.1125

Figure 7.3: A zoomed-in view of the Himalayas; Terrain Rendering - turned ON.

CHAPTER 8

TRACKING

This chapter presents KOLAM's tracking modes and capabilities. We first introduce the Kolam-Tracker dialog and give its widget description. The remainder of the chapter then details the operation of KOLAM's tracking modes and their applicability to multiple domains.

Page 1: 'WAMI'

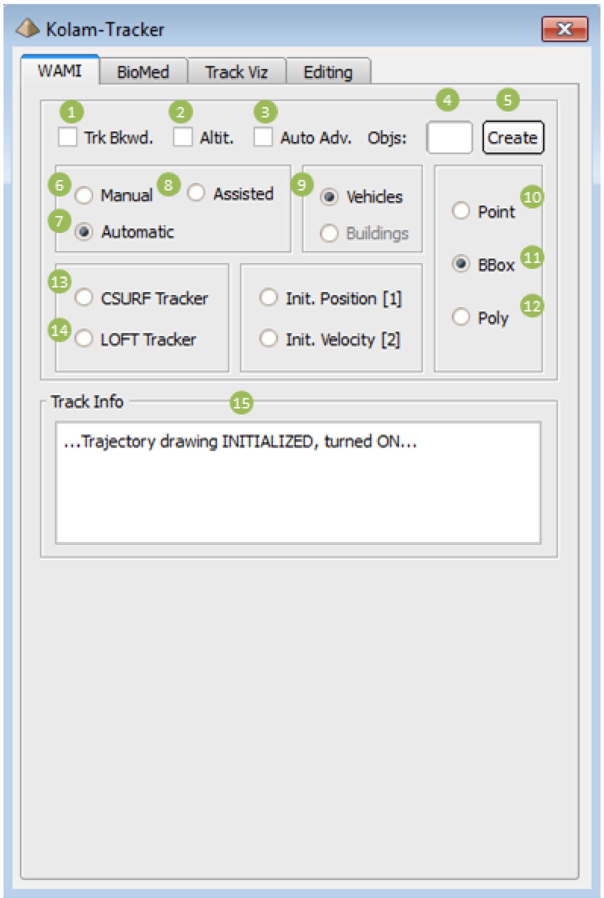
Given below is the description of the widgets on the **WAMI** page of Kolam-Tracker.

1. Track Backward toggle

This check box toggles whether or not the automated, manual or assisted tracking modes perform their specific sequential operations (tracking, ground truthing, combination of the two) backward in time. The check box is disabled by default, ie., all tracking modes operate forward in time. KOLAM *does not* currently possess any backward tracking capability; these will be incorporated as the specific needs arise.

2. (Altitude) Parameter File toggle

This check box is specifically relevant to the automated tracking module. It toggles whether or not the selected automatic tracker (CSURF or LOFT) will use an extra parameter file (the location of the same being specified by KOLAM). Though the widget name is short for Altitude (as it was originally intended to toggle ON or OFF an altitude parameter file for the chosen automatic tracker to use when tracking on a dataset with a significantly different flight altitude), it informs the tracker to use any given parameter file, with any required purpose.



□ Trk Bkwd.
□ Altit.
□ Auto Adv.
Objs:
○ Manual
● Automatic
○ Assisted
● Vehicles
○ Buildings
○ Point
● BBox
○ Poly
○ CSURF Tracker
○ LOFT Tracker
○ Init. Position [1]
○ Init. Velocity [2]

...Trajectory drawing INITIALIZED, turned ON...

Figure 8.1: WAMI page of Kolam-Tracker

3. Auto Advance toggle

This check box is specifically relevant to the manual tracking (ground truthing) module, and toggles whether or not KOLAM auto-advances to the next frame in the image sequence once the user completes creating ground truth for the current frame. The type of ground truth typically comprises only point primitives. This is because either the bounding box or polygon primitives have a significantly greater probability of needing to be corrected; meaning that there exists no uniform means for KOLAM to determine when to advance to the next frame.

4. Object ID display

Displays the most recently created (or loaded) Object ID. Each Object ID represents either a tracked or potentially trackable object or image feature that has been selected on the currently displayed image.

5. Create new Object ID

This button allows the user to create new Object IDs, either individually (pressing the button) or several at once (accessing the right-click context menu and typing in the desired number in the data entry widget that appears).

6. Manual Tracking toggle button

This radio button toggles KOLAM's manual tracking (ground truthing) mode ON. The manual, automatic and assisted tracking radio buttons are functionally exclusive, ie. enabling one disables all others. The details are presented under Manual Tracking.

7. Automatic Tracking toggle button

This radio button toggles KOLAM's automatic tracking (external tracker invocation) mode ON. The details are presented under Automatic Tracking.

8. Assisted Tracking toggle button

This radio button toggles KOLAM's assisted tracking (combination of auto & manual tracking functionality) mode ON. The details are presented under Assisted Tracking.

9. Vehicle toggle button

This radio button is specifically relevant to the automated tracking module, and toggles the type of object being tracked. It is currently non-functional, since KOLAM defers the process of distinguishing between trackable object types to the individual external tracking programs.

10. Point primitive toggle button

This radio button, alongwith the Bounding Box and Polygon radio buttons, toggles ON the specific type of primitive that will be drawn for each time step on any given track. The default primitive for all tracking modes (Automatic Tracking, Manual Tracking or Assisted Tracking) is the Point (more accurately referred to as the centroid). Any trajectory is represented by a number of centroids pairwise joined (between successive time steps) by line segments.

11. Bounding Box primitive toggle button

This radio button toggles ON the drawing of **bounding boxes** for each time step on all visible trajectories, provided that the loaded trajectory information has valid bounding boxes specified for this time step. The centroids of the bounding boxes are computed from their corners. The trajectory is then formed by the pairwise joining of these centroids, in exactly the same manner as trajectories are formed from point primitives (page 44). When performing Manual Tracking, the user is given as many tries as needed to draw the best desired bounding box; and needs to signal completion of the task to KOLAM via explicit input (in this case, by pressing the 'ENTER' key). In other words, there is no Auto-Advance capability defined for the drawing of bounding boxes.

12. Polygon primitive toggle button

This radio button toggles ON the drawing of **polygons** for each time step on all visible trajectories, provided that the loaded trajectory information has valid polygons specified for this time step. The centroids of the polygons are computed from their vertices. While drawing a polygon for a given time step, users are presented with a context menu (activatable within the polygon boundary) which allows for a multitude of polygon editing operations, these are: **add** vertex, **move** vertex, **delete** vertex, **translate**, **rotate**, **scale**, **duplicate** polygon along this trajectory and **replicate** polygon to other trajectories.

13. CSURF Tracker toggle button

This radio button toggles ON the usage of the CSURF tracker in the Automatic Tracking mode, further described on page 54.

14. LOFT Tracker toggle button

This radio button toggles ON the usage of the LOFT tracker in the Automatic Tracking mode, further described on page 54.

15. Trajectory operation feedback

This text box provides feedback to the user when tracking activities are performed, and in some occasions, the results of performing said activities.

Page 2: ’BioMed’

Given below is the description of the widgets on the **BioMed** page of Kolam-Tracker. The section only details the unique widgets under ’Available Algorithms’ and ’File Settings’; all other widgets have the same functionality as their counterparts described under Page 1: ’WAMI’.

1. Select Algorithm

This check box toggles whether or not the automated, manual or assisted tracking modes perform their specific sequential operations (tracking, ground truthing, combination of the two) backward in time. The check box is disabled by default, ie., all tracking modes operate forward in time. KOLAM *does not* currently possess any backward tracking capability; these will be incorporated as the specific needs arise.

2. Number of Inputs

This check box is specifically relevant to the automated tracking module. It toggles whether or not the selected automatic tracker (CSURF or LOFT) will use an extra parameter file (the location of the same being specified by KOLAM). Though the widget name is short for Altitude (as it was originally intended to toggle ON or OFF an altitude parameter file for the chosen automatic tracker to use when tracking on a dataset with a significantly different flight altitude), it informs the tracker to use any given parameter file, with any required purpose.

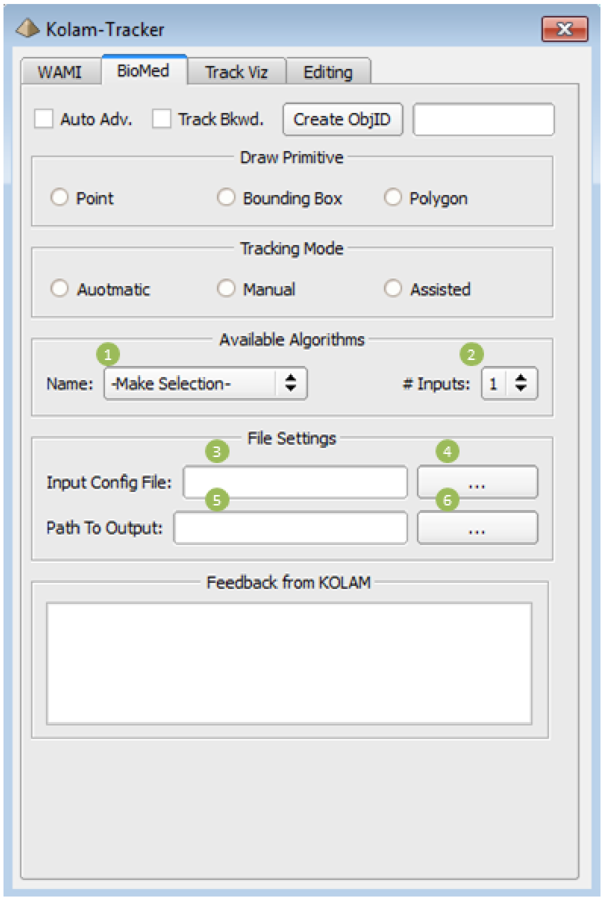


Figure 8.2: BioMed page of Kolam-Tracker

3. Input Parameter file path

This check box is specifically relevant to the manual tracking (ground truthing) module, and toggles whether or not KOLAM auto-advances to the next frame in the image sequence once the user completes creating ground truth for the current frame. The type of ground truth typically comprises only point primitives. This is because either the bounding box or polygon primitives have a significantly greater probability of needing to be corrected; meaning that there exists no uniform means for KOLAM to determine when to advance to the next frame.

4. Input Parameter File selector

Displays the most recently created (or loaded) Object ID. Each Object ID represents either a tracked or potentially trackable object or image feature that has been selected on the currently displayed image.

5. Output file path

This button allows the user to create new Object IDs, either individually (pressing the button) or several at once (accessing the right-click context menu and typing in the desired number in the data entry widget that appears).

6. Output file path selector

This radio button toggles KOLAM's manual tracking (ground truthing) mode ON. The manual, automatic and assisted tracking radio buttons are functionally exclusive, ie. enabling one disables all others. The details are presented under Manual Tracking.

Page 3: 'Track Viz'

Given below is the description of the widgets on the **Track Viz** page of Kolam-Tracker (Figure 8.3).

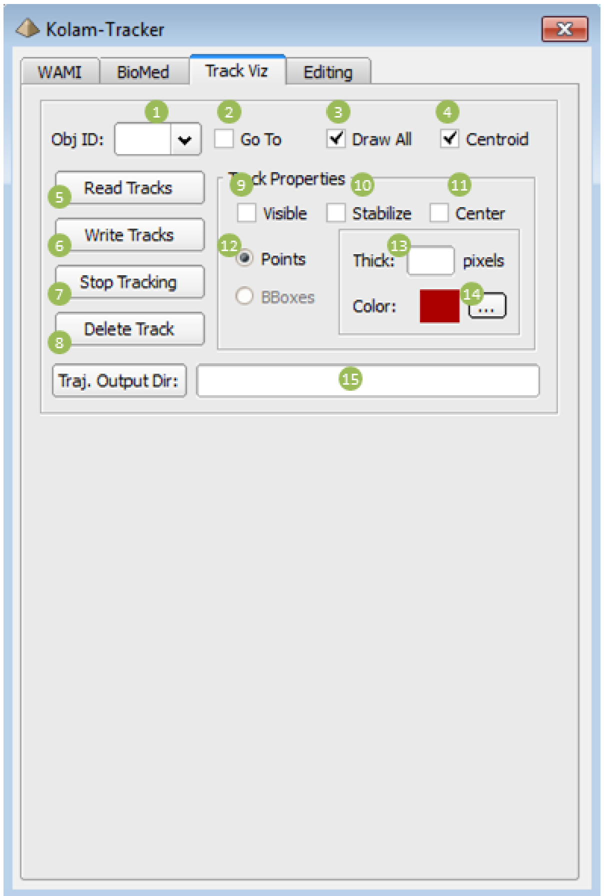


Figure 8.3: Track Viz page of Kolam-Tracker

1. Object ID Selector

This drop-down selects one of all loaded trajectories to be currently active. This trajectory is hereafter referred to as either the **current trajectory** or the **current Object-ID**.

2. Go To frame toggle

Currently non-functional.

3. Draw All Tracks toggle

This check box **toggles the display of ALL trajectories**.

4. Centroid display toggle

This check box **toggles the display of centroids** for each time step, for ALL trajectories.

5. Read Trajectories from Archive

This button loads trajectories from a user-created archive, and depending on the current tracking mode, places them in either the **Tracker_IO_Kolam/auto** or the **Tracker_IO_Kolam/manual** directory (*works to flat files only*).

6. Write Trajectories to Archive

Pops open a directory selector, so that the user may archives the trajectory files for ALL trajectories (*works to flat files only, depending on the current tracking mode, trajectories under either the **Tracker_IO_Kolam/auto** or the **Tracker_IO_Kolam/manual** directory are archived*).

7. Stop the Selected Running Tracker

When pressed, this button **halts the execution** of the invoked external tracker executable. **If multiple executables** are running, it *halts the programs in the order that they were initiated*. Thus, it is relevant exclusively to the Automatic Tracking mode.

8. Delete Trajectory files for Object ID

When pressed, this button **deletes** the trajectory files associated with the current Object-ID (*works to flat files only*).

9. Trajectory Visibility toggle

This check box toggles the **visibility** of the *currently selected trajectory*.

10. Stabilized Trajectory display toggle

This check box **toggles** between displaying only the **unstabilized trajectory**, displaying only the **stabilized trajectory** and displaying **both**; for the *currently selected Object-ID*. Such trajectory pairs are a unique by-product of trackers invoked in the Automatic Tracking mode.

11. Center screen on tracked object toggle

This check box **toggles centering the on-screen display** on the primitive (point, bbox or polygon) of the *currently selected trajectory*, for the current time step. Proper effect is seen when the image sequence is animated.

12. Draw primitive toggle

This radio button is **currently non-functional**.

13. Set Trajectory thickness

Allows the user to **alter trajectory thickness** by entering integer values greater than or equal to 1. This setting applies to *ALL trajectories.*

14. Set Trajectory color

Provides the user access to a color palette to **modify the current color** of the trajectory. This is a *per-trajectory* setting.

15. Current Trajectory output directory

Displays the **current output directory** for trajectories (*works for flat files only*).

Page 4: ’Editing’

Given below is the description of the widgets on the **Editing** page of Kolam-Tracker (Figure 8.4).

1. Display All Trajectories

This button toggles whether KOLAM **displays the full trajectories at ALL TIME STEPS**, *regardless of when the individual trajectories start or end.* This is in contrast to the traditional manner of trajectory display: a given trajectory is *not visible until* the time step on which it starts, its *visibility then transitions* from partial towards complete, and it is *once again not visible past* its ending time step. This type of display is uniquely useful when performing editing operations which alter the trajectory length in some fashion (since visibility of the trajectory will remain unchanged even if editing causes it to end before or start after the current time step).

2. Turn OFF unselected trajectories

This check box **toggles the visibility** of those trajectories **not selected** by the user.

3. Select Trajectories toggle

This check box **toggles whether right-clicking on a centroid of any trajectory (upto and including the current time step)** causes that *trajectory to be selected.* User-selected trajectories are **treated as a special subset** of all trajectories in the current KW-18; a small set of additional operations may be performed on them, and these are described below.

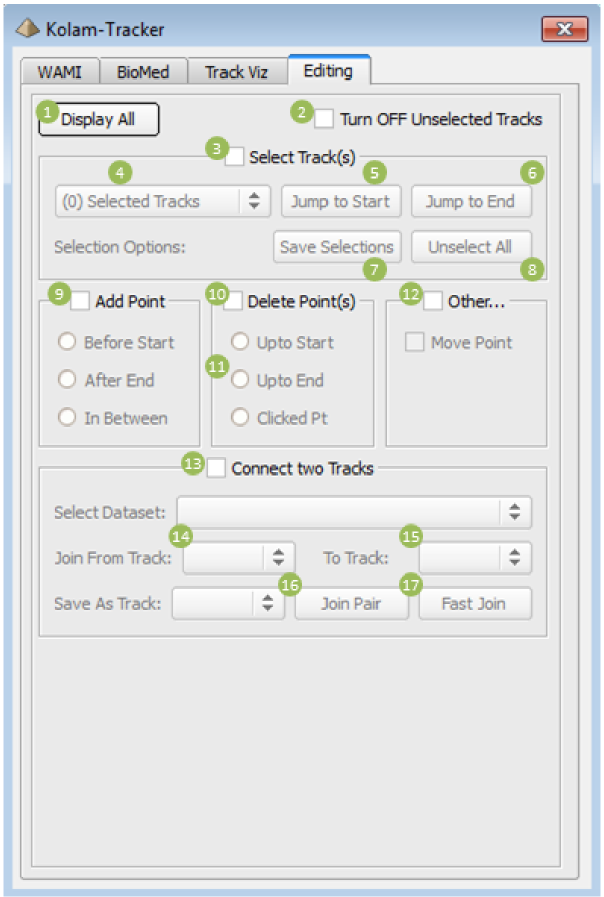


Figure 8.4: Editing page of Kolam-Tracker

4. Selected Trajectory list

Lists the **subset of currently selected trajectories**. The list is *dynamically updated* as trajectories are newly selected, created or merged together. **Selecting a trajectory** in this list *marks it as 'active'*, meaning that all potential editing operations will be performed on it only. If an active trajectory is unselected, *the most recent trajectory to be added to the list is marked as being active.*

5. Jump to Start

Jump to the **starting time step** of the *currently active trajectory*. Has no effect if no trajectory is currently active.

6. Jump to End

Jump to the **ending time step** of the *currently active trajectory*. Has no effect if no trajectory is currently active.

7. Save Selections

Pops open a file dialog to save all selected trajectories to a KW-18 file.

8. Unselect All

Unselect all selected trajectories.

9. Add Point(s) to Trajectory End(s)

This check box toggles setting ’ADD Point’ as the current trajectory editing operation. Note that **checking this box will lock the user** into ADD-ing to either end of the currently active trajectory, thus **preventing selection of any other trajectories**. Unchecking the box (and those of any other editing operations) will re-enable trajectory selection. ADD is explored in detail in Chapter 10.

10. Delete Point(s) from Trajectory End(s)

This check box toggles setting ’DELETE Point’ as the current trajectory editing operation. Note that **checking this box will lock the user** into DELETE-ing from either end of the currently active trajectory, thus **preventing selection of any other trajectories**. Unchecking the box (and those of any other editing operations) will re-enable trajectory selection. DELETE is explored in detail in Chapter 10.

11. Delete clicked point / Split Track

This check box toggles setting ’SPLIT Track’ as the current trajectory editing operation. Note that **checking this box will lock the user** into SPLIT-ing the currently active trajectory, thus **preventing selection of any other trajectories**. Unchecking the box (and those of any other editing operations) will re-enable trajectory selection. SPLIT is explored in detail in Chapter 10.

12. Move Point(s) on Trajectory

This check box toggles setting ’MOVE Point’ as the current trajectory editing operation. Note that **checking this box will lock the user** into MOVE-ing point(s) on the currently active trajectory, thus **preventing selection of any other trajectories**. Unchecking the box (and those of any other editing operations) will re-enable trajectory selection. MOVE is explored in detail in Chapter 10.

13. Connect Trajectories

This check box toggles setting ’CONNECT Trajectories’ as the current trajectory editing operation. Note that **checking this box will lock the user** into CONNECT-ing trajectories, thus **preventing selection of any other trajectories**. Unchecking the box (and those of any other editing operations) will re-enable trajectory selection. CONNECT is explored in detail in Chapter 10.

14. ’Join From’ Trajectory

This drop-down (relevant to *pairwise trajectory joining ONLY*) lists all selected trajectories, and lets the user **select a SOURCE trajectory** (the ’*source*’ is the trajectory from which points are removed in the event of temporal overlap between itself and the '*target*'). The role played by this drop-down is explored in detail in Chapter 10.

15. 'Join To' Trajectory

This drop-down (relevant to *pairwise trajectory joining ONLY*) lists all selected trajectories, and lets the user **select a TARGET trajectory** (the '*target*' is the trajectory which is completely retained in the event of temporal overlap between itself and the '*source*'). The role played by this drop-down is explored in detail in Chapter 10.

16. Join Trajectory pair

Pressing this button **JOINs the pair of trajectories** that have been selected via the drop-downs described above. Has no effect if either one or both trajectories have not been selected in the drop-downs. The PAIRWISE JOIN operation is explored in detail in Chapter 10.

17. Multiple Trajectory Join

Pressing this button **JOINs ALL selected trajectories**. Has no effect if none have been selected. The FAST JOIN operation is explored in detail in Chapter 10.

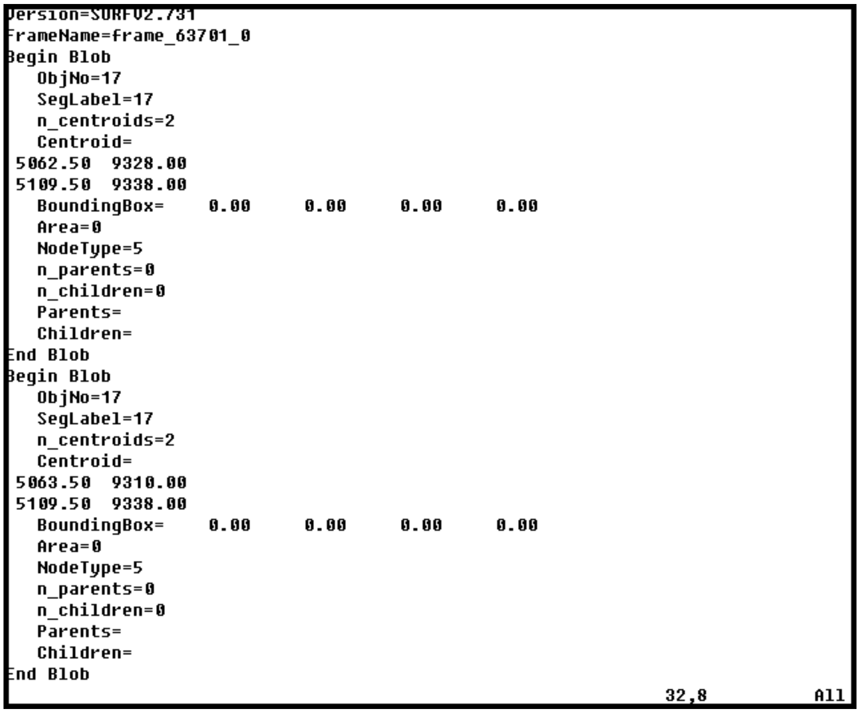
Track file formats supported by KOLAM

KOLAM provides support for two track file formats - an in-house designed *flat file format*, and the *KW-18 file format*, developed by Kitware Inc and adopted at CIVA lab for collaborative data sharing efforts. The specifications for both formats are presented below. Users desiring to have their programs output track files in either format for display in KOLAM are advised to **strictly adhere to the format details** presented in the *illustrations* and *descriptions.*

Flat file format

The flat file format is an **unstructured** (vis-a-vis structured formats like XML) file format (file extension - **.txt**) that was **designed specifically to handle the unique challenges** presented by certain wide-area datasets (extreme parallax, poor registration, wrong elevation model, inferior IMU etc.). For every trajectory, *each time step is represented by a single file*, that contains the centroids on the trajectory upto that particular time step. In other words: if a trajectory has a length of 100 time steps, the flat file system associates 100 files with this ***single*** trajectory. Unfortunately, such a data management system is the *only means* of addressing a scenario in which *all centroids on the trajectory are different for every time step* (the non-registered nature of the data necessitates that the trajectories themselves be registered). The structurs of KOLAM's flat files is illustrated in Figure 8.5.

All lines in a flat file that lie with a **'Begin Blob' - 'End Blob'** block correspond to information pertinent to the Object-ID indicated by the **ObjNo** field, for *this particular time step*. Thus, the flat file illustrated in Figure 8.5 contains data for two trajectories (since there are **2 Blobs**) corresponding to Object-ID '17', for its 2nd time step (indicated by the number of (*x*,*y*) pairs under the **Centroid** field). While all fields after the 'BoundingBox' field must be written out while creating the flat



```
Version=SURFV2.731
FrameName=frame_63701_0
Begin Blob
   ObjNo=17
   SegLabel=17
   n_centroids=2
   Centroid=
 5062.50  9328.00
 5109.50  9338.00
   BoundingBox=    0.00     0.00     0.00     0.00
   Area=0
   NodeType=5
   n_parents=0
   n_children=0
   Parents=
   Children=
End Blob
Begin Blob
   ObjNo=17
   SegLabel=17
   n_centroids=2
   Centroid=
 5063.50  9310.00
 5109.50  9338.00
   BoundingBox=    0.00     0.00     0.00     0.00
   Area=0
   NodeType=5
   n_parents=0
   n_children=0
   Parents=
   Children=
End Blob
                                                                32,8          All
```

Figure 8.5: Screen shot of a typical KOLAM flat file, for *one time step on a trajectory*.

track file(s), these fields merely have a nominal presence in the file. They are not currently used by KOLAM for drawing trajectories.

KW-18 file format

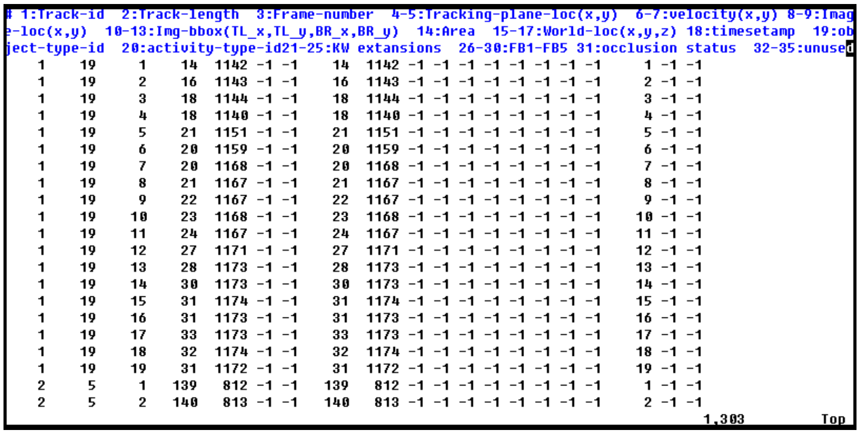
The KW-18 file format is a **structured** file format (file extension - **.kw18**). Its data organization scheme allows multiple trajectories representing a complete tracking database to be stored in a single file. A KW-18 file is partly illustrated in Figure 8.6.

Each trajectory in a KW-18 file is represented by a *contiguous group* of **lines** arranged in *increasing order* of **time**. Such trajectories are then arranged in *increasing order* of **Track-ID**. Each line represents a single time step on the trajectory, and stores multiple pieces of information about that single time step in a *space-separated* fashion. All **20 data columns** (see Figure 8.6) are **mandatory**.

Examining Figure 8.6, additional details become apparent. First, fields (**4,5**) and (**8,9**) are identical, and field (**18**) is identical to field (**3**). Second, *ALL* fields for which data are unavailable are populated with the placeholder '**-1**'.

Tracking Modes in KOLAM

KOLAM has three tracking modes: **Automatic Tracking**, **Manual Tracking** and **Assisted Tracking**. Trajectory editing is covered in Chapter 10.



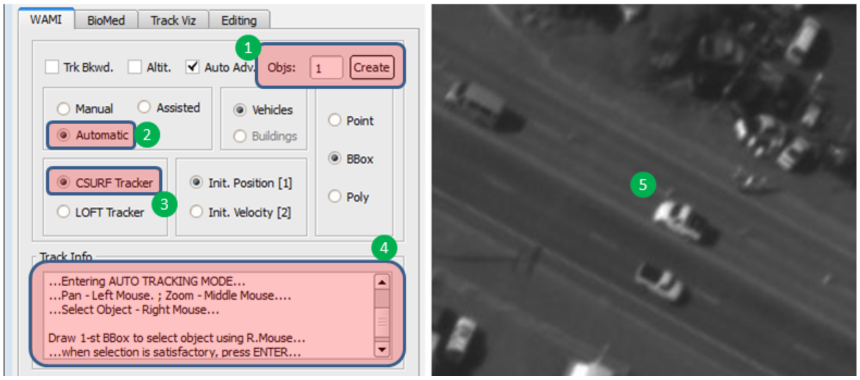
```
# 1:Track-id  2:Track-length  3:Frame-number  4-5:Tracking-plane-loc(x,y)  6-7:velocity(x,y) 8-9:Imag
e-loc(x,y)  10-13:Img-bbox(TL_x,TL_y,BR_x,BR_y)  14:Area  15-17:World-loc(x,y,z) 18:timesetamp  19:ob
ject-type-id  20:activity-type-id21-25:KW extansions  26-30:FB1-FB5 31:occlusion status  32-35:unused
     1      19       1     14  1142 -1 -1     14  1142 -1 -1 -1 -1 -1 -1 -1 -1      1 -1 -1
     1      19       2     16  1143 -1 -1     16  1143 -1 -1 -1 -1 -1 -1 -1 -1      2 -1 -1
     1      19       3     18  1144 -1 -1     18  1144 -1 -1 -1 -1 -1 -1 -1 -1      3 -1 -1
     1      19       4     18  1140 -1 -1     18  1140 -1 -1 -1 -1 -1 -1 -1 -1      4 -1 -1
     1      19       5     21  1151 -1 -1     21  1151 -1 -1 -1 -1 -1 -1 -1 -1      5 -1 -1
     1      19       6     20  1159 -1 -1     20  1159 -1 -1 -1 -1 -1 -1 -1 -1      6 -1 -1
     1      19       7     20  1168 -1 -1     20  1168 -1 -1 -1 -1 -1 -1 -1 -1      7 -1 -1
     1      19       8     21  1167 -1 -1     21  1167 -1 -1 -1 -1 -1 -1 -1 -1      8 -1 -1
     1      19       9     22  1167 -1 -1     22  1167 -1 -1 -1 -1 -1 -1 -1 -1      9 -1 -1
     1      19      10     23  1168 -1 -1     23  1168 -1 -1 -1 -1 -1 -1 -1 -1     10 -1 -1
     1      19      11     24  1167 -1 -1     24  1167 -1 -1 -1 -1 -1 -1 -1 -1     11 -1 -1
     1      19      12     27  1171 -1 -1     27  1171 -1 -1 -1 -1 -1 -1 -1 -1     12 -1 -1
     1      19      13     28  1173 -1 -1     28  1173 -1 -1 -1 -1 -1 -1 -1 -1     13 -1 -1
     1      19      14     30  1173 -1 -1     30  1173 -1 -1 -1 -1 -1 -1 -1 -1     14 -1 -1
     1      19      15     31  1174 -1 -1     31  1174 -1 -1 -1 -1 -1 -1 -1 -1     15 -1 -1
     1      19      16     31  1173 -1 -1     31  1173 -1 -1 -1 -1 -1 -1 -1 -1     16 -1 -1
     1      19      17     33  1173 -1 -1     33  1173 -1 -1 -1 -1 -1 -1 -1 -1     17 -1 -1
     1      19      18     32  1174 -1 -1     32  1174 -1 -1 -1 -1 -1 -1 -1 -1     18 -1 -1
     1      19      19     31  1172 -1 -1     31  1172 -1 -1 -1 -1 -1 -1 -1 -1     19 -1 -1
     2       5       1    139   812 -1 -1    139   812 -1 -1 -1 -1 -1 -1 -1 -1      1 -1 -1
     2       5       2    140   813 -1 -1    140   813 -1 -1 -1 -1 -1 -1 -1 -1      2 -1 -1
                                                                                  1,303          Top
```

Figure 8.6: Screen shot of a typical KW-18 file, for *ALL trajectories.*

Automatic Tracking

This mode involves the invocation of external tracking program(s) to track objects of interest, given that **an image sequence has been loaded** and **the Kolam-Tracker dialog has been opened**. All necessary widgets have been presented under Page 1: 'WAMI' . Automatic tracking may be currently performed via two trackers: the CSURF Tracker and the LOFT Tracker. The steps for invoking the **CSURF tracker** are illustrated sequentially in Figure 8.7, Figure 8.8 and Figure 8.9.

(a) **Creating** a new Object-ID

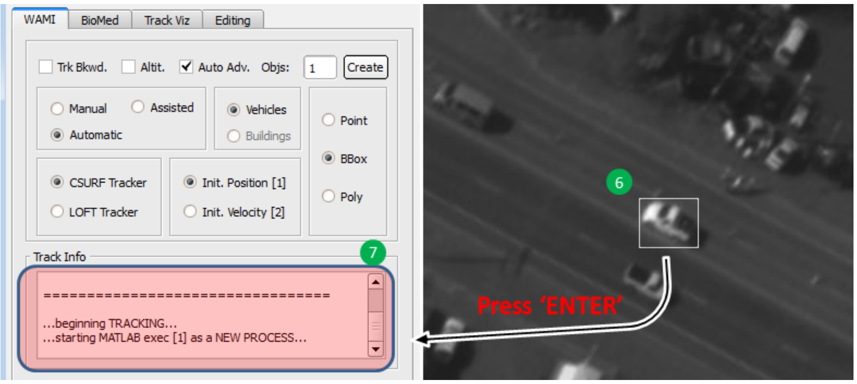


...Entering AUTO TRACKING MODE...
...Pan - Left Mouse. ; Zoom - Middle Mouse....
...Select Object - Right Mouse...

Draw 1-st BBox to select object using R.Mouse...
...when selection is satisfactory, press ENTER...

Figure 8.7: **Step I** of tracking an object using the CSURF Tracker. The user first *creates a new Object-ID* (Highlighted by rounded rectangle **1**). Creating a new Object-ID in KOLAM always *sets the Tracking Mode to Automatic* (**2**) and *sets CSURF as the selected tracker* (**3**). KOLAM *gives the user feedback and the next step to perform* in the tracking process (**4**). The object (vehicle in this case) to be tracked is also highlighted (**5**).

(b) **Selecting** the object for tracking



☐ Trk Bkwd. ☐ Altit. ☑ Auto Adv. Objs: 1

○ Manual ○ Assisted ● Automatic

● Vehicles ○ Buildings

○ Point ● BBox ○ Poly

● CSURF Tracker ○ LOFT Tracker

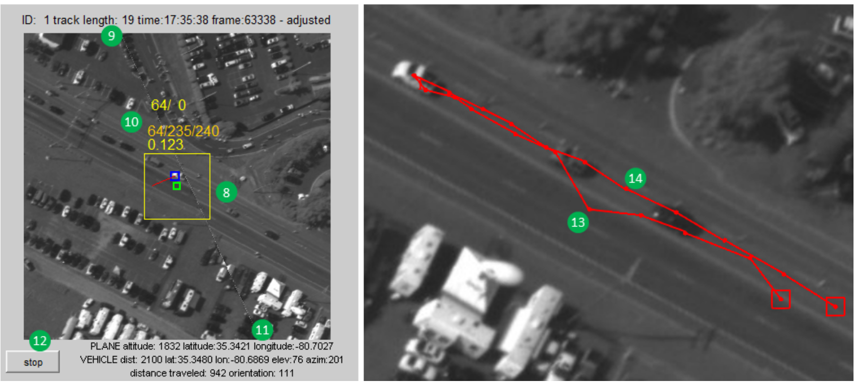
● Init. Position [1] ○ Init. Velocity [2]

Track Info

```
================================
...beginning TRACKING...
...starting MATLAB exec [1] as a NEW PROCESS...
```

Figure 8.8: **Step II** of tracking an object using the CSURF Tracker. The user now draws a bounding box around the object to be tracked with the *right mouse button* (**6**). The drawn box must include the object and a *small* amount of background (again, see (**6**)). This box *may be re-drawn as many times as the user needs* to get it right. When satisfied with the box drawn, the user presses the 'ENTER' key. This action *ends the object selection phase* of tracking; *creates a .txt1 parameter file* containing information about the drawn box, dataset path, output file path etc. *and invokes the tracker program* as an external process, with the .txt1 file as a parameter. Feedback regarding the same is presented by KOLAM to the user (**7**).

(c) Tracker **Startup**



ID: 1 track length: 19 time:17:35:38 frame:63338 - adjusted

64/ 0

64/235/240

0.123

PLANE altitude: 1832 latitude:35.3421 longitude:-80.7027

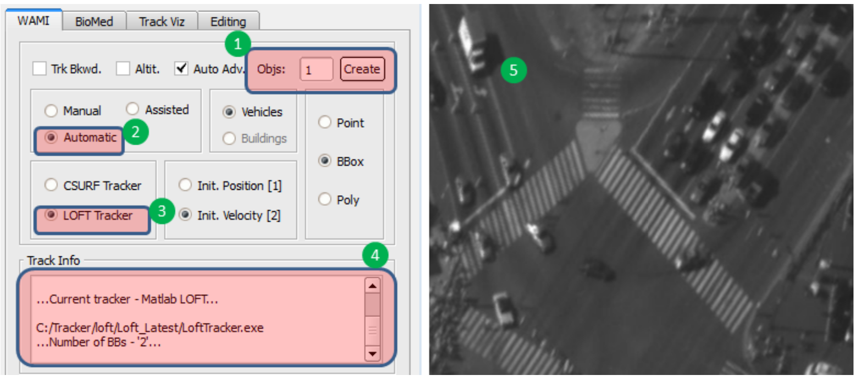
VEHICLE dist: 2100 lat:35.3480 lon:-80.6869 elev:76 azim:201

distance traveled: 942 orientation: 111

Figure 8.9: **Step III** of tracking an object using the CSURF Tracker. Once initialized, the CSURF program begins tracking the object (**8**). While tracking, CSURF presents *additional useful information* to the user (**9**, **10**, **11**) as well as the means to terminate itself (**12**). The trajectories for the object - *unregistered* (**13**) and *registered* (**14**) - are drawn in KOLAM *as soon as they are generated* by the tracker.

The steps for invoking the **LOFT tracker** are illustrated sequentially in Figure 8.10, Figure 8.11, Figure 8.12 and Figure 8.13.

(a) **Creating** a new Object-ID



□ Trk Bkwd. □ Altit. ☑ Auto Adv. Objs: 1

○ Manual ○ Assisted ● Automatic

● Vehicles ○ Buildings

○ Point ● BBox ○ Poly

○ CSURF Tracker ● LOFT Tracker

○ Init. Position [1] ● Init. Velocity [2]

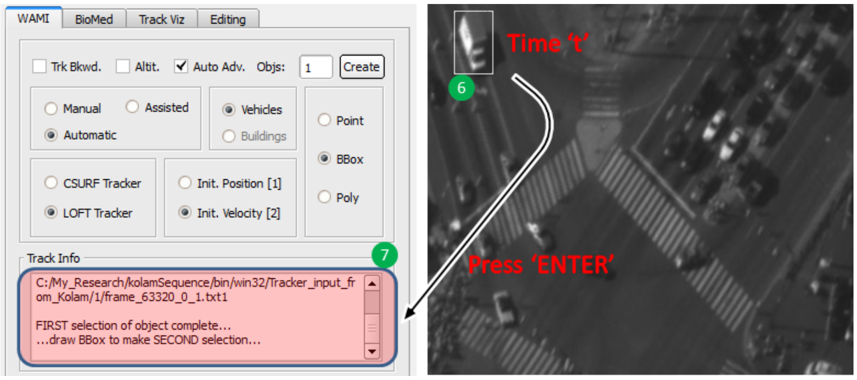
Track Info

...Current tracker - Matlab LOFT...

C:/Tracker/loft/Loft_Latest/LoftTracker.exe
...Number of BBs - '2'...

Figure 8.10: **Step I** of tracking an object using the LOFT Tracker. The user first *creates a new Object-ID* (Highlighted by rounded rectangle **1**). Creating a new Object-ID in KOLAM always *sets the Tracking Mode to Automatic* (**2**) and sets CSURF as the selected tracker. The user *needs to change the tracker to LOFT* (**3**). KOLAM *gives the user feedback and the next step to perform* in the tracking process (**4**). The object (vehicle in this case) to be tracked is also highlighted (**5**).

(b) **Selecting** the object: **PHASE 1**



□ Trk Bkwd. □ Altit. ☑ Auto Adv. Objs: 1

○ Manual ○ Assisted ● Automatic

● Vehicles ○ Buildings

○ Point ● BBox ○ Poly

○ CSURF Tracker ● LOFT Tracker

○ Init. Position [1] ● Init. Velocity [2]

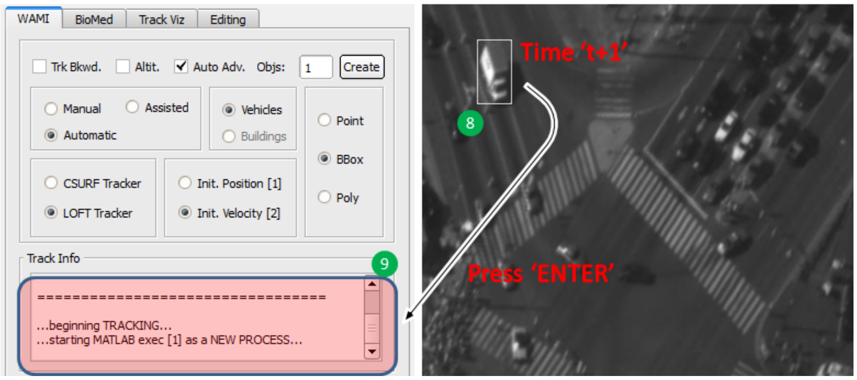
Track Info

C:/My_Research/kolamSequence/bin/win32/Tracker_input_from_Kolam/1/frame_63320_0_1.txt1

FIRST selection of object complete...
...draw BBox to make SECOND selection...

Figure 8.11: **Step II** of tracking an object using the LOFT Tracker. The user now draws a bounding box around the object to be tracked with the *right mouse button* (**6**). The drawn box must include the object and a *small* amount of background (again, see (**6**)). This box *may be re-drawn as many times as the user needs* to get it right. When satisfied with the box drawn, the user presses the 'ENTER' key. Since LOFT takes **TWO user-drawn BBoxes as input**, pressing 'ENTER' *advances the sequence by 1 time step* and lets the user draw the next BBox (**7**).

(c) **Selecting** the object: **PHASE 2**



□ Trk Bkwd. □ Altit. ☑ Auto Adv. Objs: 1 Create

○ Manual ○ Assisted
● Automatic

● Vehicles
○ Buildings

○ Point
● BBox
○ Poly

○ CSURF Tracker
● LOFT Tracker

○ Init. Position [1]
● Init. Velocity [2]

Track Info

```
================================

...beginning TRACKING...
...starting MATLAB exec [1] as a NEW PROCESS...
```

Figure 8.12: **Step III** of tracking an object using the LOFT Tracker. The user now draws a *SECOND* bounding box around the object to be tracked, for the *SECOND time step* (**8**). When satisfied, the user presses the 'ENTER' key. This action *ends the object selection phase* for LOFT; *creates a .txt1 parameter file* containing information about the drawn boxes, dataset path, output file path etc. *and invokes the tracker program* as an external process, with the .txt1 file as a parameter. Feedback regarding the same is presented by KOLAM to the user (**9**).

(d) Tracker **Startup**

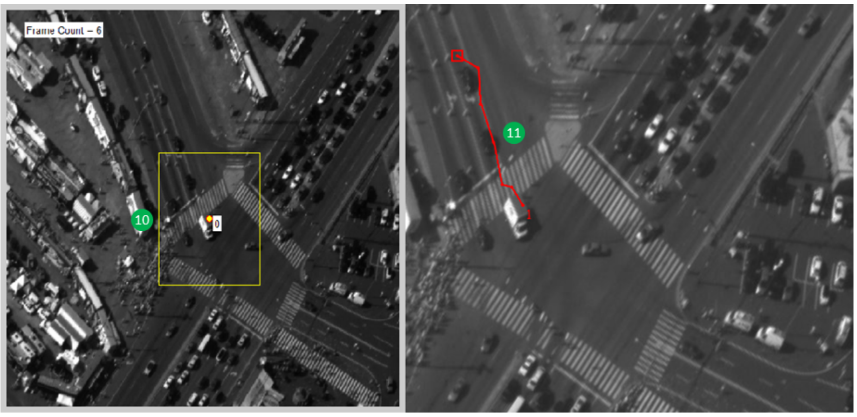


Figure 8.13: **Step IV** of tracking an object using the LOFT Tracker. Once initialized, the LOFT program begins tracking the object (**10**). While tracking, LOFT presents *additional useful information* to the user. The trajectory for the object (**11**) - is drawn in KOLAM *as soon as it is generated* by the tracker.

Manual Tracking

In the manual tracking mode the user tracks objects by hand marking the location of the target in each frame. These points can be visualized as a connected trajectory (vector plot in the overlay plane) as in the automated tracking mode. KOLAM currently supports marking of objects using points, bounding boxes or polygons.

The fastest and easiest means of manual tracking involves using the *auto-advance* option to automatically step to the next frame as soon as the user marks the location of the object. In this mode of operation, an expert user can quickly generate long ground-truth trajectories.

Tracking performed in either the automatic or manual modes may include erroneous target locations. Tracking errors can be repaired using KOLAM's trajectory editing facilities, which are the subject of Chapter 10.

Assisted Tracking

Current automatic tracking algorithms are not trustworthy enough in dense urban environments with many movers, while fully manual tracking is time consuming and error prone. Assisted tracking augments an automatic tracker with manual intervention for rapid and accurate trajectory generation, especially when tracking involves many similar targets maneuvering through multiple occlusions and shadows in complex environments and flows.

The assisted tracking mode merges KOLAM's automatic tracking and trajectory editing capabilities in a manner that maximizes user effectiveness. In the assisted tracking mode the user is able to stop the automatic tracker as necessary, manually correct trajectory errors, or add target location information in a difficult to track region, then quickly switch back to the auto-tracker mode in a seamless fashion. The sequence of operations in assisted tracking includes use of the automatic tracking process, combined with track editing procedures (not shown) which are performed by the user in an iterative fashion until the supervised tracking task is completed.

CHAPTER 9

MULTI-MONITOR DISPLAY

This chapter describes KOLAM's ability to utilize a high-performance cluster; to display huge datasets across a tiled, multi-monitor 'Wall' display system. Additional **networking parameters** affecting the performance of the Wall system may be modified via the Preferences - *Performance Tuning* section of the Kolam-Preferences tool. An illustration of KOLAM displaying multiple layers at different zoom levels on a multi-monitor display is shown in Figure 9.1.

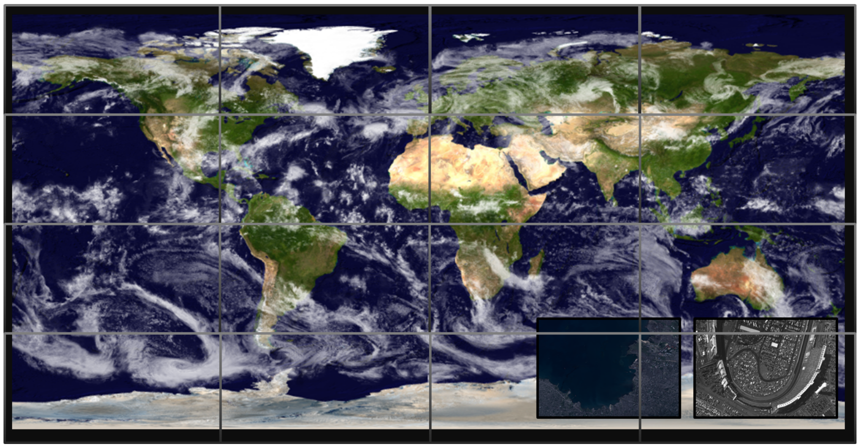


Figure 9.1: An illustration of KOLAM being used on a multi-monitor display.

Unlike the single monitor display mode, this mode of operation involves *multiple command-line invocations* of KOLAM on the nodes of the cluster, each of which takes *multiple parameters* that designate what to display on each tile (ie. monitor) of the multi-monitor system. **It is highly recommended** that the command-line instructions be saved in the form of *script files*, in order **to save initial setup time** and to **enable off-line editing** of the display parameters. Furthermore, **it is imperative** that the nodes of the cluster be on the **same network subnet**; the wall display mode will not function otherwise.

Example Setup Scenario

Consider that the available multi-monitor setup is a **2x2 grid**, with each of the four monitors having a resolution of 2560 x 1600. Let the KOLAM instance that will be designated as the broadcaster display to the *lower-left* monitor. Three other computers on the same subnet, connected to the other three monitors on the grid, are designated as receivers (since they receive update events generated by the user from the broadcaster instance). The command-line instructions to load a PSS sequence of image files, *in order*, on the **lower-left**, **lower-right**, **upper-left** and **upper-right** monitors (these must be issued on the respective computers) are as follows:

```
kolam -geometry 2560x1600 <full_path>/frame_#####_0.pss -x 2560 -y 0
kolam -geometry 2560x1600 <full_path>/frame_#####_0.pss -x 0 -y 0
kolam -geometry 2560x1600 <full_path>/frame_#####_0.pss -x 2560 -y 1600
kolam -geometry 2560x1600 <full_path>/frame_#####_0.pss -x 0 -y 1600
```

Current System Limitations

User events are processed and propagated ONLY IF they occur within the monitor(s) to which the broadcaster instance of KOLAM is displaying. Events that occur on any receiver instance are ignored in favor of display synchronization updates that are dispatched from the broadcaster. Making another running instance of KOLAM a broadcaster instance is currently a manual process: the current broadcaster instance must have its broadcaster status removed and must be designated a receiver instance, followed by designating the instance of interest (currently a receiver) as the next broadcaster. The hotkeys for setting the broadcaster/receiver status on a running KOLAM instance are listed on page 93.

Chapter 10

Trajectory Editing

This chapter describes the trajectory editing facilities available to the user in KOLAM. The various operations are: (a) **ADD** point(s) at either end of the trajectory, (b) **DELETE** point(s) from either end of the trajectory, (c) **MOVE** any point on the trajectory, (d) **SPLIT** a trajectory into two, and (e) **JOIN** two or more trajectories into a single trajectory.

ADD point(s) to either end of a Trajectory

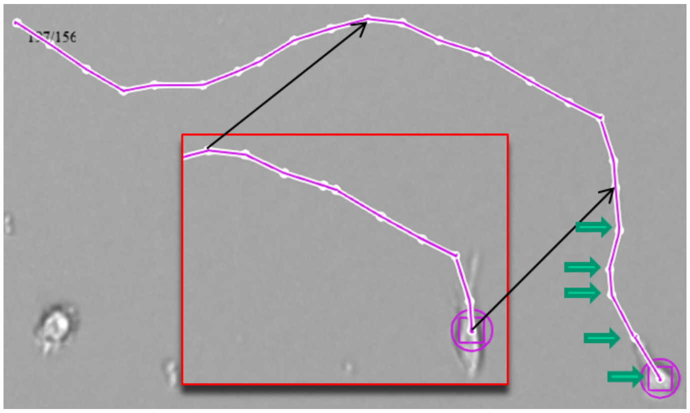


Figure 10.1: The ADD point operation. The red inset depicts a portion of the trajectory prior to addition of point(s). The correspondence between the inset and the final trajectory is denoted by the black arrows. The trajectory points indicated by the green arrows were added using the ADD operation.

The ADD operation allows for the addition of one or more points at either end of a selected trajectory. The user navigates to either one frame before the start of, or one frame after the end of the trajectory and drops the new point. Dropping a new point several frames before the trajectory start or after the trajectory end results in trajectory interpolation - new points are added by KOLAM for the missing frames. See Figure 10.1.

DELETE point(s) from either end of a Trajectory

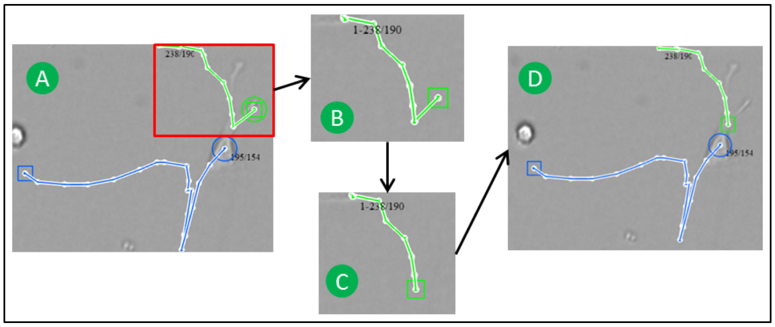


Figure 10.2: The DELETE point operation. The first point (rightmost point) on track '238/190' in inset '**A**' is an erroneous detection, and needs to be deleted. The relevant region is isolated in inset '**B**'. The effect of the deletion operation is depicted in '**C**', and the final view is shown in inset '**D**'.

The DELETE operation allows for the deletion of one or more points at either end of or at any point within a selected trajectory. The trajectory retains its ID when point(s) are deleted from either end. Deleting the whole trajectory is a special case of this operation. See Figure 10.2.

MOVE any point on a Trajectory

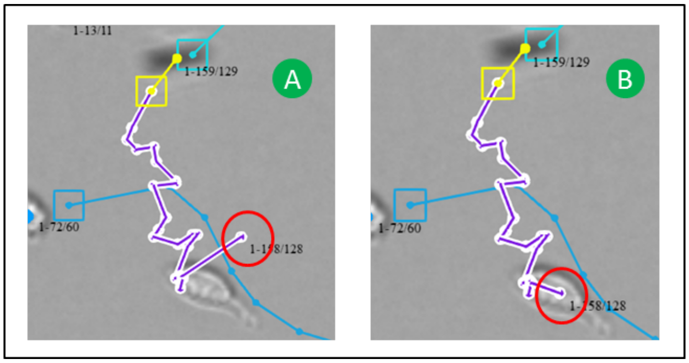


Figure 10.3: The MOVE point operation. The erroneous point position (in inset designated '**A**') and the corrected point position (in inset designated '**B**') are both highlighted with red circles.

The MOVE operation allows for the translation of one or more point(s) of the trajectory. MOVE does not involve change of trajectory ID. The MOVE operation is performed by right-clicking on the point of interest, and dragging it to its desired position. See Figure 10.3.

SPLIT a Trajectory

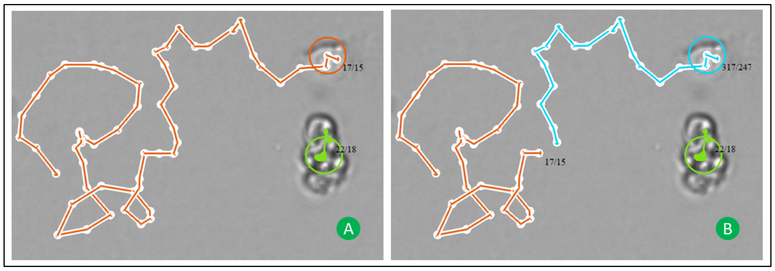


Figure 10.4: The SPLIT Track operation. The original, single track (in inset designated '**A**') and the two split tracks (in inset designated '**B**').

Deleting point(s) within the trajectory splits it in two, with the section prior to the deleted portion retaining the original trajectory ID, and the following section being given the first new available ID.

JOIN Two or More Trajectories

The JOIN operation allows for two or more trajectories to be merged together into a single, longer trajectory. A meaningful JOIN operation requires that one trajectory start at an earlier point in time than the other.

Pairwise Trajectory Join

In this mode, the user may select a pair of trajectories and join them into a single trajectory.

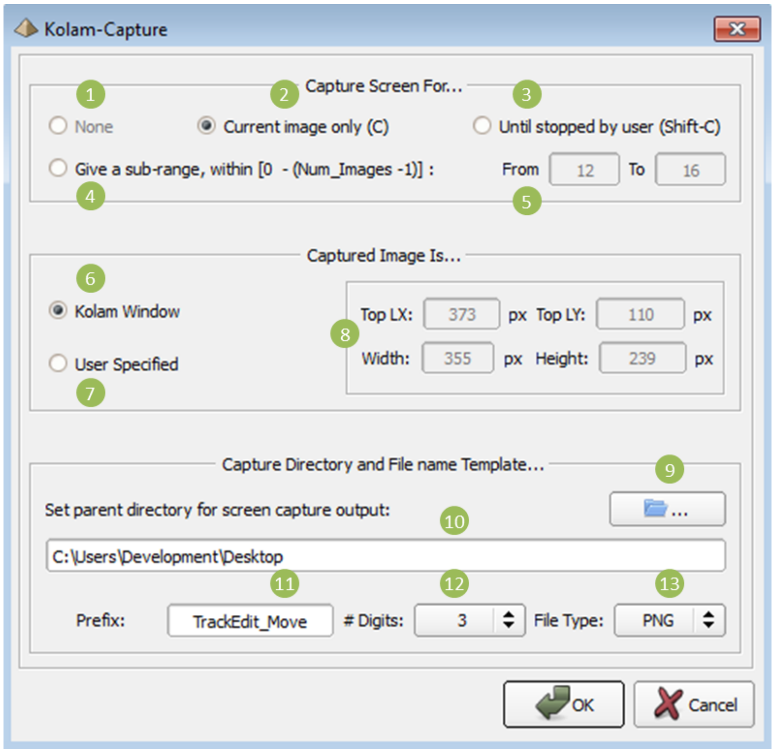
Multiple Trajectory Join

Sorts all selected trajectories according to starting frame number and joins them into a single trajectory. It involves aggregating pairwise trajectory joins in order of increasing trajectory start time. Each pairwise join is governed by the assumption that potential errors occur with increasing probability towards the end of any trajectory - the starting centroids of any automatic tracker initialization are almost always user supplied; as such, they are as reliable as ground truth. Therfore, in the event of trajectory overlap in time, one or more centroids are always deleted from the end of the aggregate trajectory, followed by appending the current trajectory to the end of the aggregate trajectory. This process is repeated for all remaining trajectories to produce the final aggregate trajectory. While extremely efficient in terms of time saved, we recommend that the user exercise caution while using this operation, as it involves centroid deletion from multiple trajectories.

CHAPTER 11

SCREEN CAPTURING

This chapter details the screen capturing facilities of KOLAM. KOLAM's tool for managing screen capturing settings, Kolam-Capture, is illustrated below.



Kolam-Capture

Capture Screen For...

○ None ● Current image only (C) ○ Until stopped by user (Shift-C)

○ Give a sub-range, within [0 - (Num_Images -1)] : From 12 To 16

Captured Image Is...

● Kolam Window

○ User Specified

Top LX: 373 px Top LY: 110 px

Width: 355 px Height: 239 px

Capture Directory and File name Template...

Set parent directory for screen capture output:

C:\Users\Development\Desktop

Prefix: TrackEdit_Move # Digits: 3 File Type: PNG

Figure 11.1: Kolam-Capture.

Widget Description

1. None

Do not perform screen capturing. This button has BEEN DISABLED; when running, KOLAM is capable of performing screen capturing for single image(s) and/or image sequences at any time.

2. Current image only

This button *enables* KOLAM's ability to capture a **single screen shot** of a single image (HotKey: see *Screen Capturing Related*). This is KOLAM's default mode for performing screen capturing (ie., it is enabled at KOLAM execution start time), and may be used for taking **single screen captures** of *Single Image Layers*, or from the **current time step** in an *Image Sequence.* The widgets for setting additional file parameters prior to capturing are described on page 66 and page 66.

3. Until stopped by user

This button *enables* KOLAM's ability to capture screen shots (of the same rectangular region) *from successive frames* of an image sequence, for **as long as it is being animated** (HotKey: see *Screen Capturing Related*). This implies that the user should disable sequence animation looping (see page 35) to avoid potentially repeating the same screen captures from being generated one or more times. User actions affecting sequence playback such as pausing (page 36), stopping (page 36), skipping (page 36) or stepping(page 36) directly affect the screen capturing capability of this mode. The widgets for setting additional file parameters prior to capturing are described on page 66 and page 66. The shortcut for performing Image Sequence capturing (reference provided above) is capable of turning this capture mode ON, even if the Kolam-Capture tool has not been opened. Once turned ON, KOLAM *will remain in sequence capturing mode* (ie., will continue to capture screen shots whenever a loaded sequence is animated), *UNTIL the same shortcut is used to switch* the capture mode BACK to single image capture.

4. Give a sub-range . . .

This button *enables* KOLAM's ability to capture screen shots (of the same rectangular region) *from successive frames* of an image, **within the frame number subrange** specified by the user (see page 65). If either the starting, or ending, or both ends of the subrange have out-of-range or invalid values specified, KOLAM appropriately substitutes these with the starting and/or ending frame index values. While KOLAM is performing screen capturing within the subrange, the user must take care **not to stop or alter the direction** of the animation.

5. From . . . To . . .

These two text entry widgets are **used to provide a valid subrange** (of the total number of frames of a loaded image sequence) for KOLAM to perform screen capturing within. The box on the left must hold the starting frame number, while the one on the right must hold the ending frame number. If either the starting, or ending, or both ends of the subrange have out-of-range or invalid values specified, KOLAM appropriately substitutes these with the starting and/or ending frame index values.

6. Kolam Window

This button sets KOLAM to **save the entire visible content of the display window** as a screen shot. Note that the display window may or may not be maximized (set to full screen); this setting works for both cases.

7. User Specified

This button sets KOLAM to **save a subregion of the visible content of the display window** as a screen shot. Subregion specification is possible via KOLAM's ROI (**Region of Interest**) Selection facility, and is described under.

8. Capture Dimensions Specified by User

These values are set in Kolam-Capture by the user using the ROI Selection facility to create a subregion of KOLAM's current display contents. It is further discussed under.

9. 'Set Capture Directory' Button

This button pops open a **directory selector dialog** which lets the user select the parent directory for KOLAM to save future screen capture image files.

10. Capture Directory full path

Displays the **full directory path** where screenshots will be written out by KOLAM. The value of this field is set by the directory selector button described above. If no path is listed in this field, KOLAM will save screen shot files in the same directory as the KOLAM executable.

11. Edit Save file Prefix

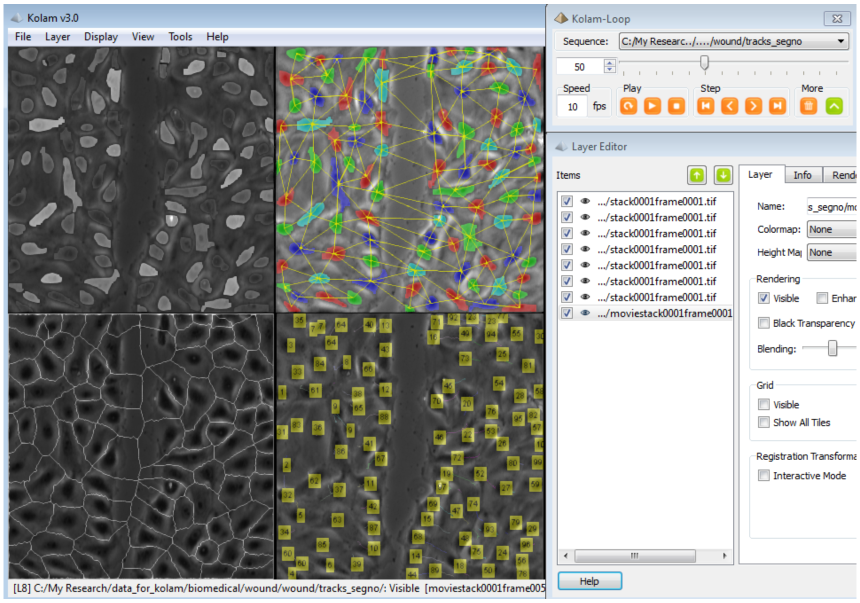
Sets the **text prefix** portion of the next to-be-generated filename. The file naming convention for screen captures is **<Text-Prefix>_<Numerical-Index>.<FileType>**. For example, with a *prefix 'Example'*, numerical *index value '4'*, the *file type being 'PNG'*, and 20 screen captures having already been performed by KOLAM, the next generated file will be named *'Example_0020.PNG'*.

12. Edit Save file Index

Sets the number of digits in the **numerical index** portion of the next to-be-generated filename. The number of digits determines the total number of unique filenames (with the same text prefix) that may be generated by KOLAM: for example, choosing '3' means filenames with indices ranging from 0 - 999, for a total of 1,000 unique file names.

13. Select Save file Type

Sets the current image **file type that all captured screenshots will be saved** as. KOLAM currently supports the JPG, TIF, PNG and BMP image file types for the saving of screenshots.



Sequence: C:/My Researc../..../wound/tracks_segno

50

Speed: 10 fps

☑ .../stack0001frame0001.tif
☑ .../stack0001frame0001.tif
☑ .../stack0001frame0001.tif
☑ .../stack0001frame0001.tif
☑ .../stack0001frame0001.tif
☑ .../stack0001frame0001.tif
☑ .../stack0001frame0001.tif
☑ .../moviestack0001frame0001

Name: s_segno/m

Colormap: None

Height Ma| None

Rendering

☑ Visible □ Enhar

□ Black Transparency

Blending:

Grid

□ Visible

□ Show All Tiles

Registration Transforma

□ Interactive Mode

[L8] C:/My Research/data_for_kolam/biomedical/wound/wound/tracks_segno/: Visible [moviestack0001frame005

Figure 11.2: KOLAM's screen capturing feature, with multiple layers and overlays. Eight sequences (4 copies of the wound healing dataset & 4 mask layers showing different results, from top left to bottom right: segmentation masks, 4-colored graph-based segmentation results, Voronoi diagram, and cell motility trajectories with ID labels; transparently overlaid on the dataset copies) have been organized in a grid pattern for comparative visualization. KOLAM can capture this grid visualization animation and output the captures utilizing user specified location, format and naming scheme.

Capturing with multiple Layers & Overlays

KOLAM is capable of capturing the screen output when multiple layers and overlays are being simultaneously interacted with. Multiple layers may be loaded, of which one or more layers may have their transparencies altered. These layers may overlap to varying degrees with each other. In addition to this, one or more overlays with trajectory data or any other type of annotation may be overlaid on the layers. See Figure 11.2 and its description for details of the image sequences and trajectories loaded. KOLAM can capture all the screen contents as fast as the screen capturing capability allows, and save the output images in the location and format of the user's choice.

Using the ROI Selection Utility

When visualizing imagery, the user needs (at times) to know the **dimensions of certain objects** of interest, their **position in the image** (both in terms of image coordinates as well as screen coordinates) and to **set a selected region to be captured** (via KOLAM's screen shot tool) for one or more images. KOLAM's Region-Of-Interest (ROI) utility fulfils these needs. The steps involved in using the ROI utility in performing these tasks are illustrated in Figure 11.3.

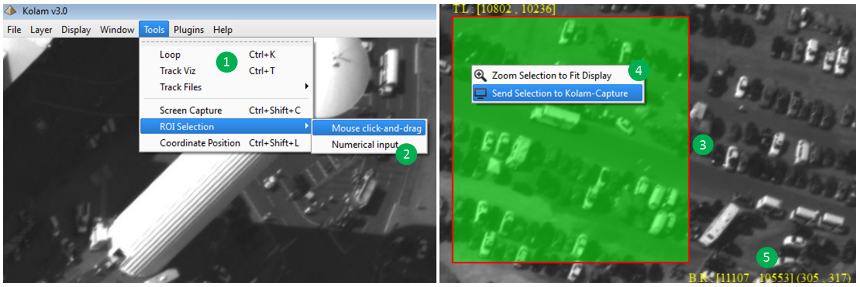

Kolam v3.0
File Layer Display Window Tools Plugins Help
Loop Ctrl+K
Track Viz Ctrl+T
Track Files
Screen Capture Ctrl+Shift+C
ROI Selection
Coordinate Position Ctrl+Shift+L
Mouse click-and-drag
Numerical input
1
2
T.L : [10802 , 10236]
Zoom Selection to Fit Display
Send Selection to Kolam-Capture
4
3
5
B.R : [11107 , 10553] (305 , 317)

Figure 11.3: KOLAM's ROI utility. The steps for its usage are as follows. The user first pans and zooms the image until the region of interest is displayed in a convenient manner. The user then activates the 'ROI Selection → Mouse Click-and-drag' menu item (**2**) from the 'Tools' Menu (**1**). This creates an overlay on which the ROI Selection may be made; the user performs this action by **Left-Click-Dragging** a (*green, semi-transparent*) **bounding box** around the Region-Of-Interest (**3**). As a convenience feature, KOLAM runs an *interactive display* of the **Top-Left** and **Bottom-Right** corners (**5**) of the selection box, in image coordinates. Once the user is satisfied with the box drawn (it may be redrawn as many times as needed to get it right), a *right-click within the box* pops up a **context menu** (**4**). Selecting **the second option** causes the screen coordinates of the selected region to be sent to the Kolam-Capture tool, where it may subsequently be used for subregion capturing. Selecting **the first option** causes KOLAM to zoom in on the selected region at the native image resolution. In both cases, **KOLAM remains in ROI selection** mode; and the *user must press the ESC key to exit* the mode.

CHAPTER 12

PLUGINS AND KOLAM

This chapter details the plugin architecture supported by KOLAM, and the requirements for writing and loading 3rd party plugins into KOLAM.

12.1 Segmentation Relabeling

We now present details regarding the Segmentation Relabeling plugin, which has been implemented in KOLAM in a tightly coupled fashion. The file management utility which allows users to specify the different files to be loaded, and allows management of project files specific to this tool, is illustrated in Figure 12.1.

Widget Description

1. Original Image Path

This text box displays the full file path to the original image. This path is normally set by using the '*Original Image Button*' (page 70), but it may also be set by a manual Copy-Paste-Edit action by the user.

2. Partition Label Image Path

This text box displays the full file path to the partition label image. This path is normally set by using the '*Partition Label Image Button*' (page 70), but it may also be set by a manual Copy-Paste-Edit action by the user.

3. Classifier Output Image Path

This text box displays the full file path to the classifier output image. This path is normally set by using the '*Classifier Output Image Button*' (page 70), but it may also be set by a manual Copy-Paste-Edit action by the user.

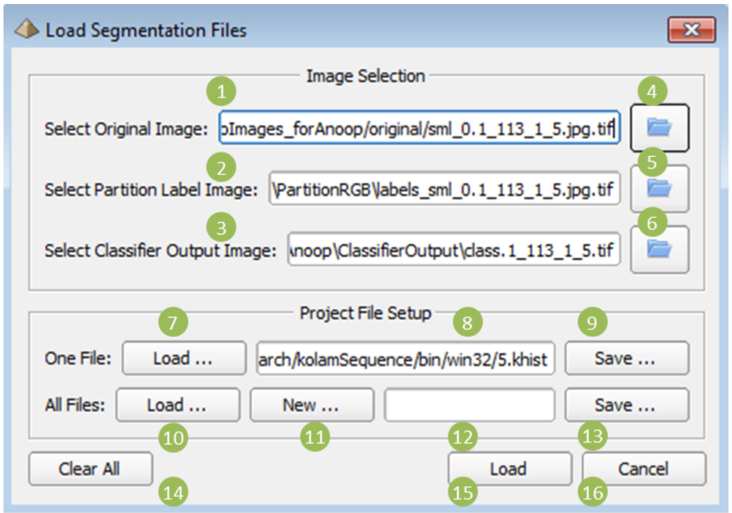

Load Segmentation Files

Image Selection

Select Original Image: bImages_forAnoop/original/sml_0.1_113_1_5.jpg.tif

Select Partition Label Image: \PartitionRGB\labels_sml_0.1_113_1_5.jpg.tif

Select Classifier Output Image: \noop\ClassifierOutput\class.1_113_1_5.tif

Project File Setup

One File: arch/kolamSequence/bin/win32/5.khist

All Files:

Figure 12.1: The file management utility for Segmentation Relabeling.

4. Original Image Button

Opens a File Selector dialog to specify the path to the original (source) image.

5. Partition Label Image Button

Opens a File Selector dialog to specify the path to the partition label image.

6. Classifier Output Image Button

Opens a File Selector dialog to specify the path to the classifier output image.

7. Load Project File Button

Opens a File Selector dialog which may be used to specify the path to a project file (extension - **.khist**) previously saved or imported by the user, to be loaded by KOLAM.

8. Project File Path

Gives the path to the most recently selected project file.

9. Save Project File Button

Opens a File Selector dialog, which lets the user save the currently set *Original, Partition Label* and *Classifier Output Image Paths* to a new project file.

10. Load Sequence Project File Button

Currently non-functional.

11. Create New Sequence Project File Button

Currently non-functional.

12. Current Sequence Project File Path

Currently non-functional.

13. Save Sequence Project File Button

Currently non-functional.

14. Clear All

Clears the display of all loaded image layers.

15. Load

Uses the currently set original image, partition label image and classifier output image paths to display the composite editable view for the user to begin performing segmentation relabeling operations. Making sure that **the paths of matching images are loaded** in each of the three fields is the **responsibility of the user**.

16. Cancel

Pressing this button closes the dialog without performing any additional actions.

Initial Setup for Relabeling

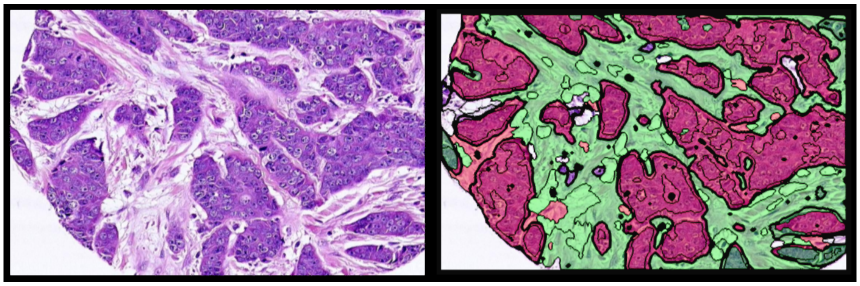
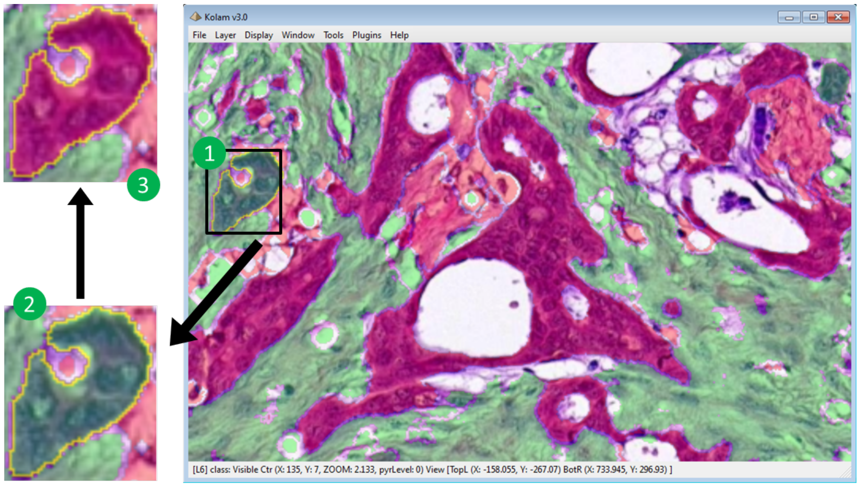


Figure 12.2: A source image from the Stanford Breast Cancer Microarray Dataset.

Figure 12.2 serves to illustrate the type of visualization KOLAM presents to the user for relabeling segmentation results. The left side of the illustration shows the original image (from the Stanford Breast Cancer microarray dataset), while the right side displays the result of overlaying and blending the segmentation results on the original image. The user may obtain this view in two different ways. ***First*** (if no previous input files have been loaded, the user wishes to change any preloaded paths, or no project files have been saved), the user employs widgets **4**, **5** and **6** (page 70) to load the appropriate input files; whose paths are then displayed in widgets **1**, **2** and **3** (page 69). Alternatively, if a project file was previously saved, the user may load its path with widget **7** (page 70). The ***second*** (and final) step involves using widget **15** (page 71), which causes all images to be loaded, overlaid and blended, to produce the aforementioned desired view; the user may edit segmentation labels on this image. KOLAM uses *three colors* to designate the different segmentation labels: RED → stroma, GREEN → epithelium and BLACK → borders. The **hotkeys** for toggling *ALL label borders*, or just the *border of the most recently relabeled* label, are listed on page 94.

Relabeling Procedure

Given that the user has completed the setup process as described on page 71, Figure 12.3 illustrates the steps involved in the relabeling process. The user can **identify the labels that have been incorrectly colored** by viewing the classifier image and the original image *separately* in order to compare them. Since the classifier image is *overlaid on* the original image, *its visibility may easily be toggled ON or OFF* using the '**V**' hotkey (page 93).



[L6] class: Visible Ctr (X: 135, Y: 7, ZOOM: 2.133, pyrLevel: 0) View [TopL (X: -158.055, Y: -267.07) BotR (X: 733.945, Y: 296.93)]

Figure 12.3: Steps involved in doing Segmentation Relabeling in KOLAM.

The user begins editing by identifying and right-clicking on a wrongly classified label, shown in (**1**) and enlarged in (**2**) to highlight the *orange border* of the selected label. The user performs relabeling by right-clicking within the selected label (**2**), this causes the color of the label to begin cycling through all available label colors. Relabeling is complete when the user cycles through to the correct label color, as shown in (**3**).

CHAPTER 13

KOLAM: How to..?

> ‘Okay, this manual is all fine and dandy, but ***how do I use KOLAM to do what I want***?’

This chapter strives to answer this question, by providing lists of actions for every possible usage scenario of KOLAM. On several occasions, the actions will reference images and sections in other parts of the manual.

> ‘But doesn’t that mean that I’ll have to scroll back and forth every single time I click on a link ?!’

Of course not ! The links used in this manual enable you to **jump right back to where you clicked** a particular link, by pressing the **Backspace key**. If you have a mouse with extra buttons that are mapped to ‘Page forward’ & ‘Page backward’, even better! Finally, on phones, tablets & other touch-driven devices, the ’Back’ button takes care of this need.

1. LOAD - a Single Image

- Use the menu: **File –> Open Image(s)...**. This opens your OS’s file browser with KOLAM’s supported image formats’ wildcards selected to display only image files with those extensions.
- Navigate to the location of the file, and double-click it.
- Alternatively, you can click it and then click the ’Open’ button, usually located at the bottom-right portion of the file browser dialog.

- OR, you can eschew KOLAM's menu system: drag-and-drop the image file of interest from a normal OS file browser dialog, onto KOLAM's main display area. (WARNING: Does not work under Mac OSX.)

2. LOAD - Images As A Sequence

- **NOTE:** It is recommended that the images of interest **all be of the same width and height**.
- Use the menu: **File –> Open Sequence...**. This opens your OS's file browser with KOLAM's supported image formats' wildcards selected to display only image files with those extensions.
- Navigate to the directory containing the files, and double-click on *any file*.
- Alternatively, you can click *any file* and then click the 'Open' button, usually located at the bottom-right portion of the file browser dialog.
- OR, you can eschew KOLAM's menu system: drag-and-drop the *directory containing the images of interest* from a normal OS file browser dialog, onto KOLAM's main display area. (WARNING: Does not work under Mac OSX.)

3. LOAD - Images into Separate Layers

- **NOTE:** Unlike for image sequences, the images of interest **can have different widths and heights** in this case.
- Use the menu: **File –> Open Image(s)...**. This opens your OS's file browser with KOLAM's supported image formats' wildcards selected to display only image files with those extensions.
- Navigate to the location of the files. All the files of interest need to be within the same directory.
- Select all the files, and then click the 'Open' button, usually located at the bottom-right portion of the file browser dialog.
- OR, you can eschew KOLAM's menu system: Select, then drag-and-drop *the image files of interest (NOT the directory)* from a normal OS file browser dialog, onto KOLAM's main display area. (WARNING: Does not work under Mac OSX.)

4. LOAD - a Color Map

- **NOTE:** A colormap may be loaded even if no image has been loaded prior. HOWEVER, be aware that if a colormap is loaded without an image layer, there will be no discernible visible effect in the main display area. When multiple images are loaded, a loaded colormap will be applied to the **currently active layer**.
- Use the menu: **File –> Open Colormap...**. This opens your OS's file browser with the .cm wildcard selected to display only colormap files with the .cm extension. To edit loaded colormaps, use the editing tool illustrated on page 27.

- Navigate to the location of the .cm file of interest. Load the file by either double-clicking on it, or by selecting it and clicking the 'Open' button.

- Once loaded, the colormap will be applied to the currently active layer. To change the layer to which the colormap is assigned, open the Layer Editor by using the menu: **Layer –> Layer Editor...**, selecting another layer in the 'Items' list thereby making it the currently active layer, and selecting the loaded colormap file from the colormap drop down on the top-right of the 'Layer' page. See Figure 4.1 and the widget description on page 25.

5. CLEAR - The display area

- KOLAM provides two options to clear the display area - clearing a single (the currently active) image layer, as well as clearing the display area of ALL loaded image layers. Once the display is cleared, you may load other image layers if so desired, or exit KOLAM.

- **NOTE:** Both the layer closing options described below currently **only work for single image layers, and not for image sequences**. If you have loaded an image sequence and wish to clear the display to load some other data, you will need to restart KOLAM. **The author is currently working on rectifying this issue**.

- Use the menu: **File –> Close Layer**. This clears the currently active image layer from the main display area.

- Use the menu: **File –> Close All Layers**. This clears ALL loaded image layers from the main display area.

6. CHANGE - Order of layers in the layer stack

- KOLAM loads layers in a stacked manner, with the most recently loaded layer being active and visible 'on top', given that no operations to alter the ordering of layers have been already performed in KOLAM. The Layer Editor (page 24) allows for the order of layers in the visibility stack to be interactively changed as desired.

- Use the menu: **Layer –> Layer Editor...**. This opens the Layer Editor tool. See Figure 4.1.

- Click on any layer's name in the Layer List (page 26) to make it active, and use the Up/Down arrow keys (page 26) in order to change its order in the Visibility Stack. This action may be repeated for as many layers as needed in order to obtain the desired visual effect.

7. CHANGE - Currently active layer

- The concept of the 'active layer' is central to KOLAM's operations. All operations in KOLAM are performed on the active layer. If the layer you wish to perform any operation is not currently active, **it must be set as so, before performing any further tasks**. The Layer Editor provides the facility to change the currently active layer.

- Use the menu: **Layer –> Layer Editor...**. This opens the Layer Editor tool. See Figure 4.1.
- Identify the layer of interest in the Layer List on the left, and click on its name to make it the active layer.
- If less than 11 layers have been loaded, the Layer Editor may be eschewed in favor of a **layer activating shortcut**: after ensuring that the main display area has focus, press a number from 0 - 9, in order to make the corresponding layer active.

8. CHANGE - Colormap associated with a layer

- Use the menu: **Layer –> Layer Editor...**. This opens the Layer Editor tool. See Figure 4.1.
- Identify the layer of interest in the Layer List on the left, and click on its name to make it the active layer.
- Select the colormap of interest from the 'Colormap:' drop-down (page 24) and click on it to associate it with the currently active layer. If you need to dissociate a colormap from this layer, select the top-most entry of the colormap drop-down list, which is 'None'. In either case, the display updates immediately to reflect the change.

9. CHANGE - Heightmap associated with a layer

- Use the menu: **Layer –> Layer Editor...**. This opens the Layer Editor tool. See Figure 4.1.
- Identify the layer of interest in the Layer List on the left, and click on its name to make it the active layer.
- Select the colormap of interest from the 'Heightmap:' drop-down (page 24) and click on it to associate it with the currently active layer. If you need to dissociate a heightmap from this layer, select the top-most entry of the heightmap drop-down list, which is 'None'. In either case, the display updates immediately to reflect the change.

10. TOGGLE - Visibility of a layer

- Use the menu: **Layer –> Layer Editor...**. This opens the Layer Editor tool. See Figure 4.1.
- Identify the layer of interest in the Layer List on the left, and click on its name to make it the active layer. This step **may be skipped** is the layer of interest was already active.
- Toggle the 'Visible' checkbox (Figure 4.1) for the desired visual effect.
- Like changing the currently active layer, toggling visibility of a particular layer **also has a shortcut** if less than 11 layers have been loaded. Perform the last step for changing the currently active layer (page 75) and press the 'V' key to toggle the visibility of the desired layer.

11. TOGGLE - Black Transparency of a layer

- In addition to alpha blending, KOLAM provides ‘Black Transparency’, which turns only those pixels of an image with pixel color ‘Black (0, 0, 0)’ fully transparent. This functionality may be toggled via the Layer Editor (Figure 4.1).
- Use the menu: **Layer –> Layer Editor...**. This opens the Layer Editor tool. See Figure 4.1.
- Identify the layer of interest in the Layer List on the left, and click on its name to make it the active layer. This step **may be skipped** is the layer of interest was already active.
- Toggle the ‘Black Transparency’ checkbox (Figure 4.1) in the ‘Rendering’ section for the desired visual effect.

12. TOGGLE - Tile grid for a layer

- When dealing with multi-resolution image data, KOLAM provides the ability to visualize the tiles of each level of the image pyramid as a grid, overlaid on the image. The visibility and color of this grid may be toggled and changed via the Layer Editor.
- Use the menu: **Layer –> Layer Editor...**. This opens the Layer Editor tool. See Figure 4.1.
- Toggle the ‘Visible’ check box located in the ‘Grid’ section on the ‘Layer’ page for the desired visual effect. The grid color may be altered by changing the color in the color picker, brought up by clicking the ‘Color..’ button.

13. TOGGLE - Per-Layer Transformation

- KOLAM’s *default mode of transformation* (ie. pan and zoom operations) involves transforming **ALL** loaded layers within a **single, global coordinate system**. It is *also possible* to transform **each** layer within its **own, local coordinate system** via a toggle in the Layer Editor.
- Use the menu: **Layer –> Layer Editor...**. This opens the Layer Editor tool. See Figure 4.1.
- Check the ‘Interactive Mode’ checkbox (Figure 4.1) in the ‘Registration Transformation’ section. Once you have completed the following steps, **remember to un-check the check box**.
- Click on the layer to be transformed in the ‘Items’ list to make it active.
- For general (but numerically imprecise) transformation, use the mouse to pan and/or zoom the active layer. For numerically precise transformation, use the ‘X Offset’, ‘Y Offset’ and ‘Scale’ numerical entry boxes located within the ‘Registration Transformation’ section.
- The two prior steps may be repeated for as many of the loaded layers, in order to obtain the desired visual presentation.

14. DISPLAY - Information about the active image layer

- Use the menu: **Layer –> Layer Editor...**. This opens the Layer Editor tool. See Figure 4.1. The 'Info' page displays all the relevant information of the currently active layer. See Figure 4.2.

15. DISPLAY - Data using Raster Rendering on a Plane

- **NOTE:** For this menu action to work, at least one layer must be loaded with either a single image or an image sequence.
- Use the menu: **Display –> 2D Raster**. KOLAM now uses raster rendering to display the image data.
- **NOTE:** Some images produce no display in Texture Rendering mode, which is the default display mode; but do display under the Raster Rendering mode. Therefore, in the event that loading a give image produces no change to the display, we recommend that the Raster Rendering mode also be tried prior to categorizing this as a bug.

16. DISPLAY - Data using Texture Rendering on a Plane

- **NOTE:** For this menu action to work, at least one layer must be loaded with either a single image or an image sequence.
- Use the menu: **Display –> 2D Texture**. KOLAM now uses texture mapped rendering to display the image data.
- **NOTE:** This is the default rendering mode for KOLAM, because texture rendering is faster, and handles the aliasing artifacts present in Raster Rendering.

17. DISPLAY - Data using Texture Rendering on a Sphere

- **NOTE:** For this menu action to work, at least one layer must be loaded with either a single image or an image sequence.
- Use the menu: **Display –> 3D Sphere**. KOLAM now uses texture mapping to map the image data onto a spherical surface.

18. CONFIGURE - Display Settings

- Use the menu: **File –> Preferences...**. This opens the KOLAM Preferences tool. See Figure 4.4.
- Click on the 'Performance Tuning' item in the list on the left.
- Open the 'Display' page. See page 29 for the widget descriptions.

19. CONFIGURE - Wall Networking Settings

- Use the menu: **File –> Preferences...**. This opens the KOLAM Preferences tool. See Figure 4.4.
- Click on the 'Performance Tuning' item in the list on the left.
- Open the 'Networking' page. See page 29 for the widget descriptions.

20. CONFIGURE - Display Fine-Tuning

- Use the menu: **File –> Preferences...**. This opens the KOLAM Preferences tool. See Figure 4.4.
- Click on the 'Performance Tuning' item in the list on the left.
- Open the 'System' page. See page 30 for the widget descriptions.

21. CONFIGURE - Tracking Executable paths for KOLAM

- Use the menu: **File –> Preferences...**. This opens the KOLAM Preferences tool. See Figure 4.5.
- Click on the 'Tracking Executable' item in the list on the left.
- See page 31 for the widget descriptions.

22. CONFIGURE - Tracking file save paths for KOLAM

- Use the menu: **File –> Preferences...**. This opens the KOLAM Preferences tool.
- Click on the 'Tracking File Paths' item in the list on the left.
- Set the file save paths as desired.

23. USE - KOLAM's Pan-Zoom Tool

- While KOLAM utilizes the mouse to perform interactive panning and zooming, a mouse might not always be available. This holds true for smaller laptop computers and other mobile devices.
- Use the menu: **View –> Pan-Zoom**. This brings up the Kolam-Pan+Zoom tool.
- The tool provides 3 capabilities: means to alter the zoom factor, to pan the display, and to navigate the different levels of a multi-resolution image.
- KOLAM's Preferences tool provides the means to alter both the **step size** and the **event frequency** for both Pan and Zoom operations, via the 'Display' page of 'Performance Tuning'. Reducing the event frequency interval provides a more fluid pan and zoom experience, but comes at the expense of what amounts to an 'event flood' for the event handling system. Increasing the pan or zoom step size will increase the 'speed' of panning and zooming, at the expense of fine grained control. Conversely, increasing the event frequency interval and decreasing the step size will have effects opposite to those mentioned above.

24. USE - KOLAM's Coordinate Position Tool

- KOLAM's Coordinate Position tool allows you to obtain image coordinates for the currently active layer in real time as you move the mouse over the display area. Where applicable (currently PSS image data ONLY) latitude and longitude information are also displayed in the same manner. If a trajectory dataset is loaded and the mouse is not being moved, this tool will display the image coordinates of the current Object ID as selected on the WAMI page of the Kolam-Tracker tool.
- Use the menu: **View –> Coordinate Position**.

25. USE - KOLAM's ROI Selection Tool

- Using the menu selection mentioned below, it is possible to interactively select a rectangular Region Of Interest (ROI) with respect to the currently active image layer. Figure 11.3 illustrates an example of interactive ROI selection.
- Use the menu: **View –> ROI Selection –> Mouse click-and-drag**.
- Usage instructions are provided with the previous illustration.

26. CAPTURE - The display area as a screenshot

- Use the menu: **View –> Screen Capture**. This opens the Kolam-Capture dialog. Details regarding the Kolam-Capture dialog and its components are presented in Chapter 11.
- Enable the 'Current image only (C)' radio button. This button is enabled by default.
- Under the 'Capture Directory and File name Template...' section, set the directory where you would like the screen captures to be saved, the name template for the image capture, the numerical range for images with this name structure (setting '2' digits will generate unique ordered names from 0 - 99, setting '4' digits will generate unique ordered names from 0 - 9999), and the image file type. For example, with a prefix 'test', '3' digits, and the 'PNG' file type; successive screen captures will be named 'test_000.PNG', 'test_001.PNG' and so on. Once a file is saved with the name 'test_999.PNG', the next file will be once again named 'test_000.PNG', over-writing any previously existing file with the same name.
- Click 'OK' at the bottom right to save the settings you've made. To discard any changes you've made, click 'Cancel' instead.
- Make sure the main display window has focus, and click the 'C' key to capture the content of the main display area. **NOTE:** In order to capture a sub-region of the whole screen, all you have to do is resize the KOLAM display window so that only the region of interest to you is visible in the display area: performing the capture operation now results in only that region being captured.

27. CAPTURE - The display area as a screenshot sequence

- **NOTE:** In order to capture screenshots from an image sequence as the sequence is being animated, an **image sequence** must already be loaded.
- Use the menu: **View –> Screen Capture**. This opens the Kolam-Capture dialog.
- Enable the 'Until stopped by user (Shift-C)' radio button. Once the sequence capturing task is complete, **make sure to re-enable the 'Current image only (C)' button, otherwise image capturing will continue!**
- Follow the same steps as for a single screen capture (page 80) for setting the screen capture parameters, and close the dialog.
- Resize the screen and position the image data (pan and/or zoom) as needed to set up your desired sequence capture area.
- Press the 'Play' key on the Kolam-Loop tool. This tool should already be visible if you are visualizing the sequence. This will cause the screen content to be captured while the image sequence is being animated.
- ONCE AGAIN! Once you've completed your sequence capturing task, **make sure to re-enable the 'Current image only (C)' button in the Kolam-Capture dialog, otherwise image capturing will continue!**

28. DISPLAY - An overview of the displayed data

- **NOTE:** For this menu action to have a visible effect, at least one image must be loaded.
- KOLAM's Overview tool allows to have a bird's eye view of the whole image data while allowing for even faster panning navigation of the data.
- Use the menu: **View –> Overview**. This brings up KOLAM's Overview tool, which is covered in detail on page 32.
- Panning and zooming the image layer in the main display area interactively updates the green grid region (Figure 4.6, zooming leads to Figure 4.7) in the Overview Tool.
- For fast panning of the image with a bird's eye view of the whole data, drag the green grid region in the Overview tool, and the data pans correspondingly in the main display area.
- The Overview tool **must be explicitly closed** once you've finished using it; it cannot be 'Escape'-d like other dialogs.

29. DISPLAY - Memory usage while interactively displaying data

- **NOTE:** For this menu action to have a visible effect, at least one image (single or sequence) must be loaded.

- KOLAM's Cache Glyph tool allows you to interactively visualize how KOLAM manages memory for multi-resolution imagery, while interacting with the imagery.
- Use the menu: **View –> Cache Glyph**. This brings up KOLAM's Cache Glyph tool, which is covered in detail on page 33.
- The Cache Glyph tool **must be explicitly closed** once you've finished using it; it cannot be 'Escape'-d like other dialogs.

30. DISPLAY - Transparent dialogs for KOLAM

- **NOTE:** For this menu action to have a visible effect, at least one of KOLAM's different dialogs must be visible.
- Having transparent dialogs is useful when the user wishes to keep multiple dialogs open, but overlapping each other so as to maximize the amount of visible display area, while being able to see settings in multiple dialogs in a relatively un-obscured manner.
- Use the menu: **View –> Transparent Dialogs**. Whatever dialogs are visible at the time will individually become visually fainter. If dialogs overlap, the content of the underlying dialogs will be partially visible through the bodies of the obscuring dialogs.

31. LOAD - Trajectories from a KW18 File

- **NOTE:** In order to load trajectories from a .kw18 file, an **image sequence** must already be loaded. If no image sequence is loaded, KOLAM will ignore the .kw18 file loading action.
- Use the menu: **Tools –> Track Files –> [KW18] - Load A File**. This opens your OS's file browser with the .kw18 wildcard selected to display only trajectory files with the .kw18 extension.
- Navigate to the location of the .kw18 file of interest. Load the file by either double-clicking on it, or by selecting it and clicking the 'Open' button.
- OR, you can eschew KOLAM's menu system: drag-and-drop the .kw18 file of interest from a normal OS file browser dialog, onto KOLAM's main display area. (WARNING: Does not work under Mac OSX.)

32. SAVE - All loaded KW18 Data to a KW18 File

- **NOTE:** In order to save all loaded KW18 data into a KW18 file, an **image sequence** must already be loaded, and at least one KW18 **trajectory dataset** must also be loaded.
- Use the menu: **Tools –> Track Files –> [KW18] - Save to KW-18 File**. This opens your OS's file browser with the .kw18 wildcard selected to display only trajectory files with the .kw18 extension.

– Navigate to the directory in which you would like to save the KW18 file. If you would like to over-write an existing KW18 file, open it using the dialog. Alternatively if you would like to create a new KW18 file into which to save the KW18 data, enter an unused file name and click the ‘Open’ button.

– All loaded KW18 data will be saved in the specified KW18 file. Please verify that the data was indeed saved by starting a **new instance of KOLAM** and checking the newly saved file.

33. SAVE - All loaded KW18 Data to a CSV File

– **NOTE:** In order to save all loaded KW18 data into a CSV file, an **image sequence** must already be loaded, and at least one KW18 **trajectory dataset** must also be loaded.

– Use the menu: **Tools –> Track Files –> [KW18] - Save to CSV File**. This opens your OS’s file browser with the .csv wildcard selected to display only trajectory files with the .csv extension.

– Navigate to the directory in which you would like to save the CSV file. If you would like to over-write an existing CSV file, open it using the dialog. Alternatively if you would like to create a new CSV file into which to save the KW18 data, enter an unused file name and click the ‘Open’ button.

– All loaded KW18 data will be saved in the specified CSV file.

– **NOTE:** Exporting KW18 trajectory information into the CSV format was added per a 3rd party request. KOLAM cannot currently *LOAD* trajectory information directly from a .csv file. Please contact the authors if you need this feature implemented. In such an event, the data columns in your .csv files *MUST* match the KW18 format, which is the top line in Figure 8.6.

34. DELETE - Most recently loaded KW18 Dataset

– **NOTE:** In order to delete the most recently loaded KW18 dataset, an **image sequence** must already be loaded, and at least one KW18 **trajectory dataset** must also be loaded. If no KW18 file has been loaded, the following actions will have no effect.

– Use the menu: **Tools –> Track Files –> [KW18] - Delete Last**.

– The trajectories from the last loaded KW18 file will be deleted: the visual effect is that all these trajectories will disappear from the display area, and can no longer be visualized unless the KW18 is reloaded.

35. DELETE - All loaded KW18 datasets

– **NOTE:** In order to delete all loaded KW18 datasets, an **image sequence** must already be loaded, and at least one KW18 **trajectory dataset** must also be loaded. If no KW18 file has been loaded, the following actions will have no effect.

– Use the menu: **Tools –> Track Files –> [KW18] - Delete All**.

- The trajectories from all loaded KW18 files will be deleted: the visual effect is that ALL trajectories will disappear from the display area, and can no longer be visualized unless one or more KW18 files are reloaded.

36. TOGGLE - Visibility of a *Single* Trajectory

- **NOTE:** In order to toggle trajectory visibility, an **image sequence** must already be loaded, and a **trajectory dataset** must also be loaded.
- If KOLAM's Tracker dialog is not visible, open it: use the menu: **Tools –> Tracker**. This opens the Kolam-Tracker dialog.
- Open the 'Track Viz' page of the dialog.
- Select the trajectory ID of the trajectory of interest in the 'Obj. ID' drop down.
- Under 'Track Properties', toggle the 'Visible' check box for the desired visual effect.

37. TOGGLE - Visibility of *All* Trajectories

- **NOTE:** In order to toggle trajectory visibility, an **image sequence** must already be loaded, and a **trajectory dataset** must also be loaded.
- If KOLAM's Tracker dialog is not visible, open it: use the menu: **Tools –> Tracker**. This opens the Kolam-Tracker dialog.
- Open the 'Track Viz' page of the dialog.
- Toggle the 'Draw All' check box for the desired visual effect.

38. TOGGLE - Visibility of *Selected* Trajectories

- **NOTE:** In order to toggle trajectory visibility, an **image sequence** must already be loaded, and a **trajectory dataset** must also be loaded.
- If KOLAM's Tracker dialog is not visible, open it: use the menu: **Tools –> Tracker**. This opens the Kolam-Tracker dialog.
- Open the 'Editing' page of the dialog, and enable the 'Select Track(s)' checkbox.
- In the main display area, pan and/or zoom in on any of the trajectories that you would like to have as 'Selected'. Right-click in close proximity to any of the centroids along the trajectory. The display will indicate that the trajectory has been successfully selected by enclosing the entire trajectory in a white boundary. See Figure 10.1 for an example of a selected trajectory. In the same manner; pan, and zoom in/out in order to select all other trajectories which you would like to be marked as 'Selected'.
- Toggle the 'Turn OFF Unselected Tracks' checkbox, located at the top-right of the 'Editing' page of the Kolam-Tracker dialog, for the desired visual effect.

39. TOGGLE - Display unstabilized and/or stabilized trajectories

- **NOTE:** Having *stabilized* and *unstabilized* versions of a given trajectory stems from whether any of the object tracker programs associated with KOLAM generate such trajectories in order to compensate for registration errors in the underlying image data. If you are not aware of this scenario, it is most likely that the trajectory data of interest to you does not have these trajectory types. In case you do, please follow these steps.
- If KOLAM's Tracker dialog is not visible, open it: use the menu: **Tools –> Tracker**. This opens the Kolam-Tracker dialog.
- Open the 'Track Viz' page of the dialog.
- When stabilized trajectories are included in the trajectory data, there are **three** visibility states: display *unstabilized* only, display *stabilized* only and display *both*. Use the 'Stabilize' check box under 'Track Properties' to toggle between these visibility states. If your trajectory data does not include stabilized trajectories, toggling the check box will have these effects corresponding to the three states just described: *visible, not visible, visible*.

40. TOGGLE - Display Names of Loaded KW18 Files

- **NOTE:** In order to display the names of loaded KW18 files, an **image sequence** must already be loaded, and at least one KW18 **trajectory dataset** must also be loaded.
- Use the menu: **Tools –> Track Files –> [KW18] - Display Names**. This causes the display of the names of all loaded KW18 files to be displayed in the top left corner of the main display area, overlaid as a form of annotation on any currently visible image data.
- **NOTE:** The menu item acts as a toggle, so use it again to turn off display of the file names.

41. TOGGLE - Group loaded KW18 Files by Color

- **NOTE:** In order to observe the visual effect of grouping KW18 file data by color, an **image sequence** must already be loaded, and at least one KW18 **trajectory dataset** must also be loaded.
- Use the menu: **Tools –> Track Files –> [KW18] - Group by Color**. This causes ALL trajectories which were loaded from a single KW18 to be rendered with the same color. Trajectories from different KW18s will be colored differently. This visualization mode is very useful in several scenarios: one such use case involving visually analyzing trajectories for individual objects that came from multiple sources: ground truth versus one or more tracker outputs, for example. You can return to the default trajectory visualization mode by activating this menu option again.

42. TOGGLE - Frame auto-advance during manual/visual tracking

– **NOTE:** Toggling frame auto-advance (or not) is applicable ONLY for the manual/visual tracking mode - auto-advance is *always enabled* for object selection while in the *automatic* tracking mode.

– If KOLAM's Tracker dialog is not visible, open it: use the menu: **Tools –> Tracker**. This opens the Kolam-Tracker dialog.

– Open the 'WAMI' page of the dialog.

– Enable the 'Auto Adv.' check box. The result will be visually apparent when an object primitive (a point, BBox or polygon) is drawn and confirmed (via the 'Enter' key). Disabling the check box will cause the image frames to no longer auto-advance once object primitives are drawn and confirmed.

43. TOGGLE - Display *full lengths* of all trajectories

– **NOTE:** In order to display the *entire temporal extent* of every loaded trajectory, regardless of the current time step, an **image sequence** must already be loaded, and at least one KW18 **trajectory dataset** must also be loaded.

– If KOLAM's Tracker dialog is not visible, open it: use the menu: **Tools –> Tracker**. This opens the Kolam-Tracker dialog.

– Open the 'Editing' page of the dialog.

– The 'Display All' button is a toggle - when pressed, it remains depressed unless pressed again.

– Pressing the 'Display All' button so that it's depressed will result in ALL trajectories being displayed for their entire temporal extents, regardless of the current time step. There is a single difference between how trajectories are visualized in this mode versus when they are not: the centroid corresponding to the current time step is surrounded by a large circle. This circle also provides a way to distinguish between trajectories of which the current time step is a part versus those that either *end before* or *begin after* the current time step: the aforementioned large circle will not be seen on any centroid along the latter set of trajectories.

44. CHANGE - Trajectory Color and Thickness

– **NOTE:** In order to change the color and/or thickness of trajectories, an **image sequence** must already be loaded, and at least one KW18 **trajectory dataset** must also be loaded.

– If KOLAM's Tracker dialog is not visible, open it: use the menu: **Tools –> Tracker**. This opens the Kolam-Tracker dialog.

– Open the 'Track Viz' page of the dialog.

– Trajectory thickness is a *SINGLE* property applied to *ALL trajectories*. Thickness can be altered by entering *any non-zero, integer value* in the text line next to the 'Thick' label, which can be found under 'Track Properties'. The display area updates immediately to reflect the newly set trajectory thickness value.

- Trajectory color is a *per-trajectory* property. First, pick the ID of the trajectory for which the color is to be altered in the 'Obj. ID' drop down. Next, change the color of the *current* trajectory by picking a different color via the color picker button, found under 'Track Properties'.

45. CHANGE - Input selection primitive used for Tracking

- **NOTE:** In order to change the input selection primitive used for tracking, an **image sequence** must already be loaded, and at least one **trajectory dataset** must also be loaded. Furthermore, the tracker program you wish to use to perform automatic tracking ***must support*** the input primitive you wish to select.
- If KOLAM's Tracker dialog is not visible, open it: use the menu: **Tools –> Tracker**. This opens the Kolam-Tracker dialog.
- Open the 'WAMI' page of the dialog, and make sure the 'Automatic' radio button is enabled (located on the top left portion of the page).
- The currently available choices for input selection primitive are 'Point', 'BBox' and 'Polygon', represented by radio buttons located on the right of the page. Select the desired primitive type to set the input primitive type as needed.
- **NOTE:** The CSURF and LOFT trackers currently use bounding box(es) (option 'BBox') as their input selection primitive. This selection is *within the tracker programs themselves* and is not within KOLAM's control. In order to use 'Point' or 'Polygon' input primitives for tracking, please contact Dr. Palaniappan for a appropriately configured tracker program.

46. CHANGE - Tracker type used for Tracking

- If KOLAM's Tracker dialog is not visible, open it: use the menu: **Tools –> Tracker**. This opens the Kolam-Tracker dialog.
- Open the 'WAMI' page of the dialog.
- The currently available tracker types to choose from are 'CSURF' and 'LOFT'. Choose the one you desire by enabling the corresponding radio button.
- **NOTE:** If you wish to use a different tracker program, please contact the author for support.

47. CHANGE - The type of Tracking to be performed

- **NOTE:** KOLAM currently supports three tracking types: automatic, manual and assisted. These types may be toggled between on the 'WAMI' page of the Kolam-Tracker dialog. However, in order for such a choice to have effect, an **image sequence** must already be loaded prior to setting the tracking type.
- If KOLAM's Tracker dialog is not visible, open it: use the menu: **Tools –> Tracker**. This opens the Kolam-Tracker dialog.

- The currently available tracking types to choose from are ‘Automatic’, ‘Manual’ and ‘Assisted’. ‘Automatic’ is selected by default. You may switch between the ‘Automatic’ type and the ‘Manual’ type of tracking by clicking on the respective radio buttons.
- **NOTE:** ‘Automatic’ tracking requires correctly installed and configured tracker programs that KOLAM can invoke

48. ADD - Point(s) to a trajectory

- **NOTE:** In order to ADD point(s) to a trajectory, an **image sequence** must already be loaded, and at least one KW18 **trajectory dataset** must also be loaded.
- If KOLAM’s Tracker dialog is not visible, open it: use the menu: **Tools –> Tracker**. This opens the Kolam-Tracker dialog.
- Open the ‘Editing’ page of the dialog.
- Enable the ‘Select Track(s)’ checkbox, and select trajectories as desired. Now, press the ‘Display All’ button to display all trajectory data. Following this, enable the ‘Turn OFF Unselected Tracks’ checkbox, so that only *selected* trajectories are still being displayed.
- Enable the ‘Add Point’ checkbox, and select the trajectory of interest from the ‘Selected Tracks’ drop down.
- Pan and zoom in on the trajectory of interest. Depending on where the new centroid is to be added, enable the ‘Before Start’, or the ‘After End’, or the ‘In Between’ radio button. Navigate to the appropriate time step in the image sequence.
- In the cases of adding the new centroid either before the current beginning, or after the current end of the trajectory, there are two methods for adding the point: either at the immediately preceding or succeeding time step, OR several time steps before or after. In the latter case, a line is drawn from the old beginning/ending to the new beginning/ending, with additional points corresponding to the skipped time steps being placed at regular intervals along this line.
- Click and add the new point. The selected trajectory is visually updated with the new point added in. See Figure 10.1 for an example.
- If additional points were created and artificially placed due to skipped time steps, these additional points can be **moved** to their correct locations in a following step. See page 89.
- Exit the **point(s) adding mode** by unchecking the ‘Add Point(s)’ checkbox.

49. DELETE - Point(s) from a trajectory

- **NOTE:** In order to DELETE point(s) from a trajectory, an **image sequence** must already be loaded, and at least one KW18 **trajectory dataset** must also be loaded. Furthermore, this description **ONLY COVERS** the cases of centroid deletion from either the *beginning* or the *end* of a trajectory; deleting any other centroid is covered under 'SPLIT a trajectory in two' (page 90).
- If KOLAM's Tracker dialog is not visible, open it: use the menu: **Tools –> Tracker**. This opens the Kolam-Tracker dialog.
- Open the 'Editing' page of the dialog.
- Enable the 'Select Track(s)' checkbox, and select trajectories as desired. Now, press the 'Display All' button to display all trajectory data. Following this, enable the 'Turn OFF Unselected Tracks' checkbox, so that only *selected* trajectories are still being displayed.
- Enable by the 'Delete Point(s)' checkbox, and select the trajectory of interest from the 'Selected Tracks' drop down.
- Pan and zoom in on the trajectory of interest. Depending on where the new centroid is to be deleted from, enable the 'Upto Start', or the 'Upto End' radio button.
- Navigate to the appropriate time step in the image sequence. This time step can be either the starting point, several points after the starting point (both in the case of 'Upto Start'), or it can be the end point, or several points before the end point (both in the case of 'Upto End').
- Click on the point to be deleted. In the cases where the point clicked is neither at the start or at the end, all points upto and including the starting (or the ending) point are also deleted. The display is updated immediately.
- Exit the **point(s) deleting mode** by unchecking the 'Delete Point(s)' checkbox. See Figure 10.2 for an example of point deletion.

50. MOVE - A point on a trajectory

- **NOTE:** In order to MOVE point(s) on a trajectory, an **image sequence** must already be loaded, and at least one KW18 **trajectory dataset** must also be loaded.
- If KOLAM's Tracker dialog is not visible, open it: use the menu: **Tools –> Tracker**. This opens the Kolam-Tracker dialog.
- Open the 'Editing' page of the dialog.
- Enable the 'Select Track(s)' checkbox, and select one or more trajectories.
- Enable the 'Other' checkbox, followed by the 'Move Point' checkbox. Now, select the trajectory of interest from the 'Selected Tracks' drop down.
- Pan and zoom in on the trajectory of interest, and the centroid to be moved.

- Right-click on, or in very close proximity to, the centroid to be moved. Now drag to the new location; the trajectory display updates immediately to reflect the changing position of the moved point.
- Exit the **point moving mode** by unchecking both the 'Other' and 'Move Point' checkboxes. See Figure 10.3 for an example.

51. JOIN - Two or more trajectories

- **NOTE:** In order to JOIN two or more trajectories, an **image sequence** must already be loaded, and at least one KW18 **trajectory dataset** must also be loaded.
- If KOLAM's Tracker dialog is not visible, open it: use the menu: **Tools –> Tracker**. This opens the Kolam-Tracker dialog.
- Open the 'Editing' page of the dialog.
- Enable the 'Select Track(s)' checkbox, and select one or more trajectories.
- Enable the 'Connect two Tracks' checkbox, and select the two trajectories of interest from the 'Join from Track' and 'To Track' drop downs.
- **NOTE:** Care *MUST* be taken to verify the temporal extents of both trajectories ***prior*** to performing a JOIN operation. If the two trajectories overlap temporally, **temporally duplicate centroids will be deleted from the ending of the trajectory that starts earlier in time. Depending on user choice, this can either be the 'Join From' trajectory or the 'Join To' trajectory**.

52. SPLIT - A trajectory in two

- **NOTE:** In order to SPLIT a trajectory in two, an **image sequence** must already be loaded, and at least one KW18 **trajectory dataset** must also be loaded.
- If KOLAM's Tracker dialog is not visible, open it: use the menu: **Tools –> Tracker**. This opens the Kolam-Tracker dialog.
- Open the 'Editing' page of the dialog.
- Enable the 'Select Track(s)' checkbox, and select *at least* one trajectory. Now, press the 'Display All' button to display all trajectory data. Following this, enable the 'Turn OFF Unselected Tracks' checkbox, so that only *selected* trajectories are still being displayed.
- Enable the 'Delete Point' checkbox, and select the trajectory of interest from the 'Selected Tracks' drop down.
- Pan and zoom in on the trajectory of interest. Enable the 'In Between' radio button.
- Navigate to the appropriate time step in the image sequence.
- Click on the point to be deleted. Deletion of the point causes the trajectory to which the point belonged to be split into two. The portion of the trajectory that comes later in time w.r.t the deleted point is constituted into a new trajectory, with the first available trajectory ID. See Figure 10.4 for an example.

53. USE - KOLAM's Interactive Segmentation Relabeling Plugin

– Use the menu: **Tools –> Vision –> Segmentation Relabeling**. This brings up the file loading interface for KOLAM's Interactive Segmentation Relabeling plugin.

– Interactive relabeling is covered in full detail in Chapter 12.

54. DISPLAY - KOLAM's Shortcuts

– Use the menu: **Help –> Kolam Shortcuts**. This opens up a web page which enumerates KOLAM's shortcuts under the different, relevant sections.

55. DISPLAY - Information about KOLAM

– Use the menu: **Help –> About Kolam...**. This displays the page with details about KOLAM, author information and executable build information.

Chapter 14

Hotkey Listing

This chapter serves as a listing of KOLAM's hotkeys, which are grouped into categories for user convenience. In order **to use any given shortcut**, the associated dialog/window in KOLAM **must be active** (click on it with the left mouse button to be sure).

Menu Bar:

File → Open Image	Ctrl + O
File → Open Sequence	Ctrl + N
Close Layer	Ctrl + W

Layer → Toggle Tile Grid	G
Layer → Layer Editor	Ctrl + L
Layer → Layer Info	Ctrl + I
Layer → Color Map Editor	Ctrl + C

Window → Overview	Ctrl + V
Window → Cache Glyph	Ctrl + G

Tools → Kolam-Loop	Ctrl + K
Tools → Kolam-Tracker	Ctrl + T
Tools → Kolam-Screen Capture	Ctrl + Shit + C
Tools → Kolam-Coordinate Position	Ctrl + Shift + L

Help → Kolam Help	Ctrl + H

Kolam-Loop (MUST have focus):

Toggle Play/Pause	Space
Next Frame in Sequence	→
Prev Frame in Sequence	←

Main Window (MUST have focus):

General

Pan Layer Left	[Click-Drag] Left Mouse [Left]
Pan Layer Right	[Click-Drag] Left Mouse [Right]
Pan Layer Up	[Click-Drag] Left Mouse [Up]
Pan Layer Down	[Click-Drag] Left Mouse [Down]
Zoom IN TO Layer	[Scroll] Mouse Wheel [Forward]
Zoom OUT OF Layer	[Scroll] Mouse Wheel [Backward]
Reset Layer Zoom & Position	Space
Enhance Layer	E
Quit KOLAM	Q
Toggle Full screen	F
Toggle Layer visibility	V
View Layer 0 . . . 9	0 . . . 9
Display at High Quality	Shift + L
Display for Best Interaction	Shift + P
Display Balancing Both	Shift + B
Toggle *All events* for Display	Z
Toggle *Compressed events* for Display	Z

Tracking Related

Re-initialize current Object ID for automatic tracking	R

Trajectory Drawing Related

Toggle Trajectory visibility	B
Toggle Trajectory Tail visibility	D
Toggle Trajectory ID visibility	Shift + D

Screen Capturing Related

Single Screen Capture	C
Toggle Sequence Capture	Shift + C

Animation Related

Click Rewind ON	Shift
Click Rewind OFF	Alt

Wall Networking Related

Enable Broadcast Mode	Shit + M
Disable Broadcast Mode	M
Enable Receiver Mode	S

Segmentation Relabeling Related

Toggle All Label Borders (*BLACK*)	A
Toggle Corrected Label Border (*ORANGE*)	B

CHAPTER 15

KNOWN PROBLEMS & WORKAROUNDS

The goal of this chapter is to summarize KOLAM's current list of problems and limitations, as well as to provide workarounds for several of them. Performing the actions listed below will result in one or more of the following: diminished performance, impaired display and program crashing. Following the specific workaround will ensure the continued operation of KOLAM without the aforementioned problems. Further details are presented below.

1. Closing Image Sequences

KOLAM's capability to close a loaded image sequence is impaired, as of version 199 of the software system. Attempting to close an image sequence via the **File Menu** will clear the display, however; subsequent image or sequence loading will cause the display to be corrupted with data from the previously loaded and closed sequence. Continued operation of KOLAM is very likely to result in the program crashing. *This problem is* ***restricted to sequences****, closing of layers with individual images works fine.*

Workaround

Close and restart KOLAM when a new image sequence needs to be loaded, **and** the old one needing to be cleared out. To avoid the effort involved in navigating multiple OS windows, and to prevent breaking the flow of presentation using KOLAM, we recommend that multiple instances of KOLAM be initiated at the same time: thus, when one needs to be closed, a fresh unused instance is immediately available for use.

2. Non-Tiled Imagery

KOLAM is highly optimized for the interactive display of tiled imagery. However, similar efficient handling of non-tiled imagery is an ongoing task. What this effectively means, is that KOLAM's interactive display performance degrades quickly as the dimensions of the non-tiled imagery increases. For example, consider an image sequence comprised of images each of dimensions 3000 x 3000 or above. Attempting to animate such a sequence at a frame rate greater than 5 frames per second will cause the display area to 'blank out', as the system cannot efficiently process the display requirements at present.

Tiled or Non-Tiled?

One means of determining whether the loaded data is tiled is to press the 'G' key while KOLAM's main display area has focus. A tiled image will have a grid overlaid on it, while a non-tiled image will have a single box enclosing the image and no grid. Another approach involves opening the 'Info' tab of the Layer Editor (Figure 4.2, page 27). A tiled image will have a 'Total tiles' value greater than 1, while a non-tiled image will not.

Tiling a Non-Tiled Sequence

The ImageMagick software suite (`http://www.imagemagick.org`) is free software distributed under the Apache 2.0 license that may be utilized to generate the multi-resolution tiled versions of non-tiled images. We recommend tiling any image sequence whose individual image resolution exceeds 1600 x 1200. The ImageMagick suite may be downloaded at:

`http://www.imagemagick.org/script/binary-releases.php`.

Tiling scripts for Windows, Linux distros and Mac OSX are provided below. This assumes that the user has correctly installed ImageMagick and all system paths have been set without any issues. If you are working with a Linux (or any oher OS) server and do not have root access, please ask your system administrator to install ImageMagick and to perform all the necessary setup so that you may access its commands from your account. Our example assumes that the output image names are the same as those of the input, with 'out_' prefixed to the name, use the TIFF (.tiff) format (mandatory, do not change output format), and that input images are in the jpeg (.jpg) format. For alternative input formats such as PNG, BMP etc. please use .png, .bmp etc. in the command strings below. To determine which image formats are supported by ImageMagick on your system, execute the following command in a command prompt or terminal:

```
identify -list format
```

For **Windows** OSes, open a DOS command prompt and navigate to the directory containing the images to be tiled. Execute the following command:

```
FOR %a in (*.jpg) DO convert %a -define tiff:tile-geometry=256x256 ptif:out_%a.tiff
```

For **Linux** distros, open a terminal and navigate to the directory containing the images to be tiled. Create a directory 'Tiled' in here for the tiled images to be generated in. Execute the following commands (assumes the 'tcsh' shell. For other shells please use the appropriate commands):

```
foreach img (*.jpg)
convert $img -define tiff:tile-geometry=256x256 ptif:Tiled/out_$img.tiff
end
```

Workaround

If the non-tiled image sequence cannot be tiled as given above, animate such non-tiled sequences at a maximum speed of 2 frames per second. Should the screen turn blank, pause the animation and use the mouse to move the blank display area around. This will refresh and restore the image display.

3. Sequence with different sized images

Loading a sequence containing images that do not have identical widths and heights results in different images being displayed at different positions on the display, that is, the images will not be centered at a common point. Further operation of KOLAM with such a sequence could potentially result in a program crash.

Workaround

Ensure that all images in a sequence have identical width and height dimensions.

4. Trajectory display hiccup on Linux and Mac OSX

Upon loading a trajectory dataset, the trajectories are not visible when animating the sequence under Linux and Mac OSX. Trajectory visibility is internally handled by a boolean state variable the handling of which by Linux and Mac OSX differs from Windows OSes. An OS-independent alternative to the current implementation is being explored.

Workaround

On the ‘Track Viz’ page of the Kolam-Tracker dialog (Figure 8.3, page 47), checking the ‘Stabilize’ checkbox twice will result in the trajectories being displayed. No further problems arise as a result of this anomalous behavior and KOLAM will continue to operate normally.

5. Centering on objects with trajectories

When the centering option (Figure 8.3, page 47) is enabled for an object with a trajectory during image sequence animation, KOLAM presents incorrect display behavior. Prolonged use in this state will result in a program crash, especially on the Linux and Mac OSX OSes.

Workaround

A solution is being formulated to fix the problem. In the interim, refrain from using the Centering option.

6. KW-18 Trajectory files with missing frame information

When loading and displaying trajectories, KOLAM cannot correctly handle broken trajectories, for instance, those generated when the tracked object is occluded for several frames. KOLAM displays a single contiguous trajectory, falling behind in time by 1 step for each missing time step. KOLAM operates normally however, and does not crash.

Workaround

Until KOLAM is updated with a fix, refrain from displaying trajectories that have missing time steps. If erroneous display behavior is observed, the problematic trajectory file may be turned off and unloaded from memory safely. KOLAM will continue to operate normally.

7. Problem with toggling Layer visibility in Layer Editor

Multiple layers loaded in KOLAM are listed to the left in the Layer Editor (Figure 4.1, page 24), and each item in this list is provided with a checkbox to toggle its visibility, alongwith an eye icon to indicate 'visible' or 'hidden' status. If multiple layers are loaded, clicking on the check boxes directly causes the incorrect layer to be toggled ON or OFF. This behavior does not cause KOLAM to crash.

Workaround

Click on the layer name first, thereby making it active, before clicking on the checkbox. This will cause the correct layer's visibility to be toggled as desired.

BIBLIOGRAPHY

[1] A. Haridas, R. Pelapur, J. Fraser, F. Bunyak, and K. Palaniappan, "Visualization of automated and manual trajectories in wide-area motion imagery," in *15th Int. Conf. Information Visualization*, 2011, pp. 288–293.

[2] J. Fraser, A. Haridas, G. Seetharaman, R. Rao, and K. Palaniappan, "KOLAM: A cross-platform architecture for scalable visualization and tracking in wide-area motion imagery," in *Proc. SPIE Conf. Geospatial InfoFusion III*, 2013, vol. 8747, p. 87470N.

[3] K. Palaniappan, F. Bunyak, P. Kumar, I. Ersoy, S. Jaeger, K. Ganguli, A. Haridas, J. Fraser, R. Rao, and G. Seetharaman, "Efficient feature extraction and likelihood fusion for vehicle tracking in low frame rate airborne video," in *13th Int. Conf. Information Fusion*, 2010.

[4] K. Palaniappan, A. Hasler, J. Fraser, and M. Manyin, "Network-based visualization using the distributed image spreadsheet (diss)," in *17th Int. AMS Conf. on Interactive Information and Processing Systems (IIPS) for Meteorology, Oceanography and Hydrology*, 2001, pp. 399–403.

[5] K. Palaniappan and J. Fraser, "Multiresolution tiling for interactive viewing of large datasets," in *17th Int. AMS Conf. Interactive Information and Processing Systems (IIPS) for Meteorology, Oceanography and Hydrology*, 2001, pp. 338–342.

[6] A. Haridas, F. Bunyak, and K. Palaniappan, "Interactive segmentation relabeling for classification of whole-slide histopathology imagery," in *IEEE Int. Symposium on Computer-Based Medical Systems (CBMS)*, Jun 2015, pp. 84–87.

[7] F. Bunyak, A. Hafiane, Z. Al-Milagi, I. Ersoy, A. Haridas, and K. Palaniappan, "A segmentation-based multi-scale framework for the classification of epithelial and stromal tissues in h&e images," in *Proc. IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 2015, pp. 450–453.

VITA

Anoop Haridas was born on October 28, 1979 in Sharjah, U.A.E.. He graduated high school from Indian High School, R.A.K, U.A.E in 1997, and obtained his undergraduate Bachelor of Technology (B.Tech) degree from the T.K.M. College of Engineering, University of Kerala in 2001. Anoop joined the Masters (M.S.) program in Computer Science at the Department of Electrical Engineering and Computer Science, University of Missouri in 2003, following which he was accepted (and transferred) into the PhD program at the same Department in 2005. Anoop began his initial PhD research with his program advisor Dr. Kannappan Palaniappan in 2005, and switched the topic and focus of his research in 2009. After a lengthy period as a doctoral student, Anoop graduated with his PhD degree from the Department of Computer Science in the Spring of 2018.

Anoop is actively seeking research oriented jobs in industry after his graduation in the field of visual analytics. Anoop was married in 2015 and is currently living with his wife in Toronto, Canada.