

**CellVisualizer: Exploring Hierarchical,
Multi-Dimensional Data with Applications to
High-Throughput Microscopy**

by

InHan Kang

Submitted to the Department of Electrical Engineering and
Computer Science

in partial fulfillment of the requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2006

© Massachusetts Institute of Technology 2006. All rights reserved.

Author
Department of Electrical Engineering and
Computer Science
August 11, 2006

Certified by
Polina Golland
Assistant Professor, Department of Electrical Engineering and
Computer Science
Thesis Supervisor

Accepted by
Arthur C. Smith
Chairman, Department Committee on Graduate Students

CellVisualizer: Exploring Hierarchical, Multi-Dimensional Data with Applications to High-Throughput Microscopy

by

InHan Kang

Submitted to the Department of Electrical Engineering and
Computer Science
on August 11, 2006, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In this thesis, we present a system for visualizing hierarchical, multi-dimensional, memory-intensive datasets. Specifically, we designed an interactive system to visualize data collected by high-throughput microscopy and processed by CellProfiler, an open-source system jointly developed by researchers at MIT CSAIL and the Whitehead Institute.

A typical high-throughput microscopy experiment produces thousands of images, with thousands of objects in each image. CellProfiler then measures hundreds of features for each cell, nuclei, and cytoplasm. In contrast to previously demonstrated visualization software, our system visualizes datasets that are on the order of hundreds of gigabytes, datasets too large to store in physical memory. We also implement tools to link the dataset to available resources such as online genetic databases and the actual images acquired by the microscope. Finally, we demonstrate how the system was used to highlight interesting genes for more detailed analysis in real biological studies.

Thesis Supervisor: Polina Golland

Title: Assistant Professor, Department of Electrical Engineering and Computer Science

Acknowledgments

Firstly, I would like to thank my advisor, Professor Polina Golland, for her guidance and support throughout this project. Her words of advice, generous funding, and help throughout this project helped ensure both this paper and this system would be completed to the best of my abilities. I also would like to thank Ray Jones for his advice and ideas and Anne Carpenter for her understanding and willingness to work with me through every step. Next, I would also like to thank Dr. David Sabatini and the rest of the members in the Sabatini Lab for giving me the opportunity to be part of the CellProfiler team. I would also like to thank Adam Papallo of the Sabatini Lab for taking the time to edit my thesis, to learn the system, and to take on the responsibility of furthering this project in the future. Finally, I want to thank my friends and family who have been by my side for the past year and a half as I worked on this project.

Contents

1	Introduction	12
1.1	Applications of Visualization Software	13
1.1.1	Validation of Images	13
1.1.2	Validation of Segmentation Algorithms	14
1.1.3	Excluding Artifacts	15
1.1.4	Verifying and Determining Gene Functionality	15
1.2	Challenges of High-Throughput Data Visualization	16
1.3	The CellVisualizer System	17
1.4	Roadmap	17
2	Background: High-Throughput Microscopy Analysis	19
2.1	Image Microscopy: Gathering the images	19
2.2	CellProfiler: Quantifying the image results	20
2.2.1	The Data in Detail	21
2.3	Visualization Requirements	22
3	Background: Common Multi-Dimensional Visualization Techniques and Software Systems	24
3.1	High-Dimensional Visualization	24
3.2	Available Tools	25
3.2.1	Commercially Available Tools	25
3.2.2	Publicly Available Tools	28

4	The CellVisualizer System	30
4.1	Overview	30
4.2	Available Plots	31
4.3	Brushing	32
4.4	Displaying Detailed Information	34
4.4.1	Linking To Images	35
4.4.2	Linking To Web Databases	36
4.4.3	Retrieving All Measurements	37
4.4.4	Plotting Per Object Plots	37
4.5	System Architecture Overview	37
4.6	Database Layer	37
4.7	Internal Data Structures	39
4.7.1	The Data Model	39
4.7.2	Data Strategy	42
4.7.3	Plots	43
4.7.4	Brushing	44
4.8	Performance Considerations	44
4.8.1	In-Memory Database	44
4.8.2	Memory Constraints on Plots	45
4.9	Summary	46
5	Example Extension - Cell Classification	48
5.1	Cell Classifier	48
5.1.1	Linking From Plots in CellVisualizer	49
6	Case Studies	53
6.1	Finding Genes that Regulate Mitosis	53
6.1.1	Quality Control	53
6.1.2	Plotting Measurements of Interest	54
6.1.3	Finding Interesting Genes	55
6.1.4	Getting More Detailed Information	56

6.2	Comparing Two Different Knockdown Experiments	59
6.2.1	Grouping Replicate Slides	60
6.2.2	Plotting Measurements of Interest	60
7	Discussion	62
7.1	Other Applications	62
7.2	Future Work	63
7.3	Conclusion	63

List of Figures

1-1	Image Analysis Sequence	13
1-2	Image Distortation	14
1-3	Scatterplot with/without errors	15
2-1	Microscopy Image Gathering	20
2-2	Fly203 database schema	22
3-1	Analysis using Excel	26
3-2	GGobi Plots	29
4-1	Types of Plots Available in CellVisualizer	33
4-2	Brushing in CellVisualizer	34
4-3	Show Detailed Information Menu	35
4-4	Image Collection Architecture	36
4-5	Simplified System Diagram	38
4-6	Group Tables UI	41
4-7	Data Retrieval Strategies	42
4-8	Class Hierarchy of Display Models	43
4-9	Class Hierarchy of Display Models	44
5-1	Cell Classifier	49
5-2	Using 2-D Histograms & Classifiers Together	51
6-1	Quality Control: Checking for artifacts	54
6-2	Cell Area vs. Nuclei Area	55

6-3	Finding Interesting Genes	56
6-4	Finding Interesting Genes	57
6-5	Linked Images and Web Information	58
6-6	Per Object Plots	59
6-7	Grouped Experiments	60
6-8	Plotting Experimental Conditions	61

List of Tables

4.1	Example Data Model Object	40
4.2	Time taken to retrieve data to display in a scatterplot	45

Chapter 1

Introduction

Biologists have been analyzing cells using microscopes to study cellular function for centuries. Recent advances in automated microscopes have enabled biologists to acquire thousands of high-resolution images, each representing a different gene knockout experiment, in a relatively short amount of time. With the development of creative new delivery techniques, full genome-scale RNAi-based screens can be performed in a few days [15]. As shown in Figure 1-1, these high-throughput microscopy screens require software to both analyze the images and visualization tools to explore the results. CellProfiler, developed through the collaborative efforts between The Whitehead Institute for Biomedical Research and MIT Computer Science and Artificial Intelligence Laboratory (CSAIL), is an example of a system designed to quantify the information captured in microscopy images [2]. CellProfiler examines each image and stores the measurements in a database.

However, a bottleneck exists once the measurements are stored in a database. Currently, there is no effective way to extract meaningful information from these measurements. There is no system in place to visualize this information nor a way to interface these measurements to other outside resources. Interactive, user-friendly, visualization software is necessary to make these measurements meaningful and useful to biologists for exploratory analysis of high-throughput data. Since this software is intended for the average biologist to use, it is important that the system be available on both the Windows and the Macintosh platforms and be functional on a personal

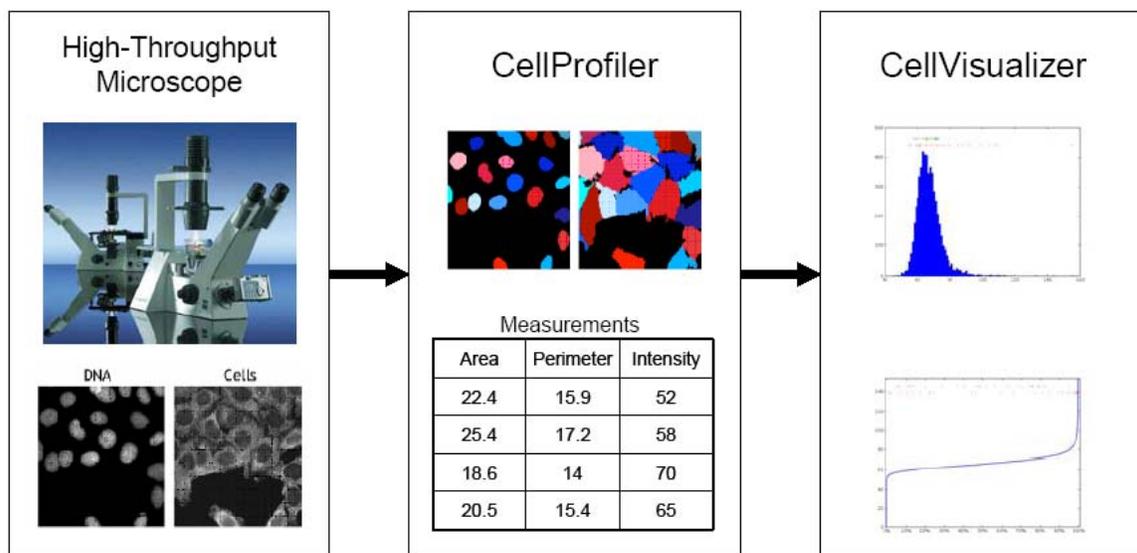


Figure 1-1: Biologists acquire thousands of images using a high-throughput microscope. CellProfiler performs quantitative measurement based on the images. CellVisualizer is then used to interactively explore and analyze the data.

computer with typical amounts of memory.

1.1 Applications of Visualization Software

Visualization of image and cell measurements from high-throughput experiments is important for many reasons such as validating that the images were taken correctly - the image may be corrupted for a variety of reasons, validating that the image segmentation algorithms work as expected, excluding errors due to incorrectly measured objects, verifying results for known controls, and allowing users to interactively explore the data, leading to the discovery of previously unknown gene functions, as well as unexpected relationships in the data.

1.1.1 Validation of Images

The preparation of samples and the acquisition of images is an automated process. The image data is often corrupted because of errors in these steps. Common errors include the presence of fibers or, as shown in Figure 1-2, blobs of dye in the image.

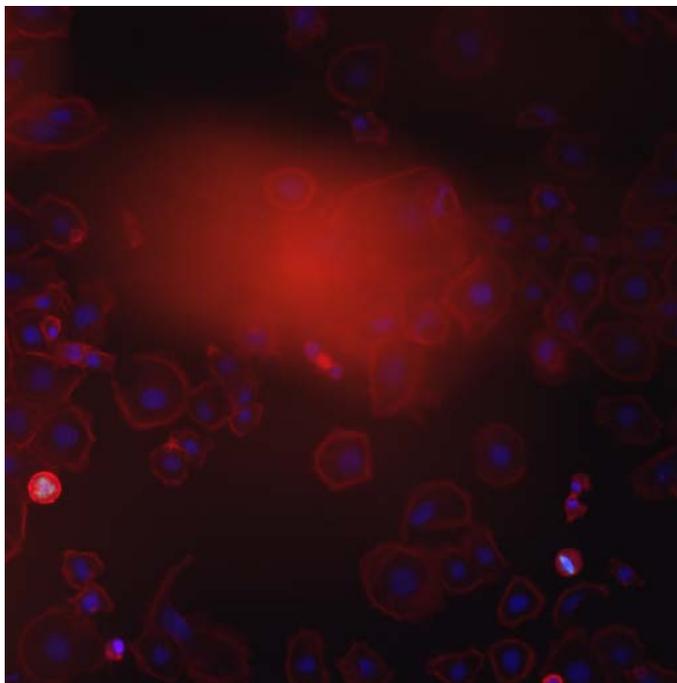


Figure 1-2: CellVisualizer allows scientists to check whether the measurements are correct or due to errors in the sample preparation or imaging process. Here is an example where a blob of dye occludes much of the image.

Such abnormalities distort the measurements in the image because they tend to be much bigger and brighter than the cells being measured. Abnormalities in an image not only occludes legitimate cells but typically also cause the segmentation algorithm to fail, thereby producing incorrect measurements.

1.1.2 Validation of Segmentation Algorithms

Cell measurement data can also be inaccurate because the segmentation algorithm may have performed poorly. Since automatic segmentation of cells remains a difficult problem [9], the users need to validate that interesting points in the dataset correspond to actual cells in images and not to errors in a complex segmentation algorithm. As before, a visualization tool that links to external resources, such as images showing the cells outlined by the segmentation algorithm, provides a much higher level of confidence in the quality of measurements.

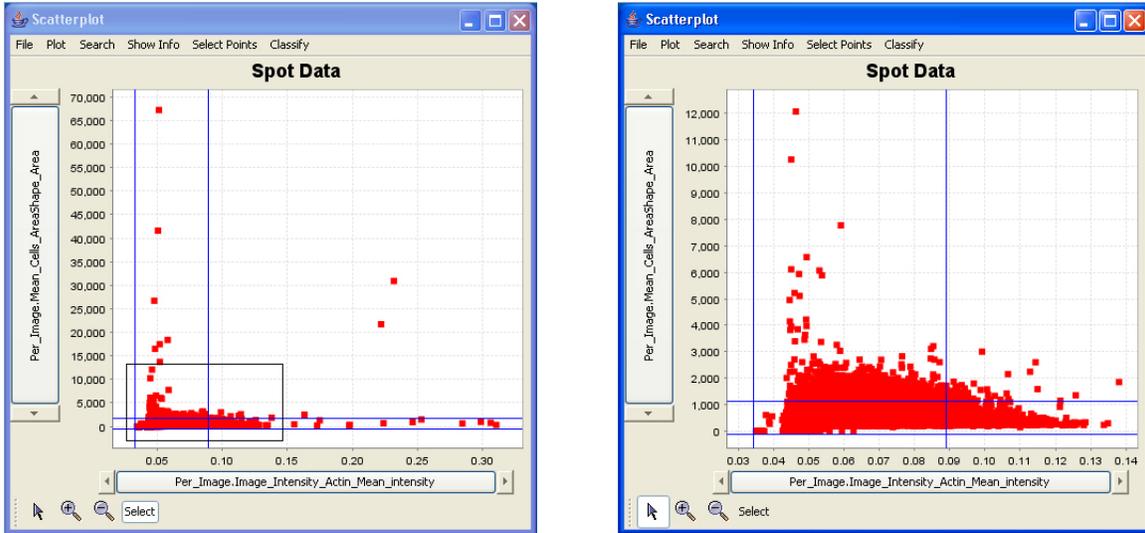


Figure 1-3: Scatterplots of the same experiment including and excluding errors such as blobs of dye. We determine which datapoints have errors by opening images that were measured to have significantly high actin intensity (x-axis) or large cell size (y-axis) and checking to see if these images have any abnormalities. The blue lines indicate which data points are above or below two standard deviations from the mean.

1.1.3 Excluding Artifacts

Since it is nearly impossible to produce a completely error-free dataset, it is important to recognize and exclude errors when examining the data. It is possible that errors in a handful of samples do not necessarily compromise the integrity of the entire experiment. Allowing the users to identify and exclude errors and concentrate on the data that has not been corrupted is crucial to the data discovery process. Figure 1-3 shows how we exclude images which are likely to be corrupted by artifacts since the actin intensity in these images is abnormally high. The blue lines indicate

1.1.4 Verifying and Determining Gene Functionality

Ultimately, a visualization tool facilitates the data-exploration process for biologists. The end goal of this process would be to either verify known gene functionality or find interesting, previously unknown, gene functionalities. As we illustrate in the later chapters, CellVisualizer facilitates this process by providing tools for interactive browsing of the dataset.

1.2 Challenges of High-Throughput Data Visualization

The visualization and exploration of high-throughput microscopy data is a challenging problem because datasets tend to be large and because the natural relationships within these dataset must be captured.

The datasets are often tens of gigabytes, well outside the range of what can be stored in physical memory of a typical computer. Further, it is beyond the human capacity to synthesize and interpret such large amounts of data. A robust visualization system would balance a user's requirements of speed and completeness - allowing users to visualize the entire dataset, albeit slowly, if required, and also allowing users quick access to various summaries of the data.

The natural relationships within the dataset also make data visualization a challenging problem. The dataset that is being analyzed has two levels of hierarchical data - "Per Image" data and "Per Object" data. The "Per Image" data stores measurements and information common to a particular image, such as the total intensity of the image, the mean cell area averaged across all cells in the image, and the gene that corresponds to the image. The "Per Object" data stores measurements for each object (i.e. cell) that was measured in each image, such as the area of each cell or the intensity of the staining in each nucleus.

Moreover, biologists often run the experiments under nearly identical situations, changing just one variable, such as which gene was co-knocked down. For example, microscopy experiments can be done where the TOR gene is knocked down in the first set of images and the PTEN gene is knocked down in the second set of images. Since all other conditions are the same, it is worthwhile to consider the differences in gene expression between the TOR knockdown experiment and the PTEN knockdown experiment. In this case, the database would have two sets (TOR and PTEN) of two levels ("Per Image" and "Per Object") of hierarchical data.

1.3 The CellVisualizer System

CellVisualizer is designed to be a complementary tool to CellProfiler and thus primarily addresses the concerns of CellProfiler users. Still, the design was carefully considered to be applicable to other problems that involve two levels of related, hierarchical data. For example, it would be possible to use CellVisualizer to analyze datasets of baseball players, on both the “Team” and “Player” level. CellVisualizer provides users with interactive visualization functionality necessary to quickly and effectively explore data. Standard visualization tools are provided, including histograms, scatter plots, and parallel coordinates.

In addition to the standard tools, CellVisualizer tightly integrates solutions for the particular exploration approaches relevant to high-throughput microscopy data. Users can build plots both on the image level and on the object (cell/nucleus) level. Users can also plot measurements from different experiments, images, or cells on the same figure to quickly compare results from these plots. When a user clicks on a data point, the point is highlighted in every open plot. Relevant statistics of that data point, the corresponding image, as well as gene information from a website of the data point is shown on demand. Specific cell sampling techniques are available to identify interesting genes.

We also designed CellVisualizer to be usable on any desktop, be it a Windows, Macintosh, or Linux machine, with typical amounts of memory. CellVisualizer manages the numerous constraints in plotting massive dataset on the order of several tens of gigabytes. Users can store several highly used columns in the “in-memory” database while still having access, albeit more slowly, to the rest of the data in the database.

1.4 Roadmap

Chapter 2 begins with an overview of the problem and the requirements of a system designed to analyze high-throughput microscopy data. Chapter 3 discusses common

multi-dimensional visualization techniques and existing visualization software. Chapter 4 describes our system, CellVisualizer, and its architecture in detail, especially in the context of the previous chapters. Chapter 5 discusses possible extensions to CellVisualizer, including a classification learner that has been incorporated into the system. Chapter 6 presents case-studies of our system in use, and demonstrates how CellVisualizer enables validation and discovery of important gene functions. Finally, chapter 7 presents a discussion of the work.

Chapter 2

Background: High-Throughput Microscopy Analysis

To fully understand the visualization problem addressed in this work, it is important to understand the dataset produced in high throughput microscopy experiments. This chapter provides an overview of how these experiments are conducted and how we would like to analyze the resulting image data.

2.1 Image Microscopy: Gathering the images

Using powerful RNA interference (RNAi) techniques developed in the past few years, we are able to perform full genome-wide RNAi-based screens using both *Drosophila* (fruit fly) and human cells. Figure 2.1 details one such procedure for preparing samples and gathering images from microscopy experiments.

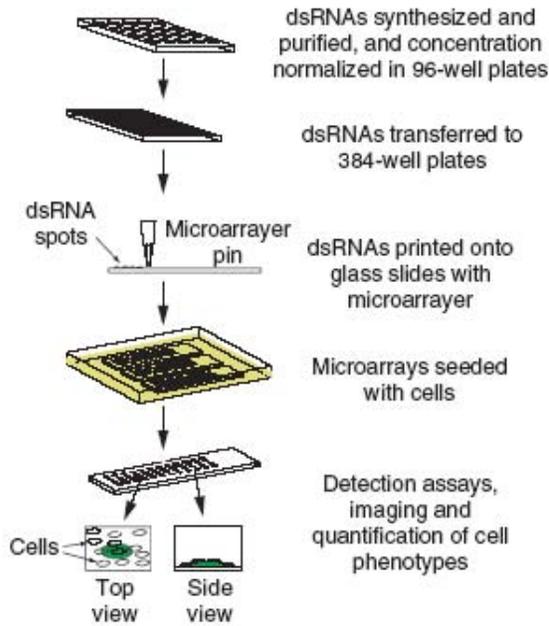


Figure 2-1: Schematic of procedure for the fabrication, cell seeding, and assaying of *Drosophila* RNAi cell microarrays.[15] This process allows biologists to perform full genome wide screens relatively quickly.

This process is very powerful because we can print arrays that allow testing thousands of gene knockdowns, even up to the full genome. We print these arrays by synthesizing and purifying thousands of RNAi reagents, each carefully designed to knockdown one particular gene. These reagents are then placed in different wells of multiwell plates, or spots in a cell microarray. The microscope typically takes 1-6 images within each well, capturing hundreds of cells. Examining these images by eye, or exploring cell measurement data acquired from the images can reveal how a particular gene-presence or gene-knockdown affected cells.

2.2 CellProfiler: Quantifying the image results

CellProfiler is cell image analysis software developed jointly by the Sabatini Lab at the Whitehead Institute and Polina Golland’s laboratory at MIT CSAIL [2]. CellProfiler reads each image taken by the microscopy imager and measures various features for each cell and/or sub-compartment (such as nuclei and cytoplasm), including the area, location, shape, intensity, texture, and interesting measurements published in

the literature [2]. As CellProfiler is designed to measure many properties of cellular objects, the number of measurements the system stores typically numbers in the hundreds.

In addition to the individual measurements for each cellular object in the images, there are measurements and information for each captured image. Such “Per Image” information includes unique gene and website identifiers for each image, the location of the actual image files that produced the measurements, and certain numerical values measured by CellProfiler, such as the average intensity of the DNA staining across the image.

2.2.1 The Data in Detail

As mentioned in Chapter 1, multiple slides are often imaged under nearly identical conditions while manipulating parameters on the image set level, such as knocking down different genes in the entire image set. That is, one particular gene is knocked down in all images of an image set in addition to specific genes being knocked down in each image. In such an experiment, there is a set of slides corresponding to each gene knockdown, and the conditions in image n of one set of slides is identical (except for the gene knockdown) to the conditions in image n of another set of slides. Furthermore, to ensure the integrity and reproducibility of the data, the knockdowns are often replicated several times. For example, in a recent experiment at the Whitehead, image sets 1-3 knocked down the GFP protein, negative control image sets 4-6 knocked down the PTEN protein, image sets 7-9 the TSC1 protein, and image sets 10-12 the TSC2 protein. This type of experiment elucidates how the knockdown of one gene affects cellular phenotypes compared to the knockdown of another gene.

This type of experiment creates one “Per Image” table and one “Per Object” table for each gene knockdown image set in the database. In the example described above, there are 12 distinct “Per Image” and 12 distinct “Per Object” tables. The database structure is identical in each of the “Per Image” and “Per Object” tables. A partial schema of one such experiment is shown in Figure 2-2. It is important to note that the number of rows is identical in each of the “Per Image” tables - each row

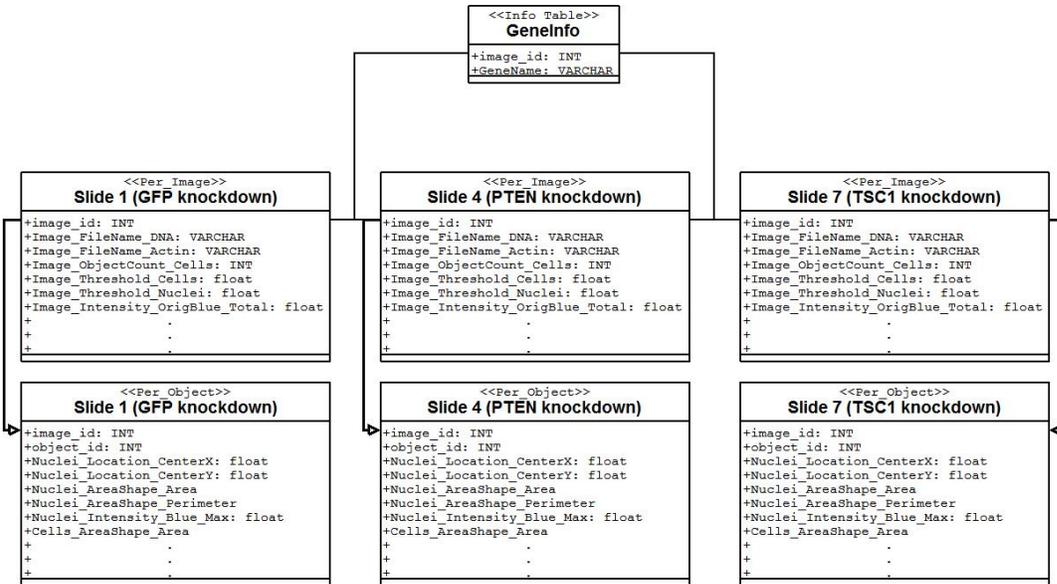


Figure 2-2: Partial schema of the Fly203 database.

captures measurements for a particular image and each image set captures the same experimental conditions and thus the same number of images. On the other hand, the number of rows in each “Per Object” table varies for each image set because the number of rows depends on how many objects were recognized in each image within the image set.

Finally, to appreciate the magnitude of the dataset that the visualization system must handle, let’s examine a sample dataset. The NIRHT dataset is made up of 43,000 images, with an average about 200 cellular objects for each image. In total, CellProfiler stored 468 measurements for each of these 7,500,000 objects. Since each of these measurements is stored as a float (4-bytes) in a database, the dataset we are measuring is over 14 gigabytes.

2.3 Visualization Requirements

The most fundamental requirement of any data visualization and exploration tool is the ability to plot the data in a variety of different ways. Histograms and scatterplots are among the most basic and useful ways to plot data. Additionally, since every

measurement in a “Per Image” table is produced from a single image and every measurement in a “Per Object” table is produced from a single cellular object, it is important to link each of the measurements back to the image or object that it came from.

Linking to Images

One of the most important requirements for a visualization system designed for high-throughput microscopy experiments is the ability to view the actual images that were acquired by the imager. It is typical in these experiments for multiple images to be collected - one image may correspond to nuclei staining, another image to cell staining, and yet another image to an additional protein of interest. After analyzing and extracting measurements from these images, CellProfiler stores the path and filenames in the “Per Image” table of the database. It is necessary to be able to retrieve these images and confirm by eye the results of segmentation and measurement steps performed by CellProfiler.

In addition to storing the path and filename information for the images, CellProfiler also stores where in each image every object it measures is located in the “Per Object” tables. This information can be used to locate a specific cell/nucleus in the dataset.

Linking to Internet Resources

It is also important to leverage the power of the Internet by linking experiments to available gene databases and look up known information about each gene. For example, FlyBase is a comprehensive database for information on the genetics and molecular biology of *Drosophila*[6]. Similar databases exist on the Internet for other genomes.

Chapter 3

Background: Common Multi-Dimensional Visualization Techniques and Software Systems

There has been a lot of research in the visualization of multi-dimensional datasets. The challenge is to present multiple dimensions on a computer screen, a two-dimensional surface, in a way that is easily comprehensible. This challenge becomes even more difficult when dealing with massive datasets, as in the case of some high-throughput experiments, because the computer screen is limited by the number of pixels available. In this chapter, we discuss existing techniques for displaying and exploring multi-dimensional datasets and some available software systems.

3.1 High-Dimensional Visualization

In this section we discuss two methods for displaying and exploring multi-dimensional data on a two-dimensional computer screen: brushing and parallel coordinates.

Brushing

First described by researchers Richard A. Brewer and William S. Cleveland at Bell Laboratories[1], brushing is the ability to allow the user to move a region (a brush)

around the data display to highlight groups of data points. When multiple plots of the same dataset are open at the same time, the brushing of points in one plot highlights the corresponding subset of points in all the other plots. This technique allows the user to more easily understand data and see relationships across multiple dimensions when the data has a large number of attributes or items, when the data spans multiple tables or data types, or when it is natural to analyze different parts of the data using different views. We extend the concept of brushing in CellVisualizer so when users highlight a point that corresponds to one particular image, all points that correspond to similar experimental conditions would also be highlighted regardless of what table the data comes from.

Multi-Dimensional Plots

There have been a lot of developments over the past twenty years in viewing multiple dimensions on a 2-dimensional output. One common approach is to use color/shape to visualization additional dimensions in a simple two-dimensional scatterplot. However, other techniques must be used to visualize more than 3 or 4 dimensions. The most notable is parallel coordinates, proposed by Inselberg in 1981, where all axes are parallel to another and equally spaced apart [7]. By drawing axes parallel to each other, one can represent multi-dimensional points. CellVisualizer users are able to visualize several dimensions at the same time using the parallel coordinates.

3.2 Available Tools

As part of this work, we researched currently available software systems to determine the viability of their use in the exploration of high-throughput microscopy data.

3.2.1 Commercially Available Tools

The simplest way to analyze the data is to export the measurements into an Excel spreadsheet. Excel's charting tools allow users to graph histograms and scatterplots of the measurements. Users can also generate hyperlinks using Excel. By copying and

pasting gene information to a separate column, gene information can be hyperlinked from a spreadsheet to known gene databases available on the Internet.

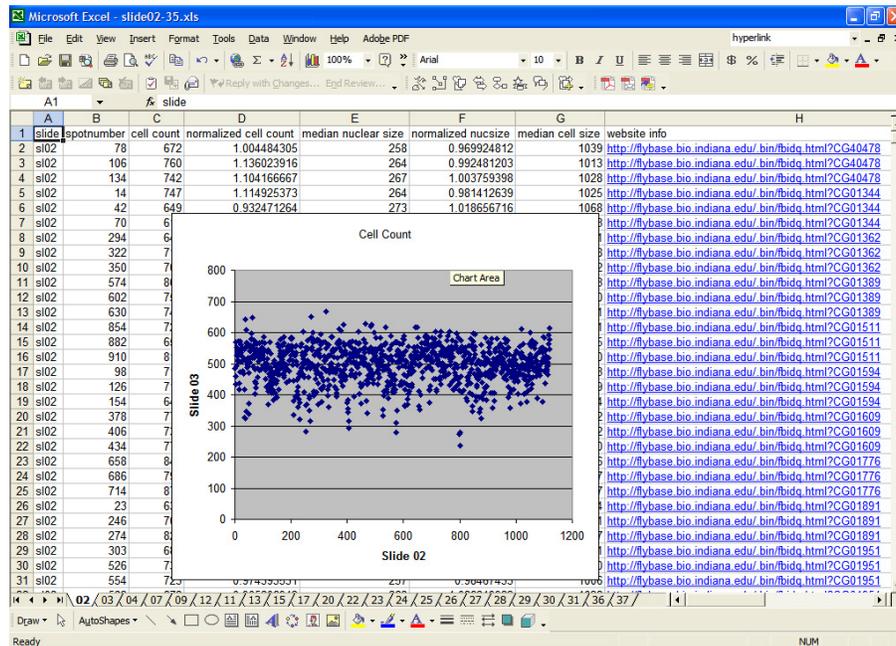


Figure 3-1: Researchers at the Whitehead initially explored the data using Excel spreadsheets.

Although Excel is a very powerful data analysis tool, there are several limitations when using it for interactive data visualization. Users cannot truly interact with their data - it takes a long of time to generate the graphs. Changing axes, gating (selecting on a particular subset of the original dataset), and finding information about a selected point is tedious and time-consuming. Another shortcoming in the current version of Excel are the limits on the number of rows and columns we can view. Finally, hierarchical data is not linked. “Per Image” information cannot be presented alongside “Per Object” data because it is not possible for Excel to know the links between the information. One way to bypass this particular limitation in Excel is to import all the data into an Microsoft Access database, create links, and then copy the generated datapoints back into Excel and plot this information. Again, it is easy to see why such a process is not conducive to interactive data exploration.

Spotfire[13] is one of the most popular commercial visualization software available for high-dimensional visualization. The visualization tools in Spotfire are impressive

yet there are several critical shortcomings when analyzing our data. Spotfire has the ability to visualize information stored on a database but its mechanism for doing this is to load all the data into memory. Since the amount of memory on a typical computer is typically less than 2 GB, it is impossible to import our entire NIRHT dataset into memory. Exceptionally large datasets (1,000,000 records or more) are cumbersome, because they take a long time to load, require a lot of memory, and do not respond instantly to queries [5]. One option that Spotfire users have is to sample the database [5]. This solution partially suffices for some of our exploratory purposes since we are usually looking for general trends in the dataset as opposed to a single outlier cell out of the 7 different million cells. Unfortunately, there are times when we would like to consider the entire dataset, such as when we want to know how many objects match a particular phenotype in each image. Another disadvantage of Spotfire is that there is no way to represent the inherently hierarchical nature of our dataset. Like Excel, it does not allow us to link the top-level “Per Image” information to the lower-level “Per Object” information. Since Spotfire provides the ability to aggregate information in a dataset, one workaround would be to redundantly copy all the top-level data for all the entries in the lower-level. While possibly useful in smaller datasets, this approach is inadequate because the amount of redundant information being copied will multiple hundreds of gigabytes worth of information. The final limitation of SpotFire is the cost. A one year Spotfire license costs thousands of dollars; the power of having free, open-source software to compute quantitative measurements, such as CellProfiler, is diminished if it costs thousands of dollars to view the results.

Other commercially available tools include software that come bundled with robotic microscopes, e.g., Cellomics [3]. There are two main problems with such software. Annual licensing fees costs tens of thousands of dollars. Bundled software often only works for images produced by the corresponding imager. Not only does this limit the type of experiments we can run to those that conform to the specifications of the imager, but it also does not allow us to leverage new analysis tools, such as CellProfiler.

3.2.2 Publicly Available Tools

There are several publicly available tools for visualizing multi-dimensional data; GGobi and VisDB are two such tools. The capabilities of these systems tend to be limited and lack functionalities essential for viewing and analyzing high-throughput microscopy data.

GGobi is the most well-known open source multi-dimensional data visualization system available. GGobi supports several types of plots: scatterplots, parallel coordinate plots, scatterplot matrices, and time-series plots [14], as shown in Figure 3-2. It also supports interactivity between several plots by providing “brushing” capabilities.

Though GGobi provides the basic tools for viewing any general multi-dimensional data, it doesn’t have some of the functionality helpful for the exploration of biological data. There is no way to extract and display the hierarchical nature of the data, nor to compare experiments of the same gene against each other, nor to group the data in a meaningful way. GGobi does not provide interaction with images or available gene databases. In addition, GGobi has the same memory limitations as SpotFire; data must be loaded completely into the memory before it is visualized. Since users cannot predict ahead of time which measurements will be useful when exploring the data, confining the users to the limitations of system memory inhibits flexibility.

VisDB [10] represents a class of freely available research tools. Such tools have been developed more to serve as experimental tools to help computer scientists evaluate the effectiveness of certain visualization techniques rather than to serve the end-user. Typically they have limitations such as highly specific data formats or particular platforms. VisDB, for example, works only on Linux/HP. Though useful in benchmarking of visualization analysis, such tools have limited practical use for researchers interested mainly in data exploration.

To summarize, though there exists both open-source and commercial software that allow users to visualize multi-dimensional data, the available systems do not provide all the functionality necessary to adequately explore the datasets produced by

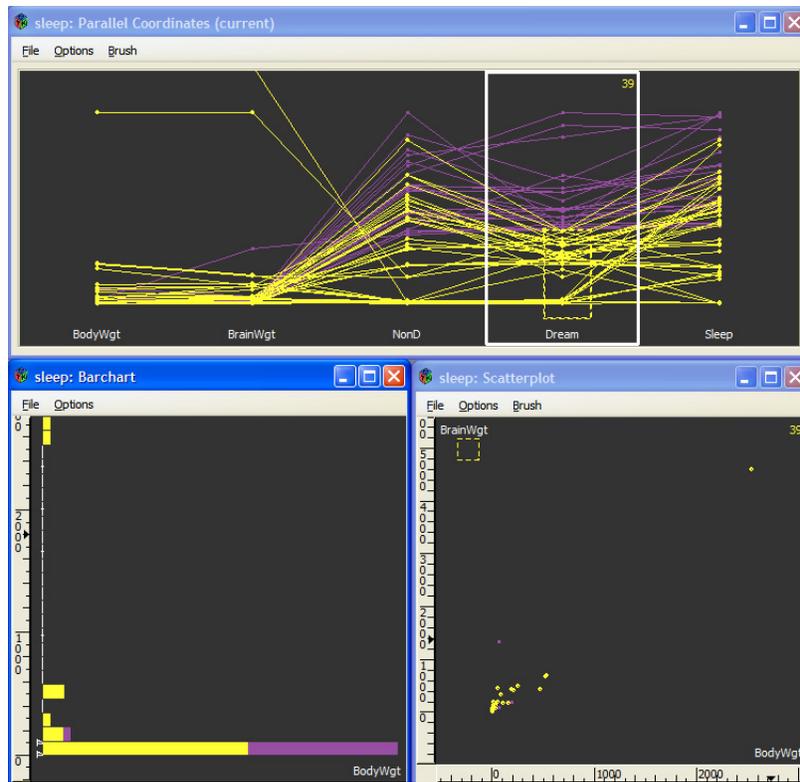


Figure 3-2: GGobi is an open-source multi-dimensional visualization tool. Scatterplots, bar charts, and parallel coordinates are some of the types of charts that can be plotted. It also includes interactive capabilities such as brushing.

CellProfiler.

Chapter 4

The CellVisualizer System

This chapter discusses the details of the CellVisualizer system and explains the design choices for the internal structures and the user interface.

4.1 Overview

The CellVisualizer system is designed to plot two levels of hierarchical data. In high-throughput microscopy screens, the two levels of data are “Per Image” measurements and “Per Object” measurements.

All the “Per Image” measurements are stored as columns in a “Per Image” table; each row in the “Per Image” table corresponds to one particular image. Each row is uniquely identified by its ImageNumber. In addition to the ImageNumber and the measurements, each row also contains auxiliary information such as where the images taken by the microscope reside in the file system.

The “Per Object” table contains all the data for individual cellular objects. The unique identifier in the “Per Object” table is the ImageNumber/ObjectNumber pair. As in the “Per Image” table, the columns in the “Per Object” table hold the measurements and each row corresponds to one cellular object.

High-throughput microscopy screens are often replicated several times to ensure the integrity of the data. As explained in section 2.2.1, experimental conditions may also be altered slightly to test how cells react when one factor changes; one particular

gene, such as PTEN, may be knocked down in every single image in that screen. In these cases, there would be several “Per Image” and “Per Object” tables, one “Per Image” and one “Per Object” table for each screen. Since experimental conditions are unchanged (except for the gene that is knocked down everywhere), the gene that is knocked down in image n in one screen would be the same gene knocked down in image n in every other screen.

Programming Language

CellVisualizer is intended to be a software tool used to visualize high-throughput microscopy data. Since scientists interested in high-throughput data use both Windows and Macintosh, CellVisualizer is designed with cross-platform operability in mind. To achieve this goal, CellVisualizer is written in Java.

4.2 Available Plots

Users can plot four different types of plots using CellVisualizer: histograms, scatterplots, 2-D color histograms, and parallel coordinates. Since there are hundreds of possible measurements to plot and these plots are only capable of displaying a handful of these measurements at one time, users must select which measurements they are interested in viewing when selecting a particular plot.

Histograms

Histograms are useful for viewing the distribution of one particular measurement. In Figure 4-1, we show a histogram of all the measurement “Mean_Cell_AreaShape_Area” in the “Per Image” table. The blue line indicates what a gaussian normal distribution looks like for the histogram. We can see that this measurement seems to be normally distributed across all the different images in the experiment.

Scatterplots

Scatterplots are useful for viewing two measurements against each other. One limitation of scatterplots is that the data points occlude each other when the dataset becomes too large. Scatterplots are most useful for “Per Image” plots where the number of datapoints is in the order of thousands. We plot the Average Nuclei Area against the Average Nuclei DNA intensity in Figure 4-1.

2-D Histograms

When there is a large number (on the order of hundreds of thousands or more) of data points, 2-D color histograms are the best plot to view two measurements against each other. In this case, every pixel represents a “bin” and the number of objects in the “bin” is indicated by the color of the bin. 2-D histograms are useful for plotting “Per Object” measurements because each table contains millions of “Per Object” measurements. In Figure 4-1, we plot the Nucleus Area against Nucleus DNA Intensity for every single object in the dataset.

Parallel Coordinates

As discussed in section 3.1, parallel coordinates is one technique for displaying multiple dimensions on a flat surface. In Figure 4-1, we plot Average Cell Intensity, Average Cell Size, Average Nuclei Size, and Average Nuclei Intensity against each other.

4.3 Brushing

Brushing in CellVisualizer incorporates the idea that the gene knocked down in image n in one table is the same gene knocked down in image n in every other “Per Image” table. Therefore, when a user highlights a points in one particular plot, CellVisualizer highlights all the corresponding points in every other plot, regardless of what table the data comes from. In Figure 4-2, though the data comes from 5 different tables (the

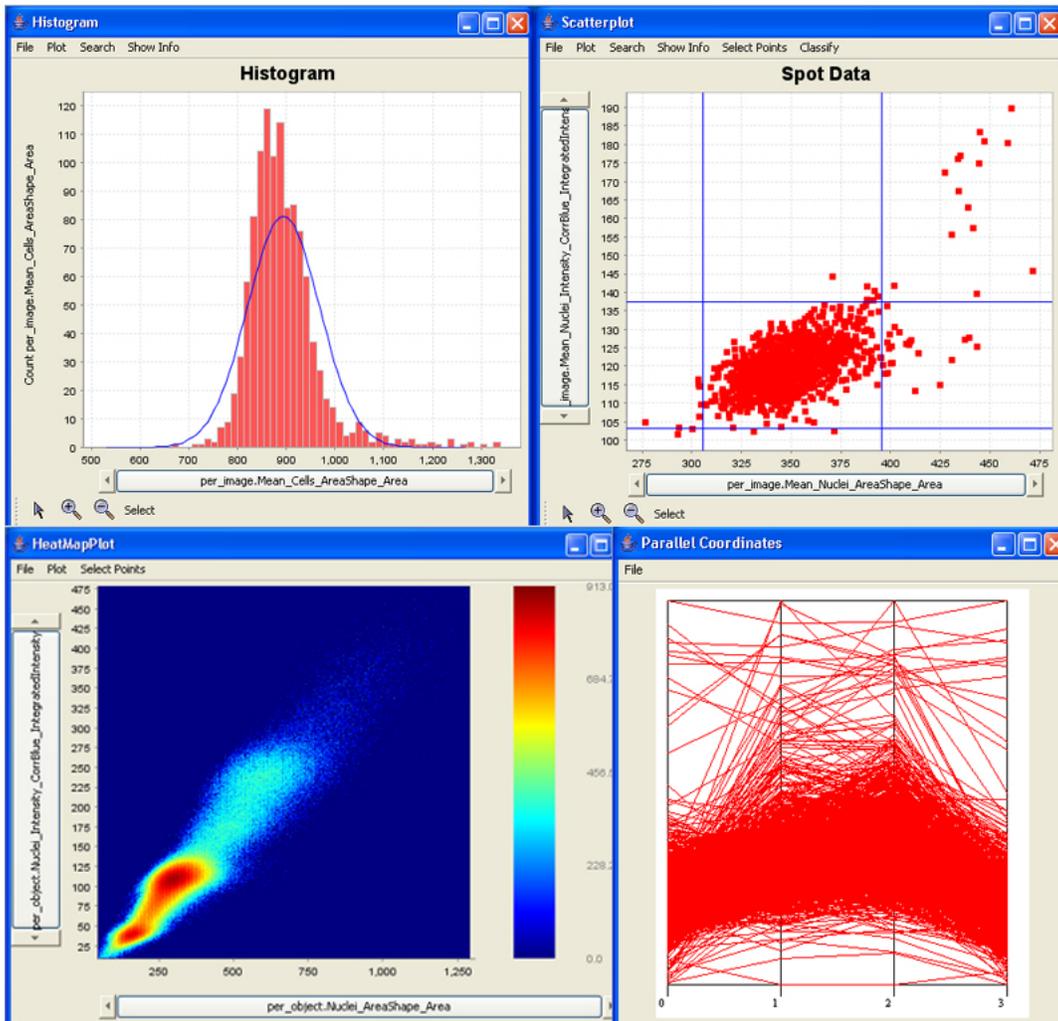


Figure 4-1: There are four types of plots available in CellVisualizer: histograms (top left), scatterplots (top right), 2-D histograms (bottom left), and parallel coordinates (bottom right).

scatterplot on the left plots average cell area per image in slide 2 against average cell area in slide 3, the scatterplot in the middle plots average cell area in slide 4 against average cell area in slide 5, and the histogram on the right plots the distribution of average cell area per image in slide 6), highlighting points in one plot highlights all the corresponding points in all the other plots.

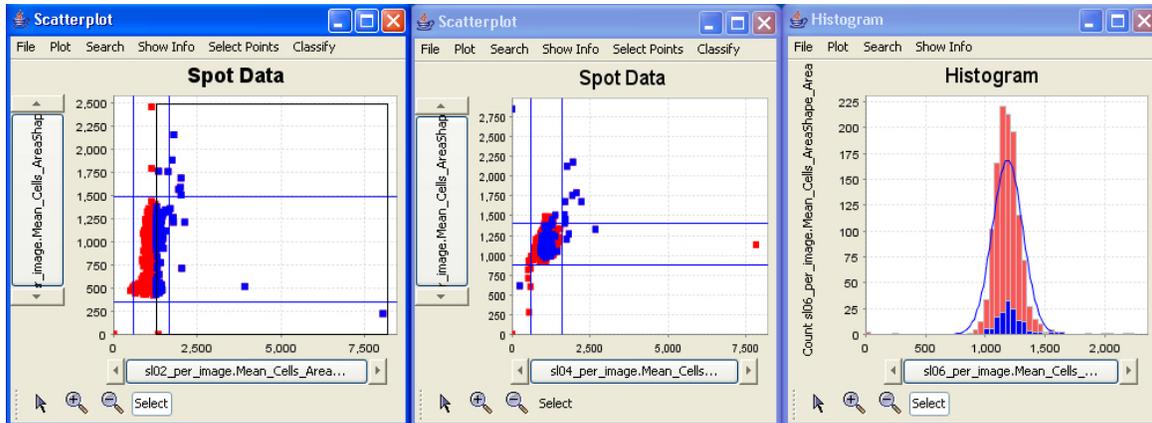


Figure 4-2: Highlighting certain points in one plot highlights the corresponding points in all the other plots. In this example, we highlighted all points with big average cell size in slide 02 and the corresponding points in the scatterplot in the middle and the histogram on the right are also highlighted.

4.4 Displaying Detailed Information

An important feature of CellVisualizer is not only the ability to plot the data, but also to view images and other detailed information once the user discovers interesting phenomena in the plots. This section describes how the system links the plots to images and other information.

The user may request different types of detailed information about a particular point. Currently, the user can select data points and request the images from which the data points came, information about the gene that produced the data points from a web database, all the measurements recorded for the image/object that produced the data point, and if the data points are from “Per Image” measurement, a “Per Object” plot of measurements for all the objects in the images that produced the “Per Image” points.

In the menu bar for each plot, we display the different detailed information available, as shown in Figure 4-3. Each plot stores information about which tables and measurements it has plotted and thus can link back to the tables to extract detailed information.

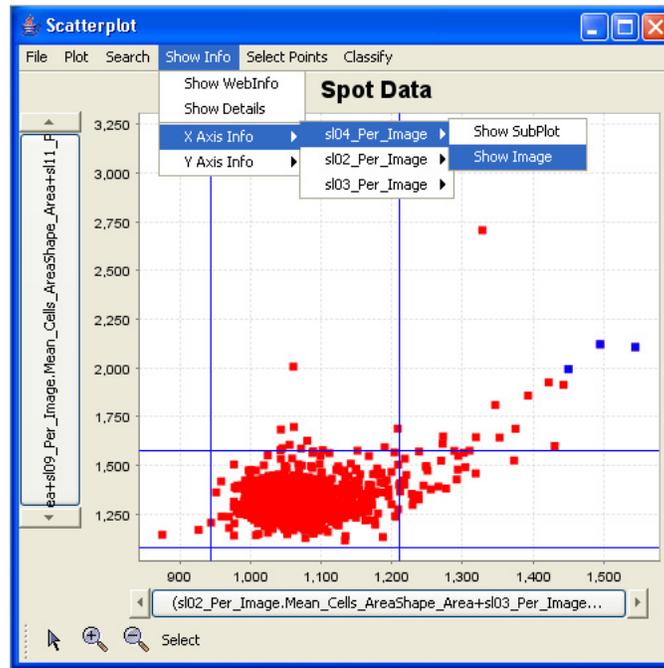


Figure 4-3: Access to detailed information relevant to a displayed plot.

4.4.1 Linking To Images

The “ShowImage” object is capable of displaying 3 grayscale images and overlaying them to create one RGB image. The GUI also provides users the ability to toggle the 3-channels on and off and the ability to locate single or multiple cells within the image. In order to display an image, the image files must first be retrieved from the database. As Figure 4-4 shows, the CellVisualizer system handles the retrieval and displaying of images on three distinct levels.

The first level is responsible for retrieving the image file stream from the remote file server. Currently, the CellVisualizer system retrieves a file stream from images on the local file system and images on the Internet. It can also remotely connect to Samba file servers and retrieve file streams with the help of the Java CIFS open source client library [4]. We have also integrated SSH2 connections to CellVisualizer by integrating the open source JSCH library [8]. Currently, input streams to SSH2 connections are established by copying a remote file to a temporary local directory and opening a stream to the local file. By isolating the fetching of the file streams from

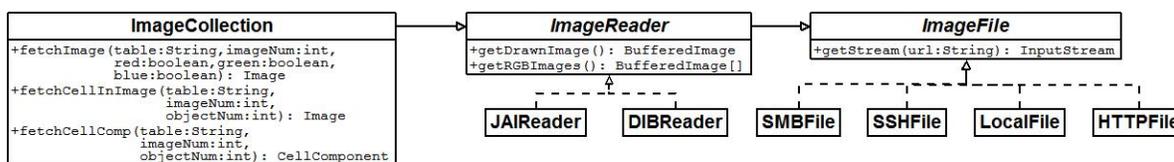


Figure 4-4: Unified Modeling Diagram (UML) diagram of objects involved in retrieving and displaying image files.

the rest of the system, the system can be easily extended to use other file-fetching protocols.

The second level is responsible for decoding the image from the input stream. CellVisualizer currently has two image readers. The JAI reader uses the Java Advanced Imaging library which supports BMP, GIF, JPEG, PNG, PNM, and TIFF files. In addition, we wrote a custom DIB reader for the Cellomics proprietary file format.

The third level of image handling in the CellVisualizer system is responsible for managing the images that have already been retrieved and decoded. Images typically take some time to retrieve, decode, and load into memory, from about half a second if the image is on the local hard drive to about five seconds per image if the image needs to be fetched via a SSH2 connection. Therefore, once the image is loaded into memory, we cache for future use. This level also supports basic image manipulation capabilities including turning on and off the red, green, and blue channels and displaying smaller portions of images showing individual cells. The Java Advanced Imaging library is used to give users additional useful imaging functions such as the ability to control the brightness and the contrast levels of an image.

4.4.2 Linking To Web Databases

CellVisualizer launches webpages by using Ostermiller Utils, an open source library for common programming tasks including opening a browser from an application on Windows, Unix, and Macintosh. Gene information for a particular point is usually stored in the database in one of two ways. Gene information may be stored directly

in each of the “Per Image” tables or there may be a separate “Gene Info” table to which “Per Image” and “Per Object” tables can link. The online gene databases are then queried to retrieve available information on the gene of interest.

4.4.3 Retrieving All Measurements

We also make it easy for users to query every measurement for any particular point. Users can select using a mouse or type in genes of interest and open a table which displays all the values in the database for that point.

4.4.4 Plotting Per Object Plots

Another useful function that CellVisualizer provides is the ability to easily plot “Per Object” data from selected points in “Per Image” plots. For example, we may notice that the average cell size may be large for a particular gene and that we are interested in whether the distribution of the cell size in this gene is different from all the other genes. We can easily select points of interest in the “Per Image” plot and ask for a “Per Object” subplot to display the data that correspond to the selected gene in the “Per Image” plot.

4.5 System Architecture Overview

The CellVisualizer system must retrieve data from a database, display data in multiple plots, retrieve images from a file server, and even open an external web browser. Achieving all these goals in a robust way requires the system to be extensible and modular. Figure 4-5 shows a high-level diagram of how different parts of the system interact with each other. The following sections describe each component in detail.

4.6 Database Layer

CellVisualizer accesses the data from a database through the database layer. Although the JDBC (Java Database Connectivity) API provides cross-Database Man-

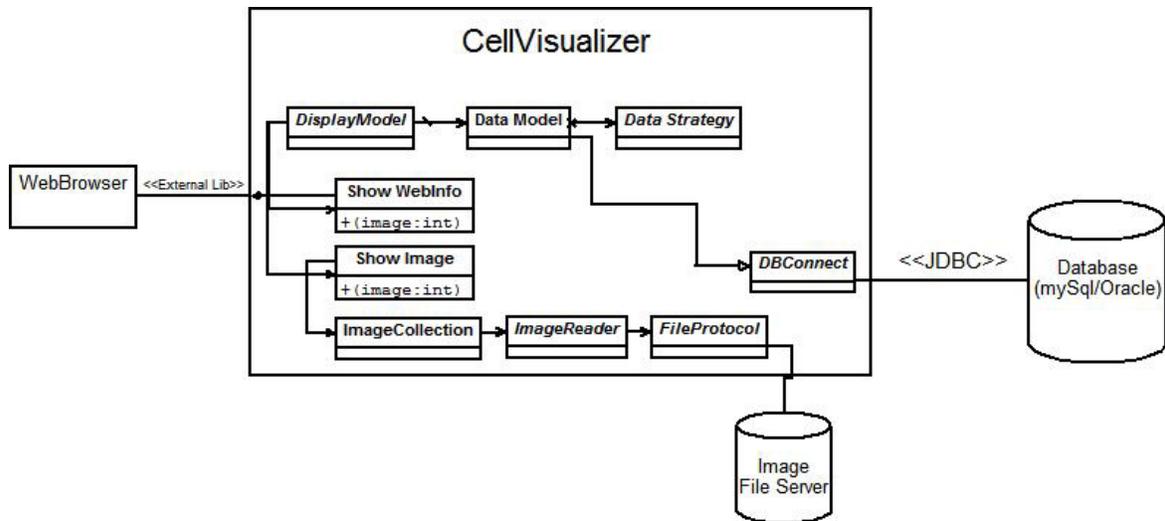


Figure 4-5: High-level diagram of CellVisualizer. The database stores measurements while the image file server stores the original images taken by the microscope.

agement System (DBMS) connectivity to a wide range of SQL databases and access to other tabular data sources, such as spreadsheets or flat files, it fails to provide an important capability - creating temporary tables. We implemented our own database layer on top of the JDBC layer to address this deficiency. This layer hides the actual connection to the JDBC database from the rest of the system to ensure that it is easy to extend the database support beyond the currently supported mySQL and Oracle databases.

Properties Files

CellVisualizer needs to know various properties about the database in order for the system to be fully functional. For example, it needs to know what column is the unique identifier in the “Per Image” tables, which columns correspond to the path/file names that specify the location of the red/green/blue images, and which column specifies the gene name. Since these column names may be different depending on the type of experiment and preference of the users, instead of hard-coding these variables into the system or forcing users to enter these values everytime they ran the system, the user creates “properties” files which specify the experimental database and are loaded

into CellVisualizer.

4.7 Internal Data Structures

The CellVisualizer system uses the model-view-controller pattern, originally articulated in the Smalltalk-80 user interface[11], to separate the user-interface/visualization front-end from the other parts of the application. This design decision allows the system to be easily extended. We would only need to create a new view-controller object if a new visualization display is needed. Similarly, we would only need to create a new model object if we wanted to support a vastly different data source.

4.7.1 The Data Model

The “data model” serves as the model for the model-view-controller abstraction and contains information about tables, columns, ranges, and other criteria that can be used to construct a SQL statement to query data from the database. All information requested by the user, whether it is data to be displayed in plots or gene identifiers used to link to an external web browser, is retrieved using a “data model” object.

The attributes of the “data model” include a `string` that specifies the type of data (`Per_Image`, `Per_Object`, `Aggregated Per_Object`), a list of `TableColumn` objects (to be described further below), a list of all tables referenced (such as in columns and `where` clauses), a list of column aggregators (e.g. `avg`, `max`, `min`, `count`), a list of `where` clauses, and parameters that specify whether to order the results and whether to limit the number of results. Each “data model” also contains a “data strategy” object, which is discussed in the next section.

Table 4.1 shows an example data model that may be used to query the dataset for a scatterplot. The data returned would be the slide02 and slide03 nuclei area measurements where the corresponding gene knockout in slide07 produced a large nuclei area (greater than 20). This “data model” produces the following SQL statement:

```
SELECT (S102_Per_Image.Mean_Nuclei_Area / S103_Per_Image.Mean_Nuclei_Area),  
       (S102_Per_Image.Mean_Cell_Area / S103_Per_Image.Mean_Cell_Area)
```

Table 4.1: Example Data Model Object

Attribute	Example Values
Type of Data	Per_Image data
Data Strategy	Per_Image strategy
TableColumn	[S102_Per_Image.Mean_Nuclei_Area/S103_Per_Image.Mean_Nuclei_Area] [S102_Per_Image.Mean_Cell_Area/S103_Per_Image.Mean_Cell_Area]
Tables	S102_Per_Image, S103_Per_Image, S107_Per_Image
Where Clauses	S107_Per_Image.Mean_Nuclei_Area > 20, S102_Per_Image.imagenumber = S103_Per_Image.imagenumber S102_Per_Image.imagenumber = S107_Per_Image.imagenumber
Order By	
Limit	

```
FROM S102_Per_Image, Slide03_Per_Image, S107_Per_Image
WHERE S107_Per_Image.Nuclei_Area > 20 AND
      S102_Per_Image.imagenumber = S103_Per_Image.imagenumber AND
      S102_Per_Image.imagenumber = S107_Per_Image.imagenumber
```

The TableColumn Object

As indicated in the previous section, CellVisualizer not only has the ability to plot simple “Per_Image” plots from columns in the database, but also allows the user to type mathematical expressions (e.g.

`S102_Per_Image.Mean_Cell_Area / S103_Per_Image.Mean_Cell_Area`) as columns to query a plot. As shown in Figure 4-6, the interface allows the user to select certain tables as groups, and then to select columns from these tables. When the user selects a grouped table, the system averages the values in the selected column across the tables in the group.

Once the user types or selects the column expression to query, `TableColumn` object is created and stored in the data model. The `TableColumn` object has two attributes: the column expression stored as a string and a list of tables that the expression references. In order to parse any arbitrary mathematical expression a user may input, we wrote a parser that identifies the variables in the expression, which in our case should be in `table.column` format. We then add the table, associated with each such

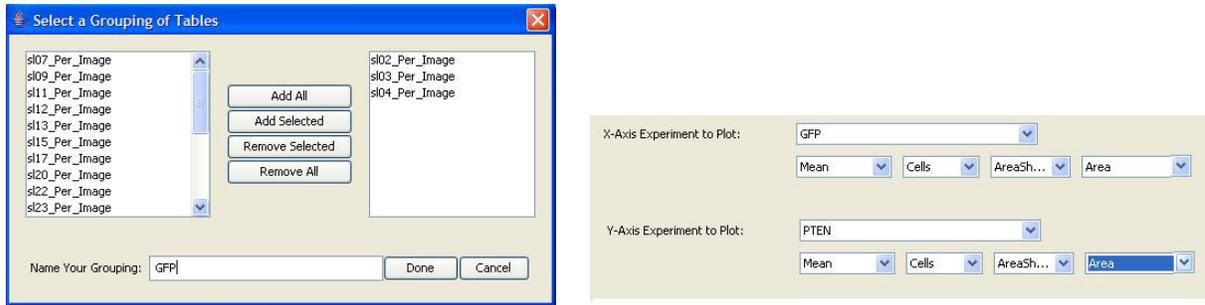


Figure 4-6: For convenience, the CellVisualizer interface allows users to label groups of tables and then select columns from these tables to plot. CellVisualizer assumes the user is requesting the averages of the values in the columns. This feature allows the users to label replicates together and compare different groups of replicates.

variable, to the list of tables.

Ensuring Table Links

Every time the data model is changed, either by adding/removing `TableColumn` objects or adding/removing statements to the `where` clause, the data model is responsible for ensuring the integrity of all the necessary table links. Specifically, the data model must make sure that all the tables that are referenced in the SQL statement are linked to each other as necessary.

Currently, the system handles this by assuming that there is a common column name that all “Per Image” and “Per Object” tables are linked by - e.g. “ImageNumber”. Notice that it doesn’t make any sense for 2 different “Per Object” tables to be linked to each other because the objects measured in each image of a “Per Object” table have no underlying association with the objects in another “Per Object” table. Moreover, the user can also enter an additional table and specify how this table is linked to the data tables. By parsing every expression that is added to the data model, the data model knows from which tables it requests data from, and adds the corresponding links.

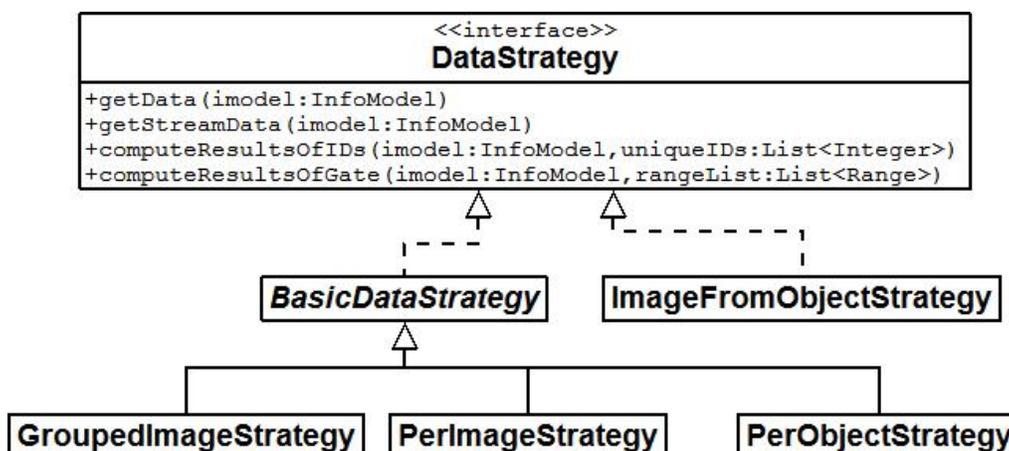


Figure 4-7: UML diagram of the data strategies currently implemented in the CellVisualizer system. Not all methods of the “Data Strategy” interface have been shown.

4.7.2 Data Strategy

In addition to all the data model attributes mentioned above, each data model has a “Data Strategy” object which specifies how the data model should retrieve the data from the database. We designed “Data Strategy” objects because we are interested in presenting the data in multiple ways. More specifically, we may want to plot the “Per Image” or the “Per Object” measurements of one particular experiment or we may be interested in plotting the averages, standard deviation, or other aggregations of the “Per Object” measurements. Other plots of interest would be measurements that group several “Per Image” tables together; for example, in the Fly203 database mentioned in Chapter 2, it would be useful to group and plot all the GFP knockout measurements versus all the PTEN measurements.

The “Data Strategy” interface defines methods for plotting the data and for selecting/brushing/highlighting subsets of the data. We created a class that implements the “Data Strategy” interface for each distinct way to plot data; the class structure is shown in Figure 4-7. If the system needs to be extended because there is a need to retrieve the data from the database in a different way, one would only have to implement the “Data Strategy” interface to ensure that all other components of CellVisualizer works as expected.

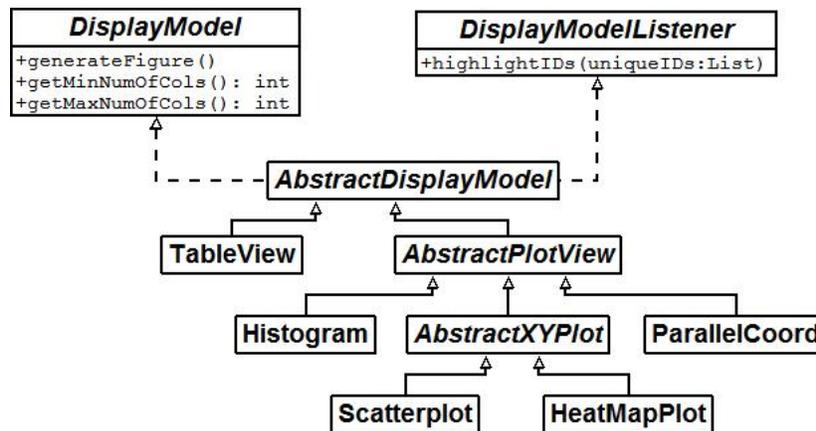


Figure 4-8: UML diagram showing the class hierarchy of display model objects.

4.7.3 Plots

CellVisualizer currently has four different types of plots: histograms, scatterplots, 2-dimensional histograms, and parallel coordinates. In addition to these plots, users can also choose to view the data in a table. The system is designed to allow programmers to add new types of plots. Figure 4-8 shows the current class hierarchy diagram of plots. Adding a new plot simply requires the programmer to implement the DisplayModel and DisplayModelListener interfaces. Plots use “data models” to get the data from the database. Users specify the “data model” for a particular plot through a graphical user interface and further alter or create new “data models” by interacting with open plots.

While we have used the word “plot” here to describe the entire figure that is drawn, including the labels and axes, it is worthwhile to note that the drawing of the actual charts (the bars in histograms and the points in scatterplots, for example) is encapsulated within each class. This design allows us to both separate CellVisualizer functionality from the charting functionality and to use different charting implementations if necessary.

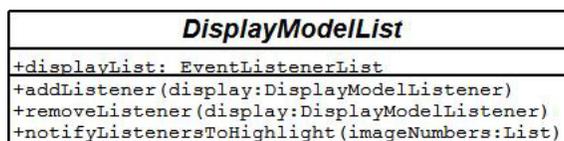


Figure 4-9: UML showing the attributes and methods of the DisplayModelList which keeps track of currently open plots and notifies plots of the points are highlighted.

4.7.4 Brushing

We use the Observer Pattern to implement brushing in CellVisualizer. As shown by the UML diagram in Figure 4-9, CellVisualizer stores a list of currently opened plots which is updated whenever the user opens or closes a plot. Whenever a user highlights points in one plot, all other plots are notified of the ImageNumbers that the user highlighted. The ImageNumbers are used to highlight the corresponding points in each plot.

4.8 Performance Considerations

A data visualization tool should facilitate user exploration and interaction with the data and should run in near real-time. However, since it is not possible to load tens of gigabytes of data into physical memory on a typical desktop, and it takes a long time to read 8 million rows from a database file, we implemented intermediate strategy for data measurement to optimize the interactivity of CellVisualizer when possible.

4.8.1 In-Memory Database

To balance the demands of user interactivity with the limitations of available memory, we integrated the hypersonic SQL database engine (HSQLDB), an open source relational database management system which supports in-memory data access. This allows the users to fetch the columns that they use frequently and store them in memory before starting to explore the data. Once the relevant data subset is stored in memory, interacting with the data becomes faster as reported in Table 4.2. The

Table 4.2: Time taken to retrieve data to display in a scatterplot

Database	Data Location	# of Rows	Time Taken
mySQL	Maxtor 6L250M0 (Serial ATA 7200 rpm)	4,095,698	1 min 5 sec
HSQLDB	In Memory	4,095,698	8 sec

number of columns that the user is able to download is limited by the amount of memory available.

Though it takes about the same amount of time to download several columns of data as it would to do a single plot, the benefit is that if the same columns were to be used in another plot, the ensuing plotting would be a lot faster. Moreover, since retrieval of data from the database on hard disk is mainly constrained by the amount of time the database takes to cycle through all the entries in the database, it is faster to download 4 columns into memory at once (one pass through all the records) and then plot 2 scatterplots than to simply plot 2 separate scatterplots which retrieves 2 sets of 2 columns from the database (2 passes through all the records).

We ensure that downloading only parts of the database doesn't prevent the user from accessing data that has not been downloaded into the in-memory database by employing the "Chain of Responsibility" design pattern. If the system attempts to execute a query that requires a column that has not been stored in the in-memory database, the system catches the error and forwards the request either the mySQL or Oracle database where all the data resides.

4.8.2 Memory Constraints on Plots

When designing the system, we had to consider the amount of functionality and interactivity needed in particular plots. Specifically, we wanted to give users a lot of tools when working with two-dimensional histograms. For example, we wanted users to be able to plot all the "Per Object" data in their database (typically millions of points), select a region of interest, display a montage of random cell objects in that gate, display a single image with the objects within that region marked, and compute the enrichment scores of each image. select a region of interest, display a montage of random cell objects in that gate, display a single image with the objects within that

region marked, and compute the enrichment scores of each image.

To balance our requirements of interactivity with our constraints of memory, we initially performed all calculations necessary on the database server, thus minimizing the amount of memory we used. For example, here would be the SQL code that computes the values of every bin in the two-dimensional histogram (Nuclei Area versus Cell Area, 0.27 is the minimum Nuclei Area, 13.42 is the maximum Nuclei Area, 0.91 is the minimum Cell Area, 13.75 is the maximum Cell Area, and 450 x 450 is the number of different bins we want)

```
select floor((Nuclei_Area-0.27)*450/(13.42-0.27))
as A, floor((Cells_Area-0.91)*450/(13.75-0.91))
as B, count(*) from Per_Object group by A,B;
```

We also wrote SQL queries to pick random cells from a region of interest. However, the database took too long to perform all these requests, especially given that before we could process the query to bin the data, we must compute the max/min values as well (thus requiring an additional pass through the database).

In the end, we determined that although users may only be able to open few “Per Object” 2-D histogram at a time, it is more important to allow users more interactive capabilities. Therefore, in addition to storing the two columns of “Per Object” measurements being plotted in 2-D histograms, we decided to also store the associated ImageNumber/ObjectNumber pair. Since every cellular object identifier is stored in physical memory, selecting random cells and computing other calculations became near instantaneous once the 2-D histogram is drawn.

4.9 Summary

We’ve designed a modular and extensible visualization system that currently allows users to view two-levels of data in multiple different ways. Users can draw histograms, scatterplots, 2-D color histograms, and parallel coordinates. Additional plots can easily be added if there is a need to view the data in a different way. CellVisualizer

makes it easy for users to compare data across similar experimental conditions by understanding that each data point in the dataset corresponds to a particular image, which corresponds to a particular gene knockdown. In addition, we link each data point back to the image it came from and to gene databases on the Internet to facilitate data exploration. The modularity of the system allows additional tools to easily be incorporated into CellVisualizer.

Chapter 5

Example Extension - Cell Classification

CellVisualizer was initially designed as a visualization tool, but we also intended the system to be easily extensible, especially to other areas of data analysis. In this chapter, we describe about how we extended CellVisualizer to incorporate an automatic classification system.

5.1 Cell Classifier

Cell Classifier was developed to discover all the cellular objects in the dataset that are characteristically similar to already seen interesting objects. This “Per Object” classification strategy takes an approach similar to content-based image retrieval in the field of computer vision. The user identifies a few cells showing a phenotype of interest, along with a few negative examples. These labeled cells are used to train a classifier. Once the classifier is trained, the user can choose to retrieve more cells, which the user can again label, which then produces a new, hopefully better, classifier. Once we have a classifier that we believe is sufficiently accurate, we can use that classifier to find every single positive cell in the entire dataset as well as compute enrichment scores for each image - i.e., find genes that produce significantly more cells of the interesting phenotype.

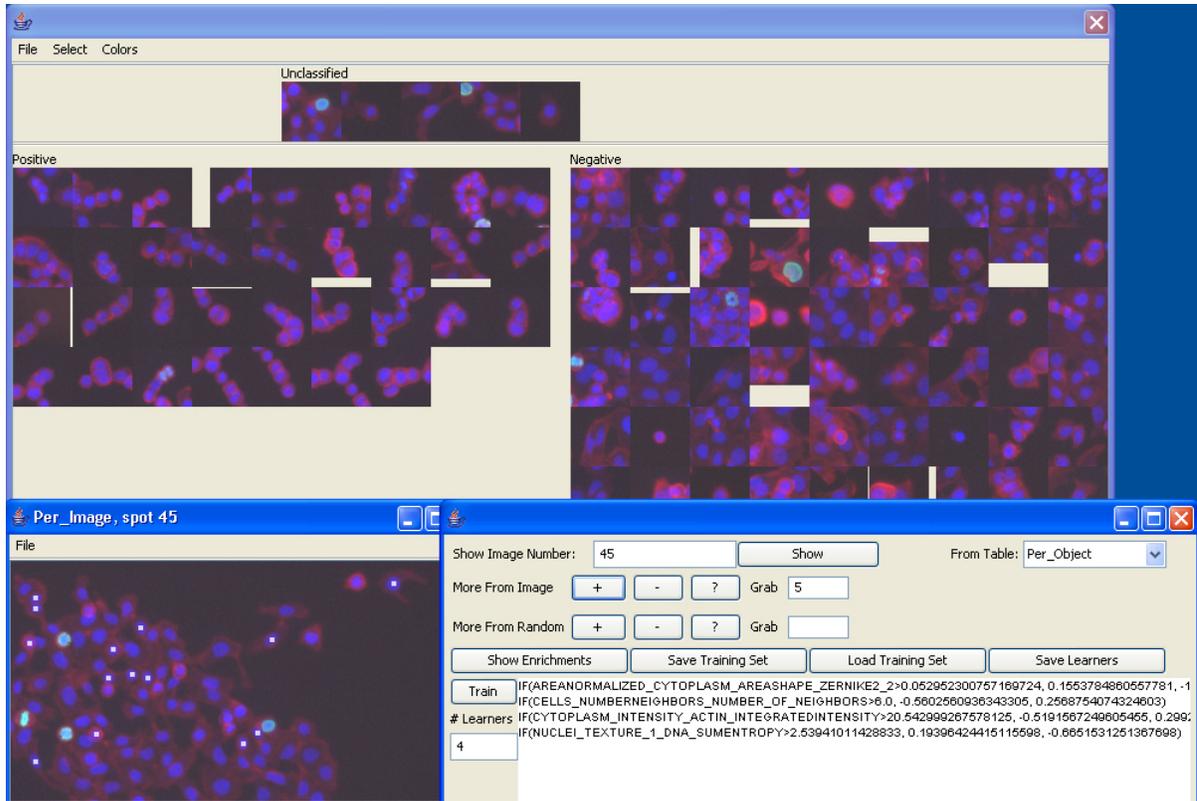


Figure 5-1: The Cell Classifier module in CellVisualizer. Users drag the individual cells from the unclassified board (top) to the positive (left) or negative (right) board and train on their examples to produce a classifier (bottom right). The results of the classifier on an entire image can be viewed (bottom left), where dots label all the positive cells.

The original cell classifier was written in Objective-C for the Macintosh. We have ported the classification tools to be usable from within CellVisualizer. Figure 5-1 shows the CellVisualizer user-interface of the classification tools. In this example, a classifier is trained to find cells which look as if they are grouped like peas in a pod.

5.1.1 Linking From Plots in CellVisualizer

In addition to simply porting classifier code from objective-C to Java, we wanted to also link the classification functionality to the CellVisualizer visualization tools. As mentioned in previous chapters, there may be several images or objects that we want to exclude from our dataset because these images or objects may be corrupted. By linking plots in CellVisualizer to the classification tools, we can exclude these

corrupted measurements from our calculations when classifying cells. CellVisualizer allows users to first plot their data, select regions of interest on these plots to include only images/cells of interest, and then use the classifier on the relevant images/cells.

Picking Random Cells

In order to further explain how CellVisualizer interacts with the classification tools, it is important to discuss two steps in the classification algorithm: selecting random cells to present to the user and determining how many cells in each image match the phenotype we are looking for.

In the case where we are considering the entire dataset or selecting entire images of interest, we take advantage of the fact that there is a column in the “Per Image” tables of the database that denotes how many objects are in each image. Since we know that the object numbers are labeled consecutively starting from 1, it is relatively straightforward to use a pseudorandom number generator to pick a random object to find the image/object number pair that uniquely identifies a cell. However, when we decide to exclude specific cellular objects, it is necessary to pick random cells without using the column that tells us how many objects are in each image. For this reason, we created a “CellPicker” interface for the classification tool to use.

Once we’ve picked random cells, the trainer picks measurements that best separates “positive” cells (cells that match the phenotype) and “negative” cells. We then categorize every single cell in the database to determine how many and what percentage of cells in each image are “positive” cells.

Per-Image Scatterplots

Currently, “Per Image” scatterplots can be used to include only relevant images in the classifier. This allows users to exclude images that may be corrupted by fibers or blobs of dye, as mentioned in section 1.1.1. In “Per Image” plots, we exclude some images from our calculations using a SQL query, both when retrieving random cells and when computing the number of “positive cells”.

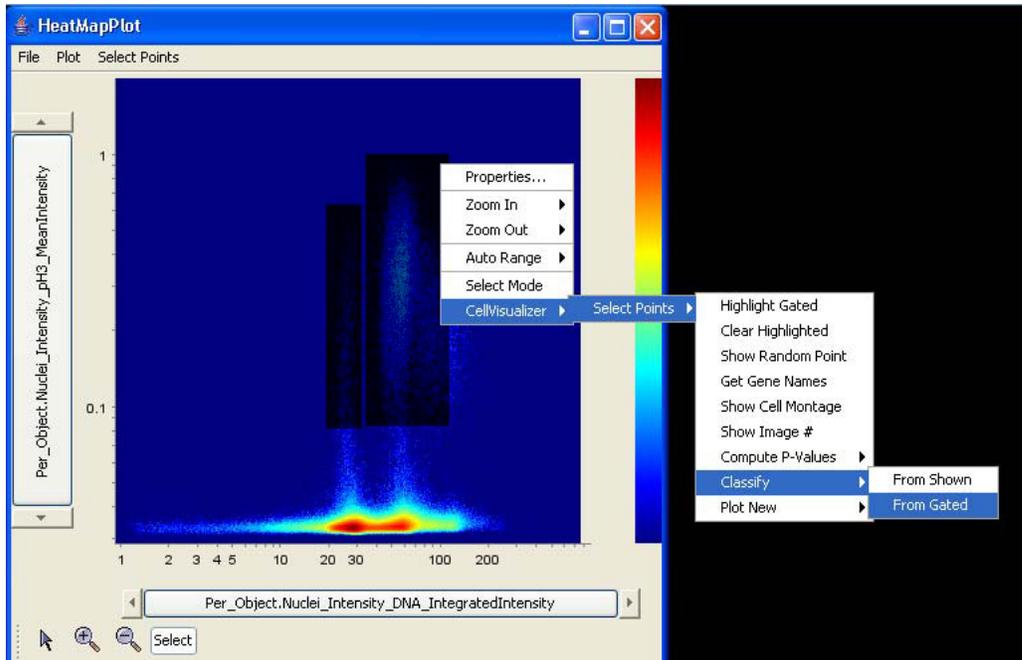


Figure 5-2: Selection of a region in a 2-D histograms to use the cell classification tool on. This opens the CellClassifier module as in Figure 5-1 but limits all the computation to cells within the selected gate.

Per-Object 2-D Histograms

To exclude individual objects from the classification process, users can select the objects of interest on 2-D histograms of their data. We implemented the selection process in 2-D histograms because we store all the ImageNumber/ObjectNumber pairs for each object that is plotted in the 2-D histogram. We numerically order all the unique cells in the selected region and randomly pick cells only from the ordered list. When determining which cells are “positive”, we retrieve every cell (ImageNumber/ObjectNumber pair) from the database and whether it matches the phenotype. We order all the cells we’ve retrieved in the same ordering as the cells in the selected region of interest and discard all cells not in the selected region before computing enrichment calculations. Figure 5-2 illustrates a 2-D histogram that plots the total nuclear DNA intensity (horizontal axis) against mean nuclear phospho-histone H3 intensity and how a user would use the two particular gated windows for classification.

We have discussed high-throughput microscopy data and the capabilities of the CellVisualizer system. Now we will discuss several examples of how CellVisualizer was actually used.

Chapter 6

Case Studies

The benefits provided by any application can best be illustrated by how the application is used in practice. In this chapter, we present several ways CellVisualizer is currently being used to the exploration of high-throughput data.

6.1 Finding Genes that Regulate Mitosis

In this section, we will describe how CellVisualizer is used to find genes that regulate mitosis. The dataset we explored was created by screening genes using *Drosophila* RNA interference living cell microarrays. There were 1120 spots of dsRNA, including three replicate spots for each of 288 genes (mostly kinases and phosphatases), plus 256 negative control spots lacking dsRNA [2]. DNA was stained with Hoechst, which marks the nuclei, F-actin was stained with phalloidin, which marks the cell, and phosphorylated histone H3 (phospho-H3) was stained using an antibody, which marks nuclei in mitosis.

6.1.1 Quality Control

As mentioned in Chapter 1, the imaging process is not perfect. Fibers and other artifacts sometimes interfere with measurements of some of the images. We have discovered that such images have abnormal image thresholding values - the value

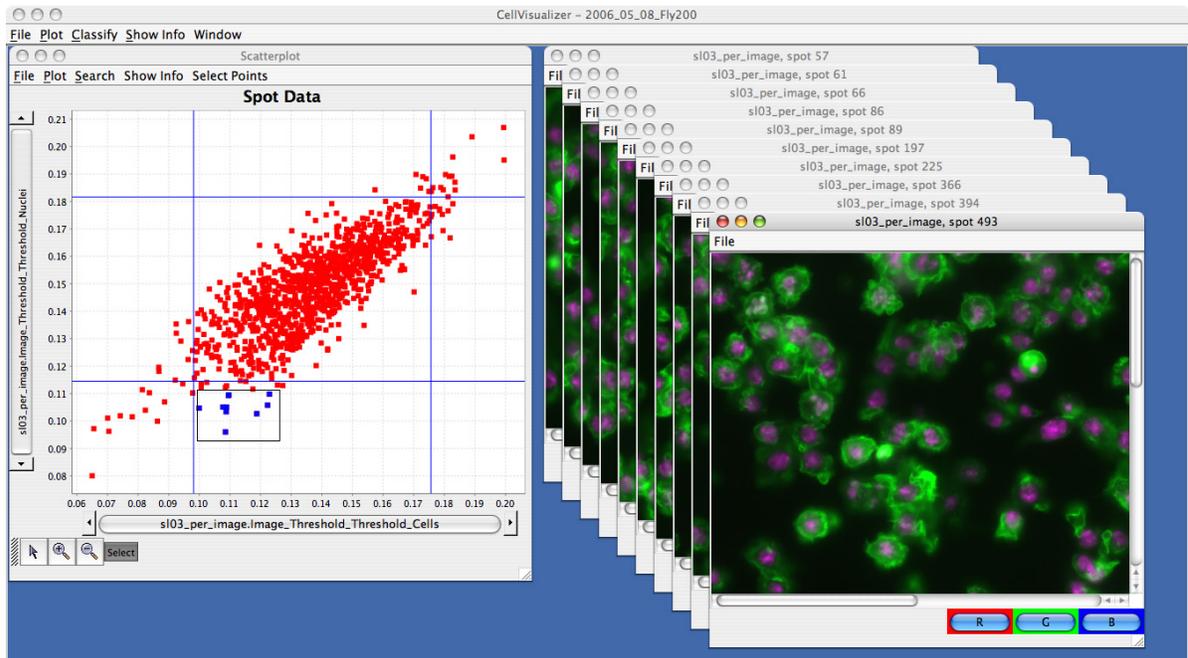


Figure 6-1: Step 1 checks to make sure the images are not corrupted by artifacts. The blue points in the scatterplot correspond to the images on the right.

that separates the background pixels from cell/nuclei pixels in an image. Thus, as show in figure 6-1, step 1 of the data exploration process is to quickly check whether such abnormal images exist by plotting `ImageThresholdCells` values against `ImageThresholdNuclei` values and displaying images with abnormally high and low threshold values.

In this dataset, all the images seem to be free of artifacts or other anomalies. If there were to be images with artifacts, we would then select all the data points without the artifacts and then plot a new scatterplot of just these normal points.

6.1.2 Plotting Measurements of Interest

The knockdown of genes that regulate the mitotic process would prevent the cells from dividing once the nucleus replicates. Such cells would have larger nuclei and cell size compared to normal cells. In step 2, we plot `MeanCellArea` vs. `MeanNucleiArea` for each image. We then use search by gene name to highlight all the control points - points where no gene has been knocked down - to again ensure that our experiment

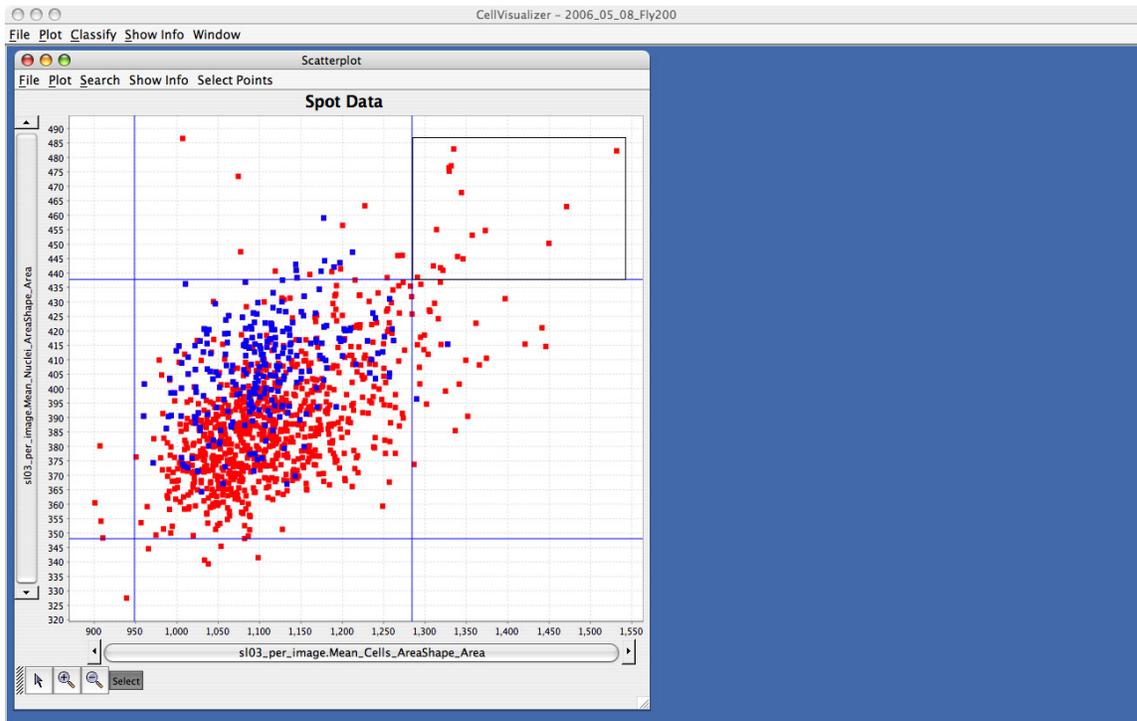


Figure 6-2: In step 2, we plot our measurements of interest and then highlight all the control points in blue. This is to make sure that the measurements for the controlled experiment is as we expect them to be.

ran as expected. Figure 6-2 shows that our control points are generally where we expect them to be.

6.1.3 Finding Interesting Genes

In the next step, we explore the dataset to find interesting genes which meet our criterion - in this case genes that regulate the mitotic cycle. We use our selection tool to drag a rectangle around all the images whose mean cell size and mean nuclei size are more than two standard deviations higher compared to all the other images (top right corner of the plot in Figure 6-2). Next, we highlight these genes and ask for the gene names of all these points. Since there are 3 replicates of every gene knockdown, we expect to find the same gene names retrieved 3 times; the knockdown of the gene should produce greater cell and nuclei size in all 3 replicates. As figure 6-3 shows, there was one such gene STG which produced such results. A few other genes had

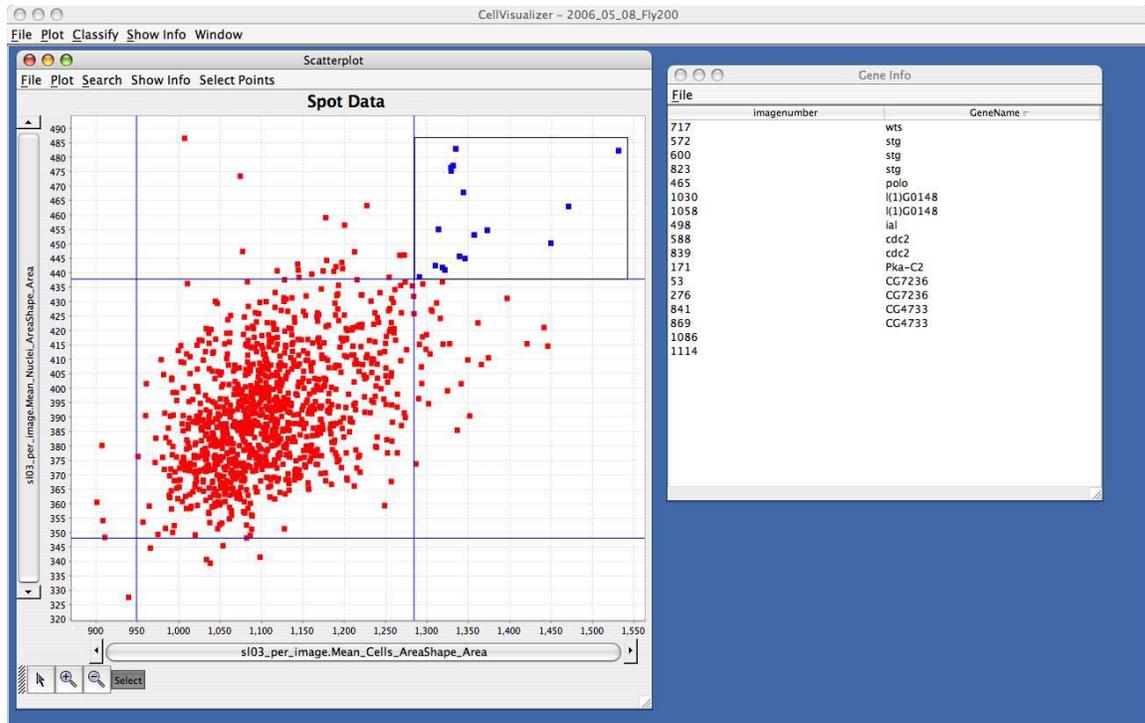


Figure 6-3: We request the gene names of all the points that have large mean cell and nuclei size. All 3 replicates of STG knockdown produce large mean cell and nuclei size.

2 replicates with abnormally large cell and nuclei area. To locate the third replicate for such genes, we use CellVisualizer and query the “gene by name”. For example, we see in Figure 6-4 that the third replicate of the CDC2 gene is located just within the two standard deviation cutoff.

6.1.4 Getting More Detailed Information

Now that we have several genes of interest, we can explore more detailed information about each of the genes. In this particular case study, we will concentrate on the STG gene because it was the one gene where all three replicates showed up on our list. We do this by using the option of searching for genes by name. Once all the STG genes are highlighted, we open the images to see if the cells/nuclei look as we would expect. We also use CellVisualizer to display web information so we can see if the current literature already indicates that STG regulates mitosis. Figure 6-5 illustrates CellVisualizer when we highlight STG and display the Flybase web information and

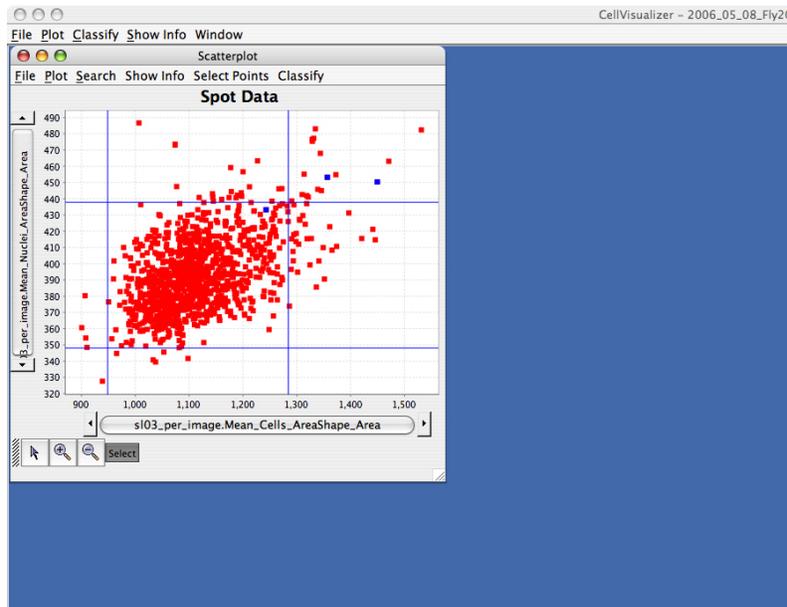


Figure 6-4: We search for gene CDC2 in the plot and highlight all 3 replicates in blue. We find that the third replicate is just within the two standard deviation lines (blue).

the underlying images.

When we look at the information available in Flybase, we see that STG has already been identified as a gene that regulates mitosis. Extensive evidence shows that differential expression of STG regulates mitosis during most stages of *Drosophila* development [12]. In this case, we haven't found anything novel but the result confirms that our system of sample preparation and data analysis is producing meaningful biological conclusions.

Per Object plots

In addition to looking at the average measurements in each of the STG images, we are also interested in comparing what the individual object measurements, specifically the total DNA intensity of the nucleus, look like in STG images versus control images. We thus select all the STG Per-Image data points and plot a histogram of the nuclei intensity of all the individual objects in all the STG points and compare it to a similar plot of all the objects in control points. Once we plot both histograms, we can select individual points (a single STG image) in the Per-Image plot to highlight

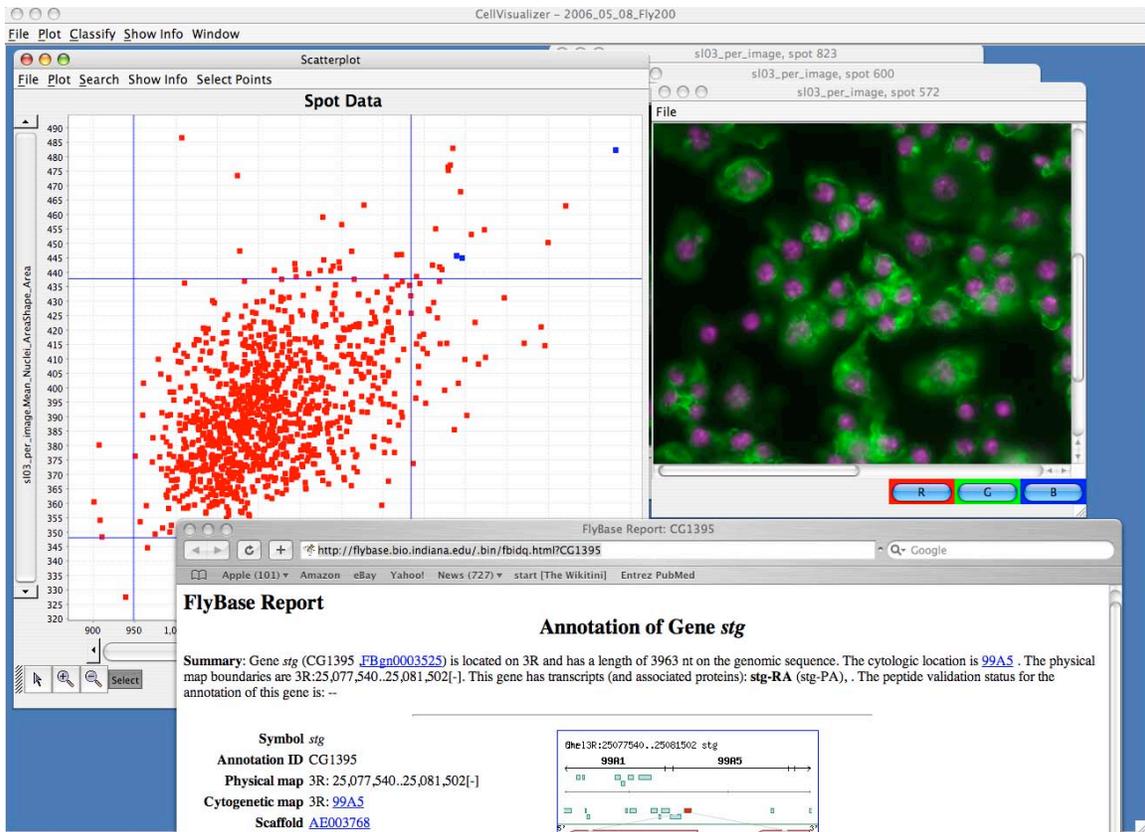


Figure 6-5: The underlying images and web information for the STG gene.

the Per-Object data in the aggregate histogram. This verifies that each individual image looks similar to the aggregate histogram.

What we hoped to see is greater DNA content (indicated by more total nuclei intensity) in the 4N, 8N, and 16N peaks in the STG images when compared to control images. This would mean the nuclei in the STG images kept duplicating without the cell actually splitting; i.e. the cells were stuck in mitosis. As illustrated in Figure 6-6, the 4N peak is greater, but it is inconclusive whether there are more cells in the 8N and 16N peaks. This type of exploration gives researchers enough information to hypothesize the mechanism of mitotic regulation by this gene and prioritize genes for further follow-up in the laboratory.

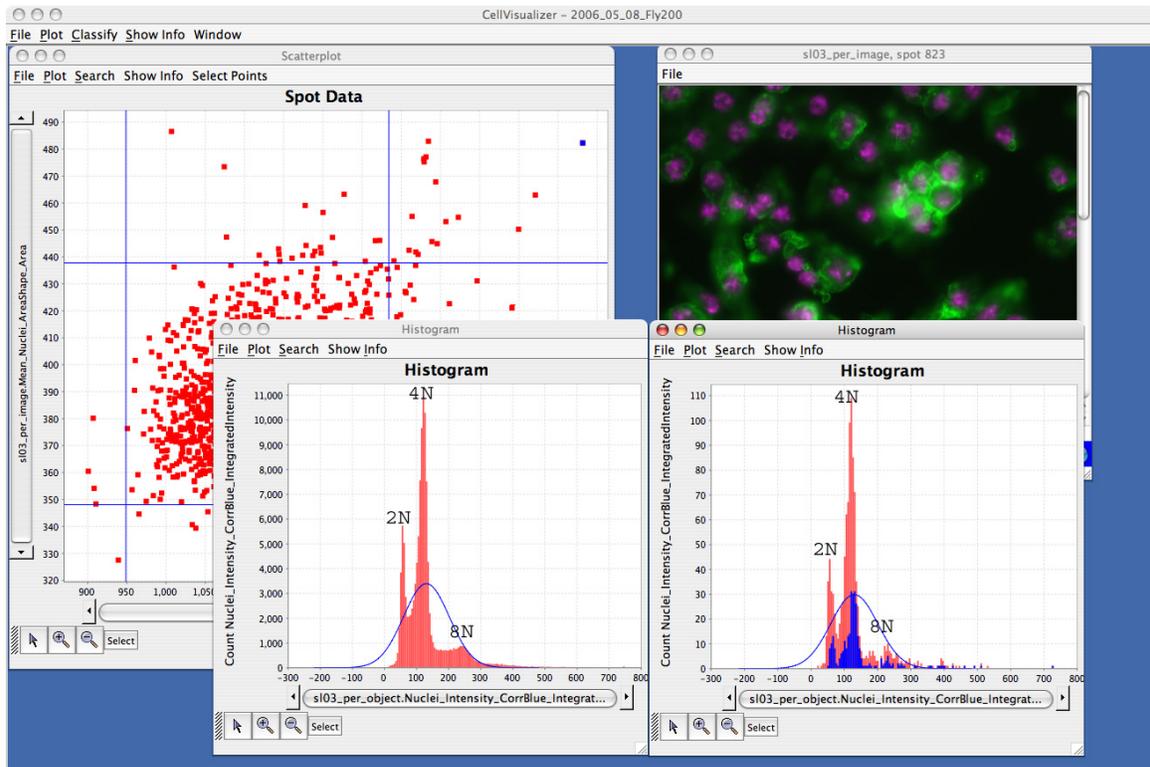


Figure 6-6: Histograms of the total nuclei intensity for each object in the control images (left) and in the STG images (right).

6.2 Comparing Two Different Knockdown Experiments

In this section, we explore *Drosophila* datasets similar to that in the previous case study. As in the previous study, we screened the genes using *Drosophila* RNA interference living cell microarrays. There were 1120 spots of dsRNA, including three replicate spots for each of 288 genes, plus 256 negative control spots lacking dsRNA. However, in this study, we are interested in how pairs of genes interact with each other. We produced three sets (to ensure the reliability of our data) of control slides where the only gene being knocked down in a particular spot was the one specified by the dsRNA. We also produced three sets of slides where in every spot the PSC2 gene was knocked down in addition to the gene specified by the dsRNA. We are interested in determining how the knock down of the PSC2 gene influences the effects of the other gene that is knocked down.



Figure 6-7: We group slides 2,3,4 together and name it “Control”.

6.2.1 Grouping Replicate Slides

We first group slides produced under the same experimental conditions together. As shown in Figure 6-7, slides 2, 3, and 4 are the “Control” slides and we name it as such. We also group the PSC2 slides, slides 11, 12, and 13, together.

6.2.2 Plotting Measurements of Interest

In this particular case, we are looking to find genes that decrease cell size under normal conditions but whose effects are mitigated when the PSC2 gene is also knockdown. After performing similar quality control steps to case 1, we draw a scatterplot with the average cell size of control slides on the X-axis and the average cell size of PSC2 slides on the y-axis. We select the region of the scatterplot where the average cell size is small in the control experiments but normal in the PSC2 experiments. As illustrated in Figure 6-8, when we query for the genes that correspond to these points, we find that one particular gene, CG02028, where our criteria is met in all three replicates. Our lab decided to conduct more extensive experiments to determine the interaction between gene CG02028 and PSC2 due to these findings as well as supporting results from coulter counter experiments and the available literature in the gene database.

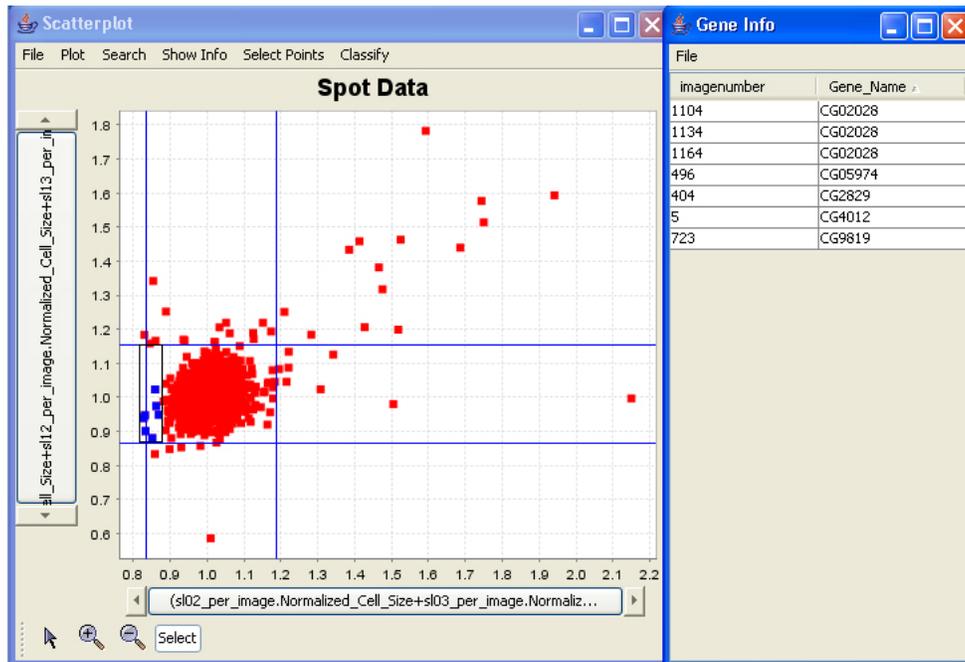


Figure 6-8: We plot the average cell size of images in control experiments (X-Axis) against the average cell size of images in PSC2 knockdown experiments (Y-Axis). We highlight points where cell size is decreased in the control experiments but restored to normal size in the PSC2 experiments. We find one gene, CG02028, where all three replicates produce this behavior.

Chapter 7

Discussion

In this thesis, we designed and implemented a system for visualization and exploration of large, multi-dimensional, hierarchical datasets. We tackled the challenges of linking and displaying hundreds of gigabytes of data in multiple ways from numerous tables both within and across several levels of a database. We evaluated the usefulness of the tool in real biological applications and sought to make the system as interactive as possible. In this chapter we will discuss possible extensions to the project as well as how changing technology may alleviate some concerns such as memory.

7.1 Other Applications

Although the CellVisualizer system is designed with CellProfiler datasets in mind, it can be used with other datasets as well. The types of datasets that CellVisualizer is particularly useful for would be datasets where there are two-levels of related hierarchical data. An example outside the high-throughput microscopy context would be exploring baseball players' statistics. The top level of data would be "Per Team" data and hold the summaries of statistics of each player on the team. We would be able to compare related statistics, such as the team's first baseman's batting average, across different teams. The second level of data would be detailed "Per Player" data and store information such as how a player performed in each game.

7.2 Future Work

In this work, we developed a system that allows users to visualize and explore two-levels of multi-dimensional hierarchical data. Along with optimizing the system to take advantage of further advances in technology, future work will extend the system to explore multiple levels of hierarchical data. For example, we may be interested not only in nuclei but also in all the nucleoli within each nucleus.

Another consideration for the future is how advancements in memory and storage technology improve the ability to store, retrieve, and view data. Currently, the visualization of data is limited by two main factors: the amount of RAM that the computer has, and how fast data is retrieved from a hard drive where the database is stored.

Most computers typically have less than 1 gigabyte of RAM (the maximum memory ceiling of 32-bit architecture machines is only 4 GB) and are thus unable to store the entire dataset in memory. However, as computers are upgraded to 64-bit processors and memory continues to become cheaper, it is conceivable that computers may have terabytes of memory in the future (the maximum memory ceiling of 64-bit architecture is 16 exabytes).

A consideration that goes hand-in-hand with memory usage is how fast data is retrieved from hard-disk storage. Currently, the fastest workstation and server hard drives spin at 15,000 rpm [16]. Since the time to seek and retrieve information from the database is limited by how fast the database can find this information in its hard drive, improvements in hard drive spindle spin would directly enhance the CellVisualizer interactivity.

7.3 Conclusion

We have described CellVisualizer, a system to visualize large, multi-dimensional, hierarchical datasets. In addition, we have linked the data to images and gene databases to enhance the data exploration process. We demonstrated the system on preliminary

biological experiments. CellVisualizer is currently in active use in our lab and will be made available to a broader community in the future. The ability to explore vast amounts of data produced in high-throughput experiments will significantly enhance the usefulness of this data in understanding biological phenomena at a cellular level.

Bibliography

- [1] Richard A. Becker and William S. Cleveland. Brushing scatterplots. *Technometrics*, 29(2):127–142, 1987.
- [2] Anne E. Carpenter, Thouis Ray Jones, Michael Lamprecht, Douglas B. Wheeler, Colin Clarke, InHan Kang, Ola Friman, David A. Guertin, Joo Han Chang, Robert Lindquist, Jason Moffat, Polina Golland, and David M. Sabatini. Cell-profiler: image analysis for high throughput microscopy. *Submitted*.
- [3] Cellomics. <http://www.cellomics.com>, 2006. [Online; accessed 03-August-2006].
- [4] Java CIFS. <http://jcifs.samba.org>. [Online; accessed 03-August-2006].
- [5] Spotfire DecisionSite. Handling large databases, 2004, version 7.3. Help Section.
- [6] G. Grumbling, V. Strelts, and The FlyBase Consortium. Flybase: anatomical data, images and queries. *Nucleic Acids Research* 34: D484-d488; doi:10.1093/nar/gkj068. <http://flybase.org/>. 2006.
- [7] Alfred Inselberg and Bernard Dimsdale. Parallel coordinates: A tool for visualizing multi-dimensional geometry. *IEEE Visualization*, pages 361–378, 1990.
- [8] JCraft. <http://www.jcraft.com/jsch>. [Online; accessed 03-August-2006].
- [9] Thouis Jones, Anne Carpenter, and Polina Golland. Voronoi-based segmentation of cells in image manifolds. *LNCS*, 3765, 2005.

- [10] Daniel A. Keirn and Hans-Peter Kriegel. Visdb: Database exploration using multidimensional visualization. *IEEE Computer Graphics and Applications*, pages 40–49, September 1994.
- [11] G. Krasner and S. Pope. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of Object Oriented Programming*, 1(3):26–49, 1988.
- [12] Dara A. Lehman, Briony Patterson, Laura A. Johnston, Tracy Balzer, Jessica S. Britton, Robert Saint, and Bruce A. Edger. Cis-regulatory elements of the mitotic regulator, string/cdc25. *Development*, 126(9):1793–1803, 1999.
- [13] Spotfire. <http://www.spotfire.com>, 2006. [Online; accessed 03-August-2006].
- [14] Deborah F. Swayne, Duncan Temple Lang, Andreas Buja, and Dianne Cook. Ggobi: Evolving from xgobi into an extensible framework for interactive data visualization. *Computational Statistics & Data Analysis*, 2003.
- [15] Douglas B Wheeler, Steve N Bailley, David A Guertin, Anne E Carpenter, Caitlin Higgins, and David M Sabatini. Rnai living-cell microscopy for loss-of-function screens in drosophila melanogaster cells. *Nature Methods*, 1(2):1+, 2004.
- [16] Wikipedia. Hard disk — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Hard_disk&oldid=60531470, 2006. [Online; accessed 26-June-2006].