COMPUTATIONAL INFORMATION DESIGN

Benjamin Jotham Fry

BFA Communication Design, minor in Computer Science
Carnegie Mellon University, May 1997
SM Media Arts and Sciences,
Massachusetts Institute of Technology, May 2000

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy at the
Massachusetts Institute of Technology

April 2004

Benjamin Fry
Program in Media Arts and Sciences
Author

John Maeda
Muriel Cooper Chair
Associate Professor of Media Arts and Sciences
MIT Media Laboratory
Thesis Supervisor

Andrew B. Lippman
Program in Media Arts and Sciences
Chair, Departmental Committee on Graduate Students

COMPUTATIONAL INFORMATION DESIGN

Abstract

The ability to collect, store, and manage data is increasing quickly, but our ability to understand it remains constant. In an attempt to gain better understanding of data, fields such as information visualization, data mining and graphic design are employed, each solving an isolated part of the specific problem, but failing in a broader sense: there are too many unsolved problems in the visualization of complex data. As a solution, this dissertation proposes that the individual fields be brought together as part of a singular process titled Computational Information Design.

This dissertation first examines the individual pedagogies of design, information, and computation with a focus on how they support one another as parts of a combined methodology for the exploration, analysis, and representation of complex data. Next, in order to make the process accessible to a wider audience, a tool is introduced to simplify the computational process for beginners, and can be used as a sketching platform by more advanced users. Finally, a series of examples show how the methodology and tool can be used to address a range of data problems, in particular, the human genome.

Thesis Supervisor: John Maeda
Associate Professor of Media Arts and Sciences
MIT Media Laboratory

Doctoral dissertation committee

John Maeda
Associate Professor of Media Arts and Sciences
MIT Media Laboratory
Thesis Advisor

David Altshuler MD, PhD
Assistant Professor of Genetics and of Medicine
Harvard Medical School
Massachusetts General Hospital
Director, Program in Medical and Population Genetics
Broad Institute
Thesis Reader

Christopher Pullman
Vice President for Design
WGBH Boston
Thesis Reader

Acknowledgements

It's difficult to explain how much I owe to these people.

Mom & Dad, for everything.

Mimi, Jamie, Leif, Erika, and Josh, for helping me maintain perspective.

Mentors, in reverse order of their appearance: John Maeda, John Lilly, Suguru Ishizaki, Mark Mentzer, Darcy Bowden, Mrs. Heinemann, and John Hillegonds.

Committee members, David Altshuler and Chris Pullman.

Fellow ACG members, who provided motivation and kept me to a higher standard: Peter Cho, Elise Co, Megan Galbraith, Simon Greenwold, Bill Keays, Golan Levin, Casey Reas, Nikita Pashenkov, Bernie Manor, Allen Rabinovich, Jared Schiffman, James Seo, and Tom White.

The PLW crew: James Dai, Noah Fields, Patrick Menard, Carlos Rocha, Marc Schwartz, Seung-Hun Jeon, Mariana Baca, Annie Ding, Quinn Mahoney, Ricarose Roque, Mimi Liu, and Jessica Rosenkrantz.

Architecture expatriats with temporary residence in ACG: Axel Kilian, Omar Kahn, Afsheen Rais-Roshani, and John Rothenberg.

Those who ensured the smooth operation of all things ACG: Connie van Rheenen, Elizabeth Marzloff, and Heather Childress; and all things academic: Linda Peterson and Pat Solakoff.

Pardis Sabeti, my bud.

Friends who helped keep me sane, especially through the past year: Scott Corliss, Ravi & Sonia Hampole, Eugene Kuo, Joshua Schuler, Amanda Talbott.

Still more students and professors at the Media Lab who made me happy to be here: Erik Blankinship, Bill Buterra, Mark Feldmeier, Jeana Frost, Saul Griffith, Hiroshi Ishii, jh, Joe Paradiso, Wendy Plesniak, Brygg Ullmer, Max van Kleek, Ben Recht, Parul Vora, and Paul Yarin.

Table of Contents

1 Introduction

Biology has rapidly become a data-rich science, where the amount of data collected can outpace the speed with which it can be analyzed and subsequently understood. Sequencing projects have made available billions of letters of genetic code, as analysis continues to add layers of annotation that describe known or predicted features along the linear sequence. Ten years ago, a few thousand letters, representing a few hundred genes, were known from a small number of 'model' organisms, where today this has become billions of letters representing tens of thousands of genes across a rapidly growing number of organisms.

The quantity of such data makes it extremely difficult to gain a "big picture" understanding of its meaning. The problem is further compounded by the continually changing nature of the data, the result of new information being added, or older information being continuously refined. The amount of data necessitates new software-based tools, and its complexity requires extra consideration be taken in its visual representation in order to highlight features in order of their importance, reveal patterns in the data, and simultaneously show features of the data that exist across multiple dimensions.

One significant difficulty with such problems is knowing, given a set of data, how to glean meaningful information from it. To most, the process is entirely opaque. Fields such as statistics, data mining, graphic design, and information visualization each offer components of the solution, but practitioners of each are often unaware of, or unskilled in, the methods of the adjacent fields required for a solution.

Visual design—the field of mapping data to visual form—aids understanding, but typically does not address how to handle extremely large amounts of data. Data mining techniques can handle large amounts of data, but are disconnected from the means to interact with them. Software-based information visualization adds building blocks for interacting with and representing various kinds of abstract data, but typically the aesthetic principles of visual design are treated as less important or even superficial, rather than embracing their strength as a necessary aid to effective communication. For someone approaching a data representation problem (such as a scientist trying to visualize the results of a study involving a few thousand pieces of genetic data), they will often find it difficult to know where to begin (what tools to use or books to read are frequent questions). Similarly, it is difficult for the same person to critically evaluate the representation used, lacking the necessary background.

In order to properly address the issue of complex data visualization, several fields need to be reconciled as parts of a single process. By combining the necessary skills into a single, clearly documented field, they are made more accessible to those with some partial set of them—graphic designers can learn the computer science necessary for visualization, or statisticians can communicate their data more effectively by understanding the visual design principles behind data representation. The methods themselves are not new, but their isolation to individual fields has prevented them from being used as a whole, since it is rare for someone to obtain the requisite background in each.

The pages that follow outline a process titled *Computational Information Design* that seeks to bridge the individual disciplines, placing the focus instead on the data and how it is to be considered—rather than from the viewpoint and tools of each individual field.

1.1 DATA & UNDERSTANDING

This thesis is about the path from data to understanding. The data under consideration might be numbers or lists or relationships between multiple entities. The primary focus is *information visualization*, where the data is primarily numeric or symbolic rather than physical (i.e. genetic sequence data, where an abstraction of A, C, G, T letters are used to describe a physical structure, and are dominated by layers of annotation that accompany it), as opposed to another category of *visualization*, which concerns representation of primarily the physical nature of its subject (i.e. the physical shape of a molecule, where the significance is placed on its shape, rather than numeric attributes that describe it). There is overlap between the two categories, but they're used to describe the primary focus of the diagram (physical versus numeric features). These definitions are discussed further in the third chapter.

As a matter of scope, this thesis considers visual methods for the representation of data (as opposed to other methods, such as sound, see discussion in chapter eight). Because of its ability to process enormous amounts of data, the human visual system lends itself as an exceptional tool to aid in the understanding of complex subjects.

1.2 PROCESS

The process of understanding data begins with a set of numbers and a goal of answering a question about the data. The steps along this path can be described as follows:

1. *acquire* – the matter of obtaining the data, whether from a file on a disk or from a source over a network.

2. *parse* – providing some structure around what the data means, ordering it into categories.

3. *filter* – removing all but the data of interest.

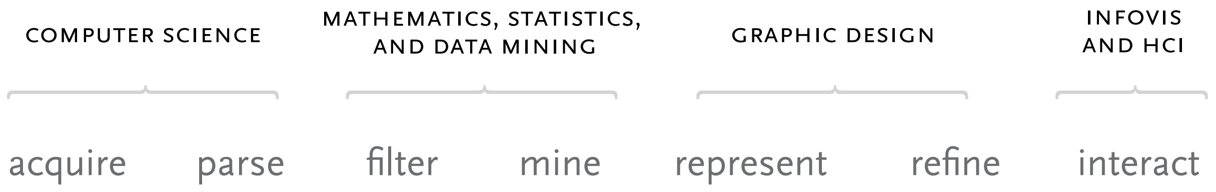
4. *mine* – the application of methods from statistics or data mining, as a way to discern patterns or place the data in mathematical context.

5. *represent* – determination of a simple representation, whether the data takes one of many shapes such as a bar graph, list, or tree.

6. *refine* – improvements to the basic representation to make it clearer and more visually engaging.

7. *interact* – the addition of methods for manipulating the data or controlling what features are visible.

Part of the problem with the individual approaches of dealing with data is that the separation of the fields leads to each person solving an isolated part of the problem, and along the path towards a solution, something is lost at each transition—a "telephone game" for context, where



each step of the process diminishes aspects of the initial question under consideration. The initial format of the data (how it is acquired and parsed) will often drive how it is structured to be considered for filtering and statistics or data mining. The statistical method used to glean useful information from the data might drive how the data is initially presented—the representation is of the results of the statistical method, rather than a response to the initial question.

A graphic designer brought in at the next stage will most often respond to specific problems with its representation as provided by the previous steps, rather than focusing on the initial question itself. Implementation of the visualization step might add a compelling and interactive means to look at the data filtered from the earlier steps, but the result is an in-depth, interactive look at a data set, using a particular statistical model, not a clear answer to the original question.

Further, for practitioners of each of the fields that commonly deal with data problems, it's often unclear the necessary methods to make it through the wider set of steps necessary to arrive at a solution to the problem in question.

This thesis describes the background of each of the individual fields in chapter three, while chapter five describes this process in depth, also citing the unique aspects of their combination.

1.2 TOOLS

An integrated approach usually implies that a single person is meant to handle the entire process. For larger projects, it might be possible to have one such person oversee the process as acted out by others, but still requires that person be thoroughly versed in the individual issues in order to maintain the same kind broad overview of the process.

The focus of this thesis is on the single practitioner, and the way that single practitioners are enabled is by better tools. Where video production was once exclusively the domain of large teams of people with expensive equipment, it's now possible to edit movies on a desktop workstation through tools that better enable individual users. This hasn't lead to the larger teams disappearing, but highlights the power of individual practitioners enabled by the right tools.

As a necessary supplementary component, a tool called *Processing* has been developed in conjunction with this research. It is a Java-based software development environment that aims to simplify the construction of graphically-oriented software. This goal helps beginners with programming concepts by placing the more interesting parts at the surface, while also providing a sketching environment for advanced users.

The majority of the projects described in this dissertation were developed using *Processing*, and provide examples of its capabilities.

Processing was conceived of and implemented by Casey Reas and this author, and is supported by an international community of collaborators. The project is discussed further in chapter six.

1.3 DOMAIN

While every problem is unique, the principles remain the same across data sets and across domains. This dissertation has a particular focus on the application of these methods across data problems from genetics. A domain was chosen so that specific examples could be addressed as a real test for the process introduced in this dissertation.

The difficulty in selecting a problem domain is that the reader must then come to understand parts of that domain in order to evaluate the strength of the solution. For this reason, the initial introduction of the process in chapter two uses a far simpler data set.

In spite of a focus on genetics, the issues are identical to those that must be considered by practitioners in other data-oriented fields. Chapter four describes a specific example concerning genetic variation data, and additional genetics projects are outlined in chapter seven, which catalogs information design experiments across several additional problem domains.

2 Basic Example

This section describes application of the Computational Information Design process to the understanding of a simple data set—the zip code numbering system used by the United States Postal Service. The application demonstrated here is purposefully not an advanced one, and may even seem foolish, but it provides a skeleton for how the process works.

2.1 QUESTIONS & NARRATIVE

All data problems begin with a question. The answer to the question is a kind of narrative, a piece that describes a clear answer to the question without extraneous details. Even in the case of less directed questions, the goal is a clear discussion of what was discovered in the data set in a way that highlights key findings. A stern focus on the original intent of the question helps the designer to eliminate extraneous details by providing a metric for what is and is not necessary.

2.2 BACKGROUND

The project described here began out of an interest in how zip codes relate to geographic area. Living in Boston, I knew that numbers starting with a zero were on the East Coast. Having lived in San Francisco, I knew the West Coast were all nines. Growing up in Michigan, all our codes were 4-prefixed. In addition, what sort of area does the second digit specify? Or the third?

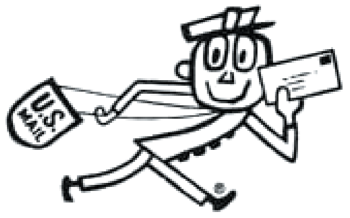
The finished application, called *zipdecode*, was initially constructed in a matter of a few hours as a way to quickly take what might be considered a boring data set (45,000 entries in a long list of zip codes, towns, and their latitudes & longitudes) and turn it into something that explained how the codes related to their geography and, as it turned out, was engaging for its users.

2.3 PROCESS

The Computational Information Design process, as it relates to the data set and question under examination here.

2.3.1 *Acquire*

The acquisition step refers to obtaining the data, whether over the network, or from a file on a disk. Like many of the other steps, this can often be extremely complicated (i.e. trying to glean useful data out of a large system) or very simple (simply reading a readily available text file).

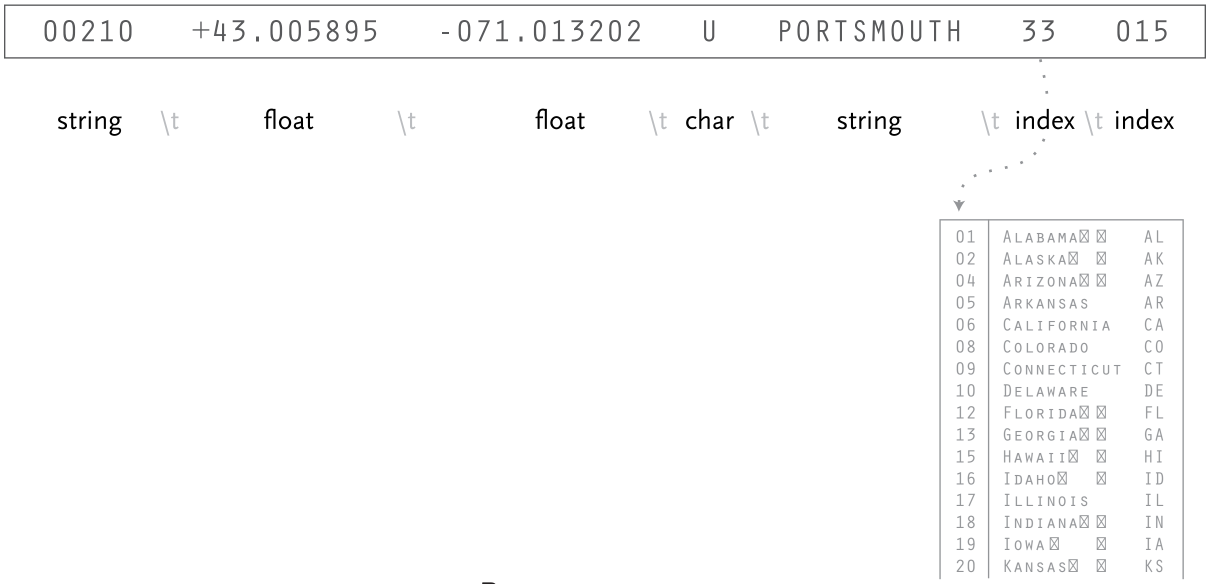


The acronym ZIP stands for Zoning Improvement Plan, and refers to a 1963 initiative to simplify the delivery of mail in the United States. Faced with an ever-increasing amount of mail to be processed, the zip system intended to simplify the process through a more accurate specification of geographic area where the mail was to be delivered. A more lengthy background can be found on the U.S. Postal Service's web site.

www.usps.com/history/

Today, the zip code database is primarily available via the U.S. Census Bureau, as they use it heavily as a method for geographic coding of information. The listing is a freely available file with approximately 45,000 lines, one for each of the codes:

```
00210   +43.005895   -071.013202   U   PORTSMOUTH    33   015
00211   +43.005895   -071.013202   U   PORTSMOUTH    33   015
00212   +43.005895   -071.013202   U   PORTSMOUTH    33   015
00213   +43.005895   -071.013202   U   PORTSMOUTH    33   015
00214   +43.005895   -071.013202   U   PORTSMOUTH    33   015
00215   +43.005895   -071.013202   U   PORTSMOUTH    33   015
00501   +40.922326   -072.637078   U   HOLTSVILLE    36   103
00544   +40.922326   -072.637078   U   HOLTSVILLE    36   103
00601   +18.165273   -066.722583       ADJUNTAS      72   001
00602   +18.393103   -067.180953       AGUADA        72   003
00603   +18.455913   -067.145780       AGUADILLA     72   005
00604   +18.493520   -067.135883       AGUADILLA     72   005
00605   +18.465162   -067.141486   P   AGUADILLA     72   005
00606   +18.172947   -066.944111       MARICAO       72   093
00610   +18.288685   -067.139696       ANASCO        72   011
00611   +18.279531   -066.802170   P   ANGELES       72   141
00612   +18.450674   -066.698262       ARECIBO       72   013
00613   +18.458093   -066.732732   P   ARECIBO       72   013
00614   +18.429675   -066.674506   P   ARECIBO       72   013
00616   +18.444792   -066.640678       BAJADERO      72   013
```


00210 +43.005895 -071.013202 U PORTSMOUTH 33 015
string \t float \t float \t char \t string \t index \t index

01	Alabama	AL
02	Alaska	AK
04	Arizona	AZ
05	Arkansas	AR
06	California	CA
08	Colorado	CO
09	Connecticut	CT
10	Delaware	DE
12	Florida	FL
13	Georgia	GA
15	Hawaii	HI
16	Idaho	ID
17	Illinois	IL
18	Indiana	IN
19	Iowa	IA
20	Kansas	KS

2.3.2 Parse

Having acquired the data, it next needs to be parsed—changed into a format that tags the meaning of each part of the data with how it is to be used. For each line of the file, it must be broken along its individual parts, in this case the line of text is separated by tabs. Next, each piece of data is converted to its useful format:

STRING – a set of characters that forms a word or a sentence (used for city/town names), or because the zip codes themselves are not so much numbers as a series of digits (if they were numbers, then the code 02139 would be the same as 2139, which is not the case) they are also considered a string.

FLOAT – a number with decimal points (used for the latitudes and longitudes of each location). The name is short for "floating point," from programming nomenclature of how the numbers are stored in the computer's memory.

CHAR – a single character, in this data set sometimes used as a marker for the designation of special post offices.

INTEGER – any generic number

INDEX – data (commonly it might be an integer or string) that points to another table of data (in this case, mapping numbered "FIPS" codes to the names and two digit abbreviations of states)

Having completed this step, the data is successfully tagged and more useful to a program that will manipulate or represent it in some way. This is common in the use of databases, where a such a code is used as a lookup into another table, sometimes as a way to compact the data further (i.e. a two digit code is better than listing the full name of the state or territory).

2.3.3 Filter

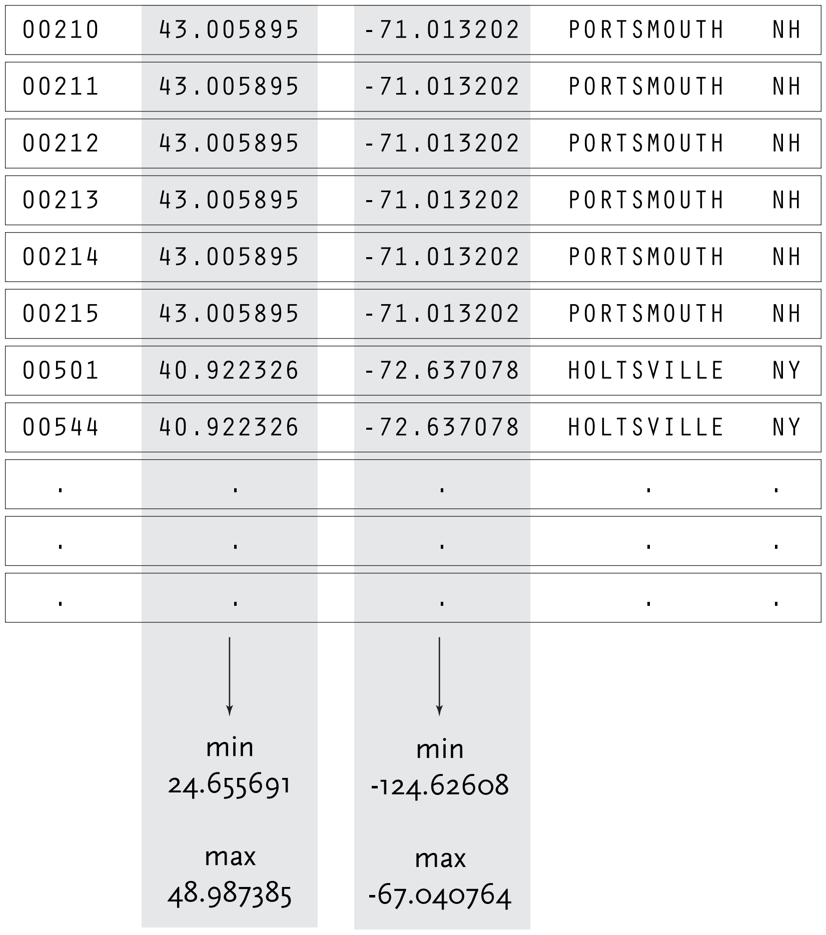
The next step involves the filtering of data, in this case the records not part of the contiguous 48 states will be removed. This mean Alaska and Hawaii will be removed (as this is only a simple sketch) along with other territories such as Puerto Rico.

00210	43.005895	-71.013202	PORTSMOUTH	NH
00211	43.005895	-71.013202	PORTSMOUTH	NH
00212	43.005895	-71.013202	PORTSMOUTH	NH
00213	43.005895	-71.013202	PORTSMOUTH	NH
00214	43.005895	-71.013202	PORTSMOUTH	NH
00215	43.005895	-71.013202	PORTSMOUTH	NH
00501	40.922326	-72.637078	HOLTSVILLE	NY
00544	40.922326	-72.637078	HOLTSVILLE	NY
00601	18.165273	-66.722583	ADJUNTAS	PR
00602	18.393103	-67.180953	AGUADA⊠ ⊠	PR
00603	18.455913	-67.14578	AGUADILLA	PR

$\longrightarrow$

00210	43.005895	-71.013202	PORTSMOUTH	NH
00211	43.005895	-71.013202	PORTSMOUTH	NH
00212	43.005895	-71.013202	PORTSMOUTH	NH
00213	43.005895	-71.013202	PORTSMOUTH	NH
00214	43.005895	-71.013202	PORTSMOUTH	NH
00215	43.005895	-71.013202	PORTSMOUTH	NH
00501	40.922326	-72.637078	HOLTSVILLE	NY
00544	40.922326	-72.637078	HOLTSVILLE	NY

Again, while simplistic in this project, this is often a very complicated and can require significant mathematical work to place the data into a mathematical "model" or normalize it (convert it to an acceptable range of numbers). In this example, a basic normalization is used to re-orient the minimum and maximum longitudes and latitudes to range from zero to the width and height of the display. More of the mathematical approaches to filtering are discussed in chapter six.



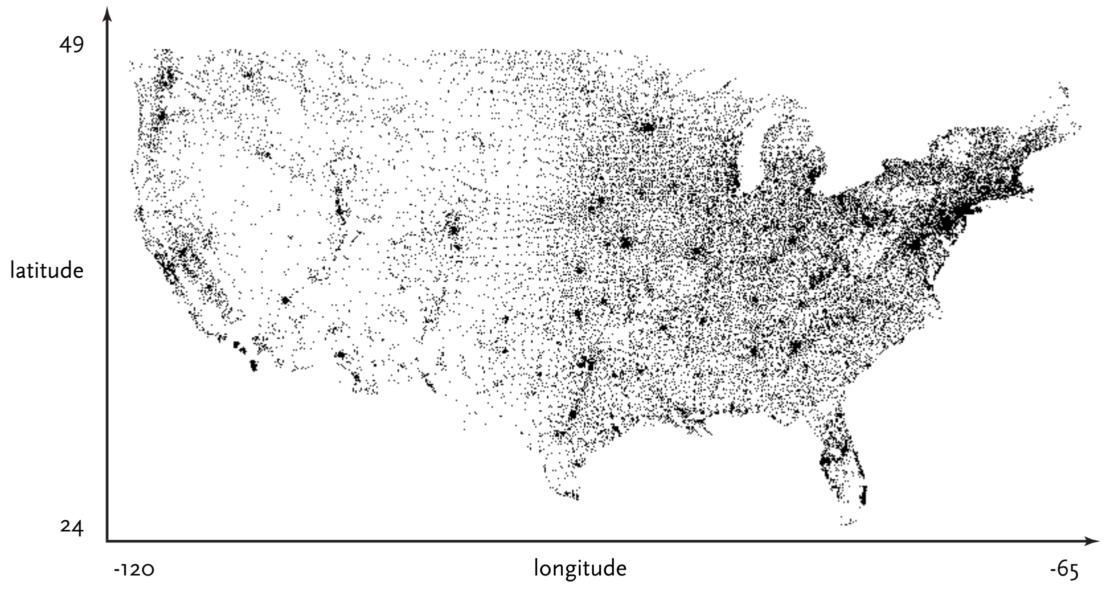
00210	43.005895	-71.013202	PORTSMOUTH	NH
00211	43.005895	-71.013202	PORTSMOUTH	NH
00212	43.005895	-71.013202	PORTSMOUTH	NH
00213	43.005895	-71.013202	PORTSMOUTH	NH
00214	43.005895	-71.013202	PORTSMOUTH	NH
00215	43.005895	-71.013202	PORTSMOUTH	NH
00501	40.922326	-72.637078	HOLTSVILLE	NY
00544	40.922326	-72.637078	HOLTSVILLE	NY
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.

2.3.4 *Mine*

This step involves basic math, statistics and data mining. The data in this case receives only simple treatment: the program must figure out the minimum and maximum values for latitude and longitude, so that the data can be presented on screen at a proper scale.

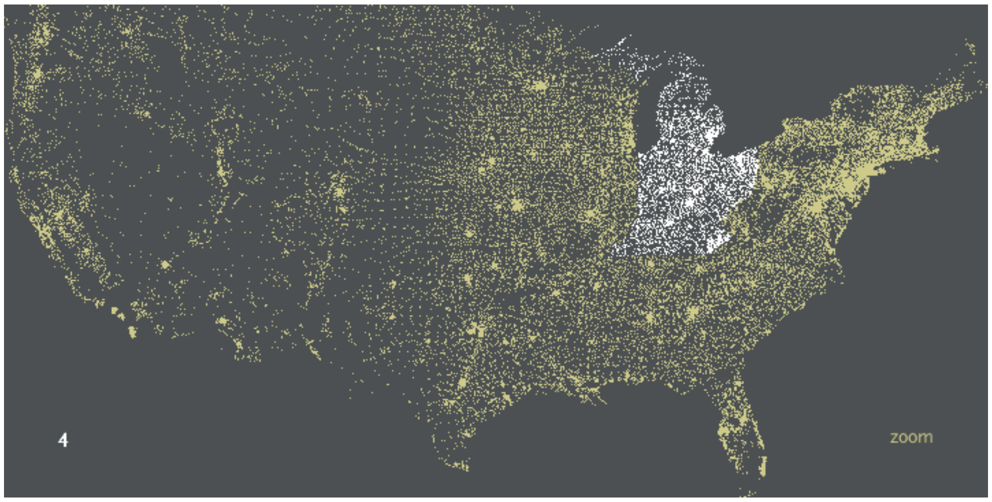
2.3.5 *Represent*

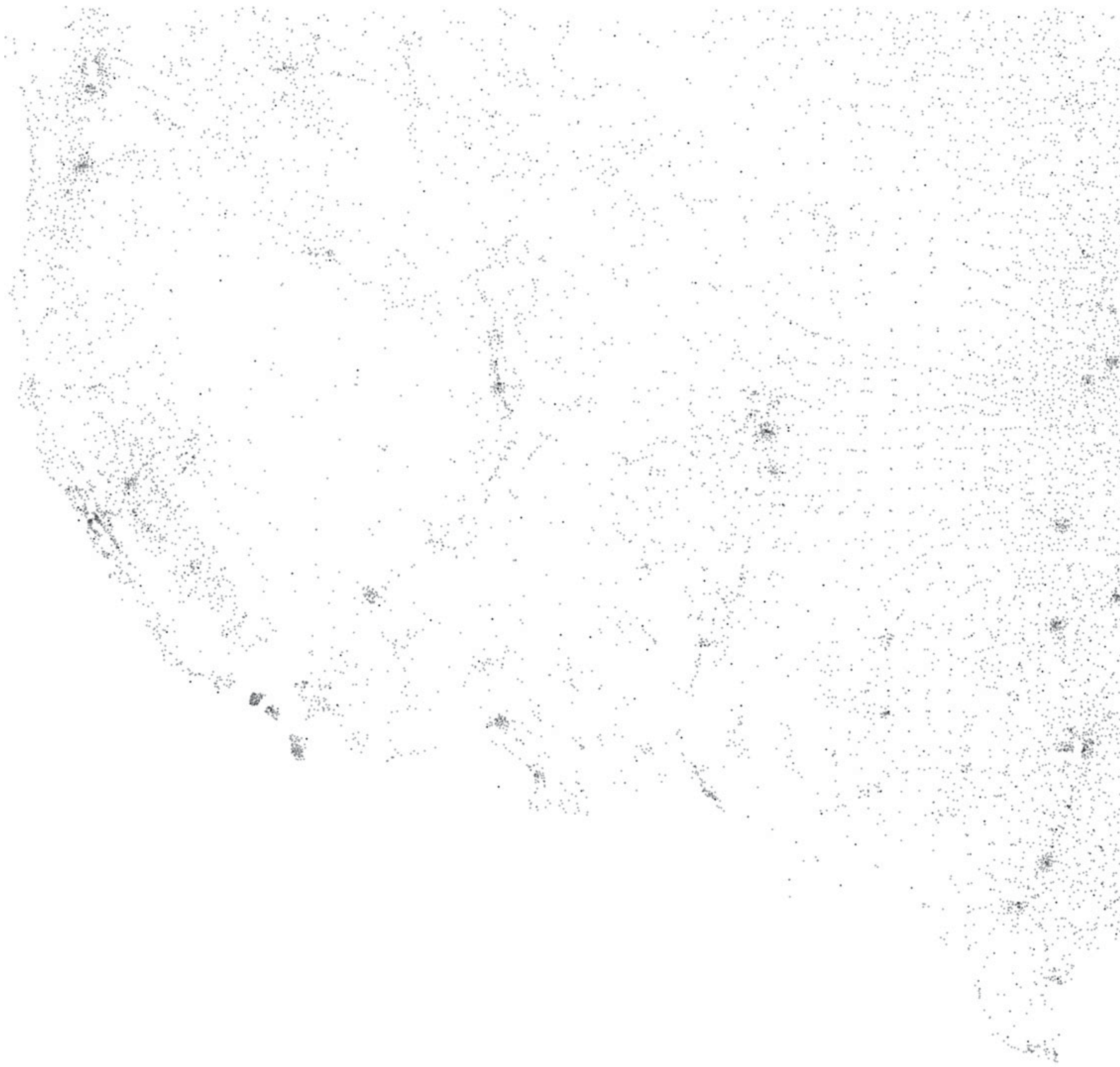
The representation step has to do with the basic form that a set of data will take. Some data are lists, others are structured like trees. In this case, each zip code has a latitude and longitude, so they can be mapped as a two-dimensional plot, with the minimum and maximum values for the latitude and longitude being used for the start and end of the scale in each dimension.



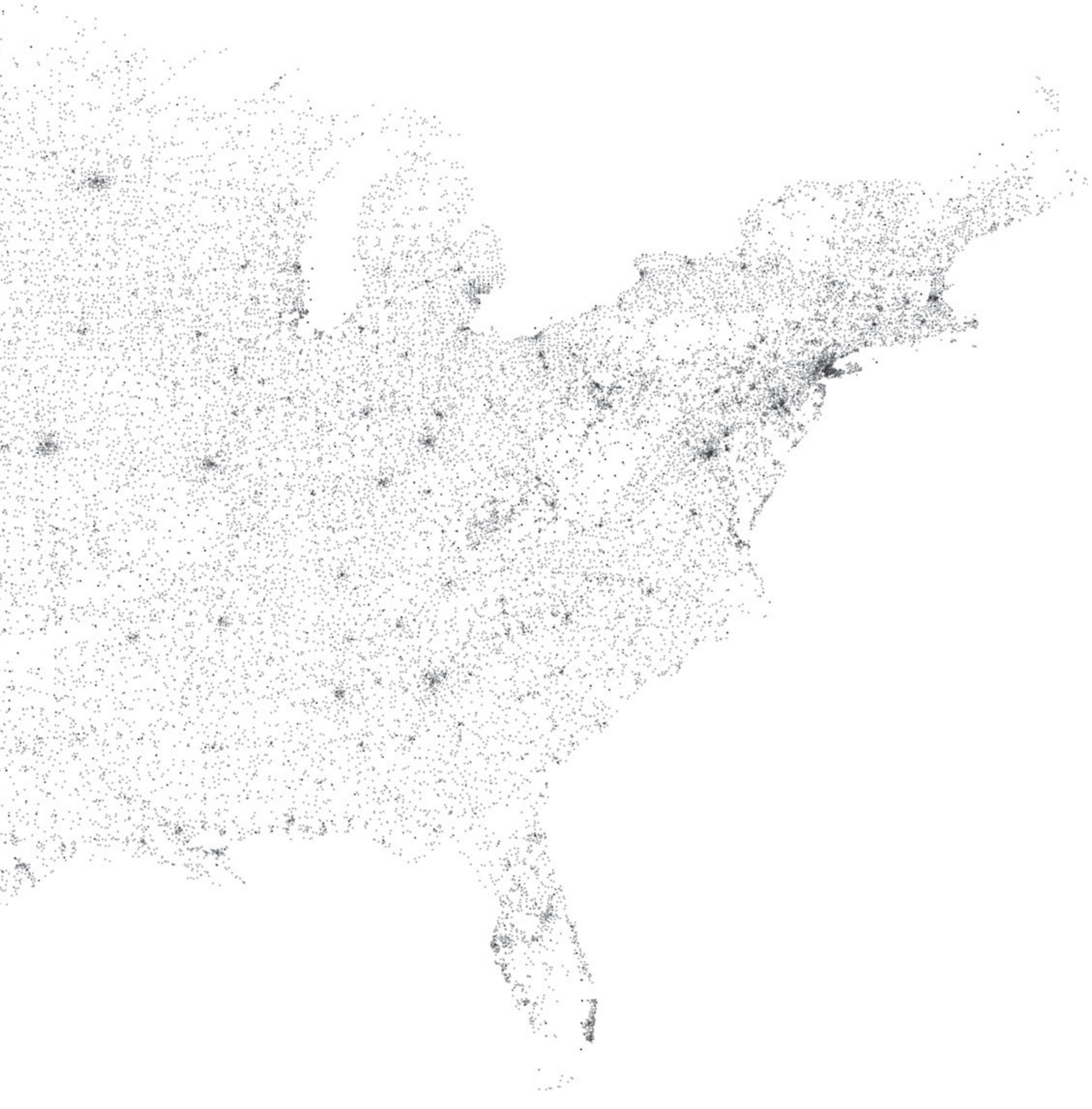
2.3.6 Refine

In the refinement step, the graphic design methods are used to more clarify the representation by calling more attention to particular data (establishing hierarchy), or changing attributes like color that have an impact on how well the piece can be read. While it doesn't reproduce well here, the on-screen coloring becomes a deep gray, and each point a medium yellow signifying that all the points are currently selected.





u.s. zip codes for the contiguous 48 states, plotted by their latitude and longitude



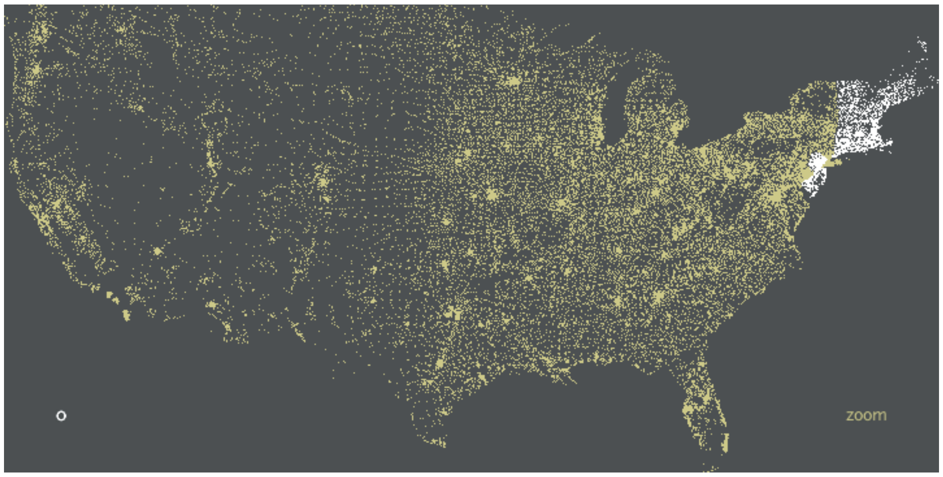
2.3.7 *Interact*

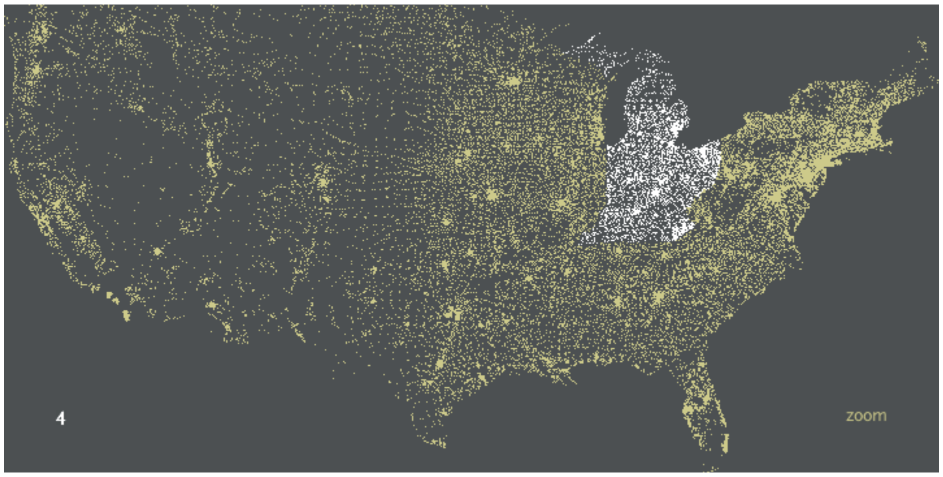
The next stage of the process adds interaction as a way to give the user a way to control or explore the data. Interaction might cover things like selecting a subset of the data (controlling the filter) or changing the viewpoint. It can also affect the refinement step, as a change in viewpoint might require the data to be designed differently.

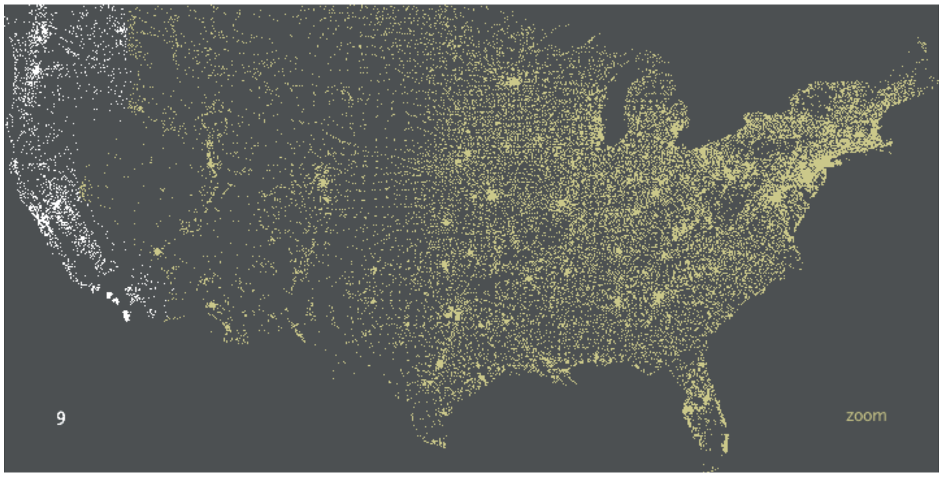
In the zipdecode piece, typing a number begins the selection of all zip codes that begin with that number. The following images show all the zip codes beginning with zero, four, and nine respectively.

As the user will often want to traverse laterally—running through several of these prefixes, holding down the shift key will allow them to replace the last letter typed, without having to hit the 'delete' key to back up.

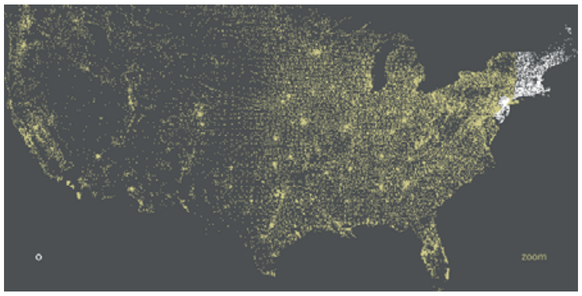
The interaction is primitive, but allows the user to very rapidly gain an understanding of how the layout of the postal system works.

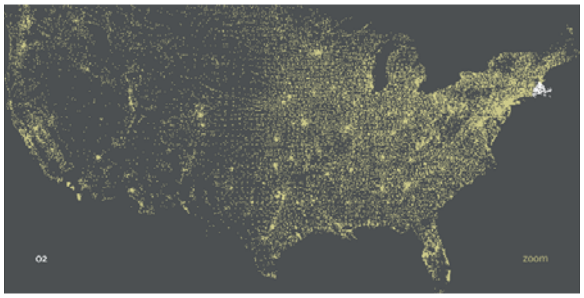

0
zoom


4
zoom


9
zoom

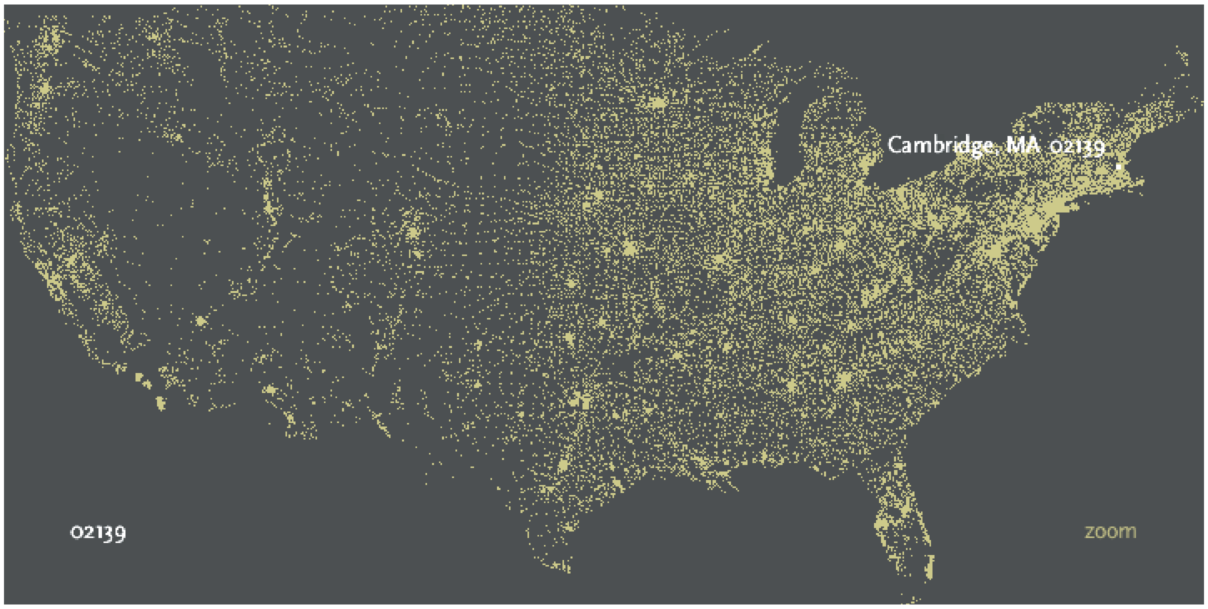
Past the initial number, the viewer can continue to type digits to see the area covered by each subsequent set of prefixes:

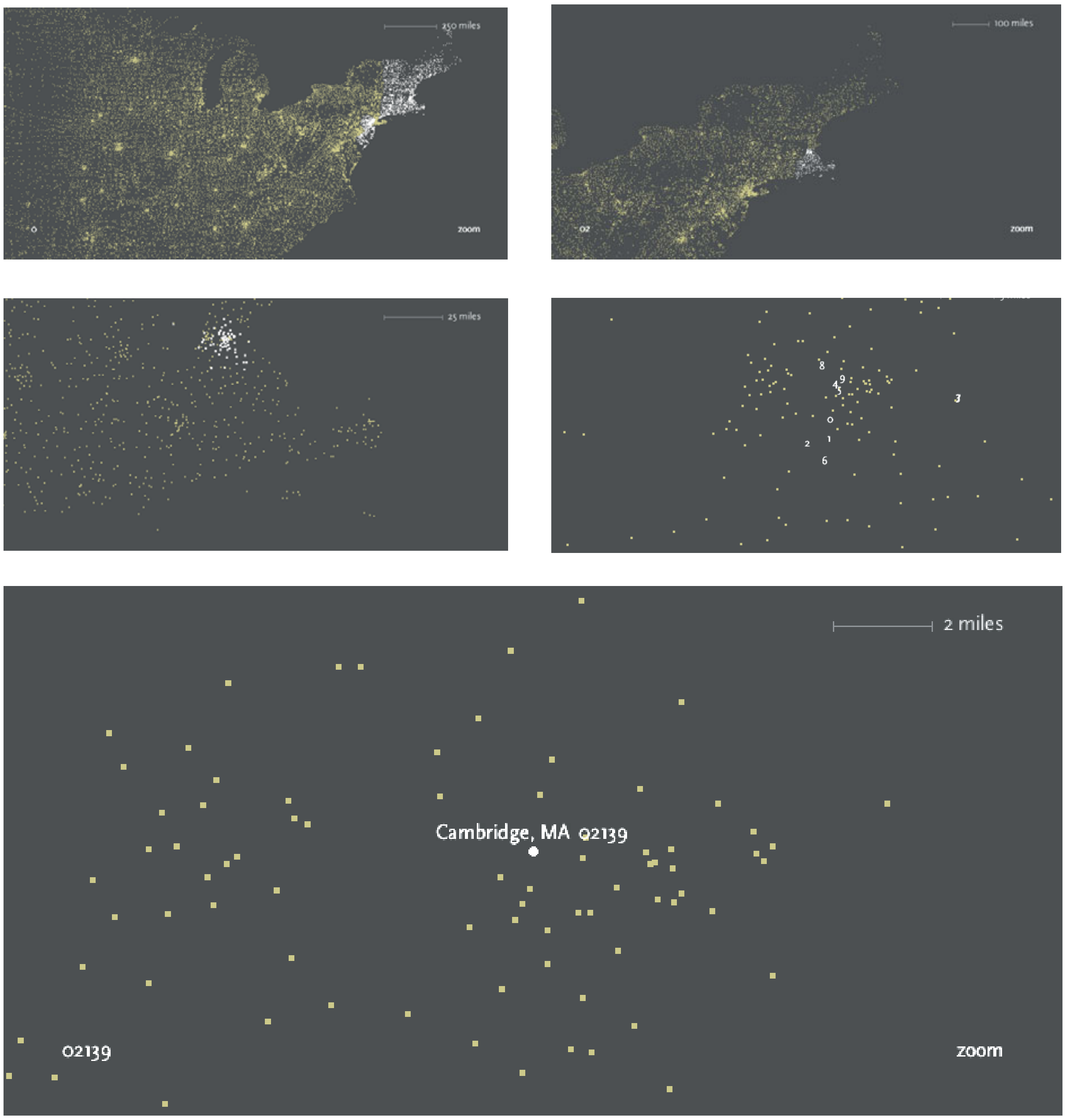




Prefix '0' is New England, '02' covers Eastern Massachusetts.

'021' limits down to entries in Middlesex County, and '0213' is a grouping of nearby cities. Finally, '02139' hones in on Cambridge, MA itself.





In addition, users can enable a ‘zoom’ feature which draws them closer to each specific choice as they’re made, to reveal more detail around the area. Each level of zoom adds more detail to the features, so that a constant rate of details is seen at each level. In the case of mapping, additional details of state and county boundaries, or other geographic features that help the viewer associate the "data" space of zip code points to what they know about the local environment.

This notion of re-layering the data as the perspective shifts is a useful aspect of computational design, a unique feature that comes from the combination of several of the steps involved.

Not visible in the steps shown so far is the kind of iteration that went into the project. Each step of the process is inextricably linked because of how they affect one another. Because of the need for the representation to fit on the screen in a compact way, the data was re-filtered to exclude territories not part of the contiguous 48 states.

$$\begin{tikzcd}
\text{acquire} \arrow[r] & \text{parse} \arrow[r] & \text{filter} \arrow[r] & \text{mine} \arrow[r] & \text{represent} \arrow[r] \arrow[ll, bend right] \arrow[llll, bend left] & \text{refine} \arrow[r] & \text{interact} \arrow[l, bend right] \arrow[lll, bend left]
\end{tikzcd}$$

The method of interaction by typing successive numbers impacted the visual refinement step, where the colors were modified to show a slow transition as points in the display were added or removed. This prevents the interaction from becoming too jarring and helps the user maintain context.

Later, the representation step affected the acquisition step, as the application was modified to show the data as it was downloaded over what might be a slow internet connection. This allows the points to slowly appear as they are first read from the data file as it is streamed over the network--employing the data itself as a "progress bar" to depict completion.

The interconnections between the steps of the Computational Information Design process helps illustrate the importance of addressing the fields as a whole.

acquire	parse	filter	mine	represent	refine	interact
live or changing data sources	modular parsers for new data sources	automation of tedious manual processes	modify parameters of statistical methods in real-time	rapid prototyping and iteration	change design rules without manual redesign	smooth transition between states to maintain context
		modify filter in real-time		juxtapose large amounts of data	computation as its own “medium”	additional information as viewpoint shifts
				try multiple representations		

2.3.8 Properties

At the intersection between these fields are the more interesting set of properties that demonstrate the strength of their combination. In terms of acquisition, consideration is given to data that can be changed, whether once a month or on a continuous basis. This opens up the notion of the focus of graphic design on solving a specific problem for a specific data set, and instead considers the meta-problem of how to handle a certain *kind* of data, that might be updated in the future.

In the filtering step, data can be filtered in real time, as it is done in the zipdecode application. In terms of visual refinement, changes to the design can be applied across the entire system. For instance, a color change can be automatically applied to the thousands of elements that require it, rather than requiring the designer to painstakingly make such a tedious modification. This is the strength of a computational approach, where tedious processes are minimized through automation.

Moving further ahead, by making these methods available to a wider audience, the field can mature into a point where “craft” is re-introduced into the medium, that the hand of the advanced practitioner can be seen in the work, even in a medium of the computer, which is typically considered algorithmic, unexpressive, and “cold.”

2.3.9 Conclusion

The zipdecode project doesn't quite solve a pressing need in the understanding of data, serves to demonstrate the principles used in a Computational Information Design approach. It received a surprising level of response from the viewing public, where as of the time of this writing it receives nearly a thousand visitors a day. This is perhaps surprising for something that might considered as boring as zip code data, and even several months after the projects initial introduction and spike of early interest.

3 Background

The visualization of information gains its importance for its ability to help us 'see' things not previously understood in abstract data. It is both a perceptual issue, that the human brain is so wired for understanding visual stimuli but extends to the notion that our limited mental capacity is aided by methods for "externalizing" cognition. One of the few seminal texts covering information visualization, "Information Visualization: Using Vision to Think" [Card, 1999] recognizes this notion in its title, and spends much of its introduction explaining it, perhaps most clearly in their citation of [Norman, 1993] who says:

> *The power of the unaided mind is highly overrated. Without external aids, memory, thought, and reasoning are all constrained. But human intelligence is highly flexible and adaptive, superb at inventing procedures and objects that overcome its own limits. The real powers come from devising external aids that enhance cognitive abilities. How have we increased memory, thought, and reasoning? By the invention of external aids: It is things that make us smart.*
>
> NORMAN, 1993

As an example of external cognition [Card, 1999] describes how a task like multiplication is made far easier by simply doing performing it on paper, rather than completely in one's head.

The idea of externalizing ideas too difficult to understand is carried out in many disciplines and is remarkably prevalent. As an example, this recent passage found in the New York Times:

> *"Since our theories are so far ahead of experimental capabilities, we are forced to use mathematics as our eyes," Dr.. Brian Greene, a Columbia University string theorist, said recently. "That's why we follow it where it takes us even if we can't see where we're going."*
>
> *So in some ways the men and women seen here scrutinizing marks on their blackboards collectively represent a kind of particle accelerator of the mind.*
>
> OVERBYE, 2003

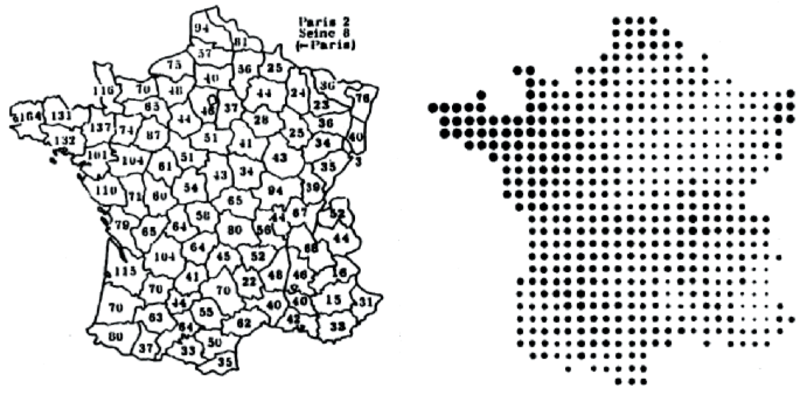
With external cognition through visual means as a basis, the process of Computational Information Design draws on the fields of Perception, Graphic Design, Information Visualization, Statistics, Data Mining, and Cartography in an attempt to bring them together as a unified approach for the effective handling of complex data.

3.1 PSYCHOLOGY OF PERCEPTION

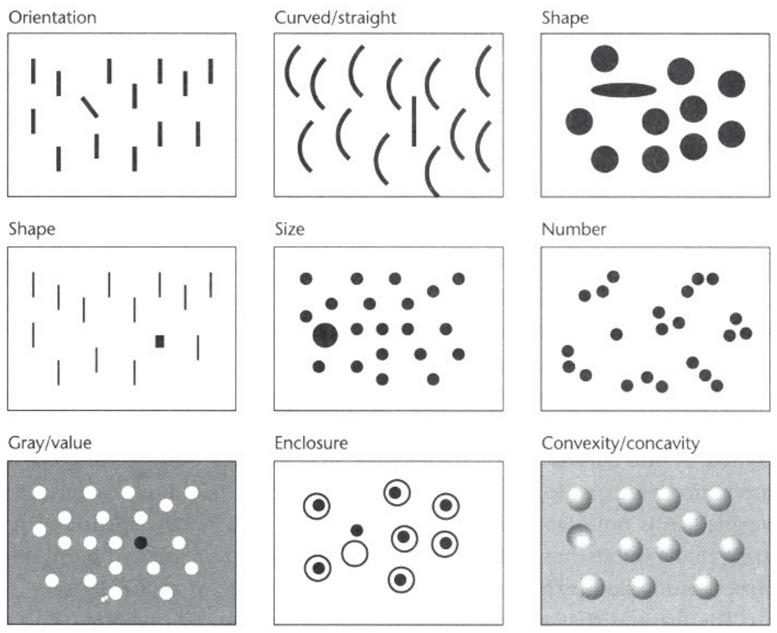
Colin Ware, an expert in the psychology of perception, makes the case for visualization by pointing out the powerful capabilities of the human visual system for understanding patterns, and by extension, data:

> *Why should we be interested in visualization? Because the human visual system is a pattern seeker of enormous power and subtlety. The eye and the visual cortex of the brain form a massively parallel processor that provides the highest-bandwidth channel into human cognitive centers. At higher levels of processing, perception and cognition are closely interrelated, which is the reason why the words "understanding" and "seeing" are synonymous.*
>
> WARE, 2000

A simple example of this point is seen in the following image from [Bertin 1983]. The image compares two methods for representing a set of sociographic data sampled from a few hundred regions of France:



Reading the left-hand image requires the viewer to search the image for the lowest and highest values, and the short-term memorization of the general layout of the numbers. On the right, a qualitative understanding of the image is immediately conveyed—that something is important in the Northwest corner, and to a lesser extent in a small region on the Eastern edge. This information is conveyed without any active viewing, meaning that it is "pre-attentive." The term is assigned to objects that are processed faster than 10 milliseconds; as compared to non-pre-attentive features requiring 40 milliseconds or more [Triesman, 1988 via Ware, 2000]. It is only one of many important aspects of perception, but is highlighted here for its importance.



The diagram above (taken from Ware's text) demonstrates several pre-attentive features, he continues further, listing them as:

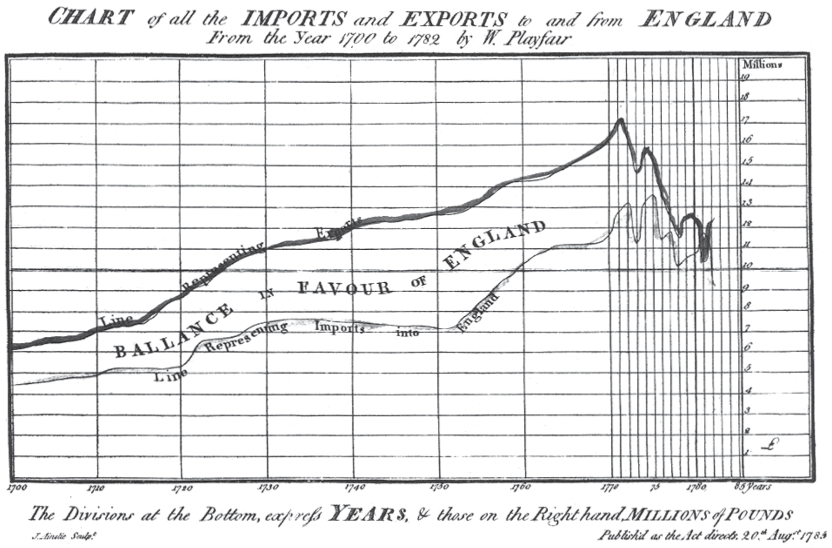
> *Form* – line orientation, line length, line width, line collinearity, size, curvature, spatial grouping, added marks, luminosity.
>
> *Color* – hue, intensity
>
> *Motion* – flicker, direction of motion
>
> *Spatial position* – 2D position, stereoscopic depth, convex/concave shape from shading.

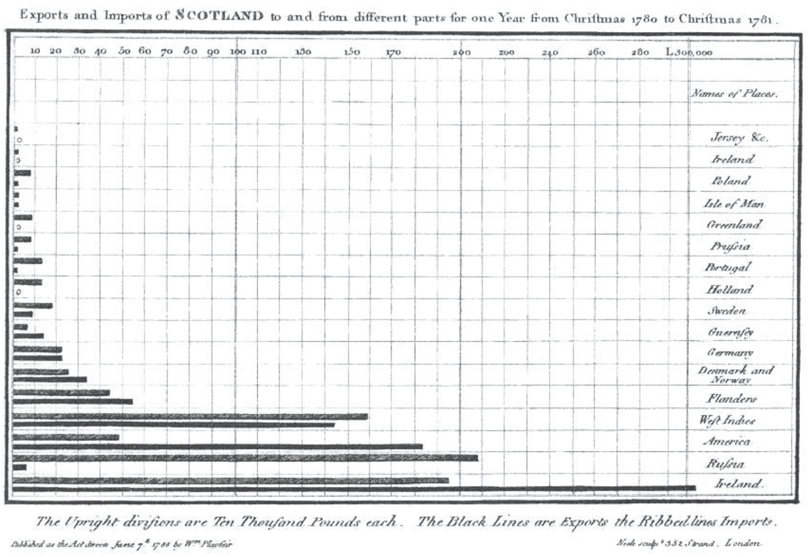
The task is to make the most relevant aspects of the data apparent through pre-attentive features (along with consideration to other aspects of perception). However it is not enough to haphazardly assign one such feature to each variable of interest. Instead, the field of graphic design can be employed, which provides the skills to weigh the variables against one another in the context of the overall data set, and to handle the resulting image as a whole.

3.2 GRAPHIC DESIGN AND DATA GRAPHICS

Given the abilities of the human visual system, it next becomes necessary to consider how to best present the data visually. Graphic design is the broad title for the field that addresses this issue, and data graphics (also called information design) is a subset that is specifically focused on representation of data, as opposed to projects like a corporate logo.

William Playfair is widely held to be the father of data graphics. His text, The Commercial and Political Atlas [Playfair, 1786] was a series of 44 charts of time-series data that sought to explain the increasingly complicated economy of the day. One such example is below, which places a perspective on the trade balance "to and from England, from the year 1700 to 1782."



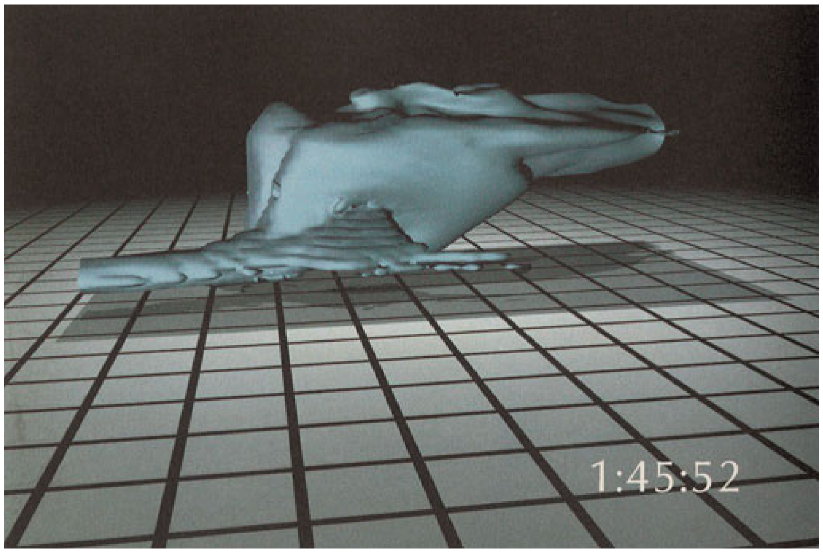


He is also believed to be the inventor of the bar chart, which he created because one year's data was missing from another diagram, and he did not want to show continuity where there was none [Tufte, 1983, p. 33]

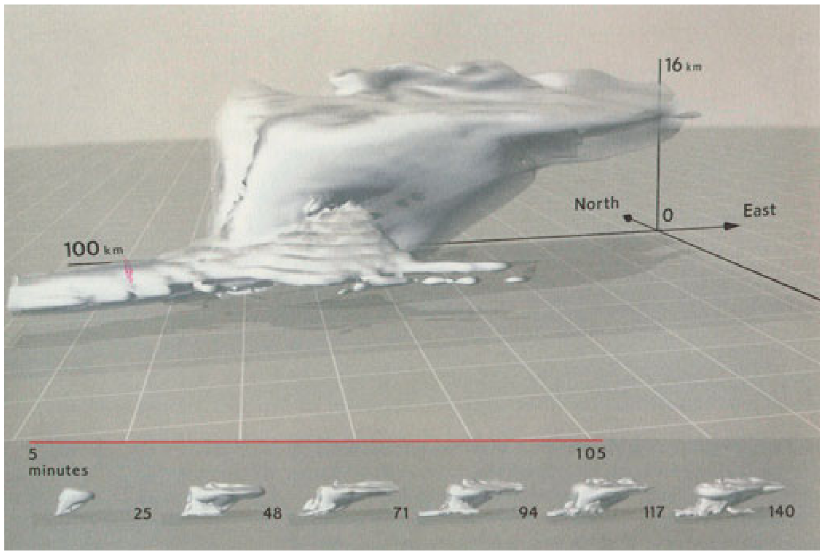
Edward Tufte is often cited as the modern day standards-bearer with regard to data graphics, primarily for his three texts [Tufte, 1983, 1990 and 1997] on the subject. These books enjoy a best-selling status and much acclaim perhaps most significantly for their ability to convey the principles of proper design to an exceptionally wide audience. The books are easy to read and visually appealing with specific rules (such as "reduce non-data ink") for proper design. Such rule sets are comforting for an unfamiliar audience, and a useful introduction to the field.

The principles are a well-ordered compendium of the contemporary knowledge of codified as a series of proper practices for the design of data. Contemporary graphic designers, for instance, will often have procured identical knowledge simply because it was embedded in their training (though one could argue that the texts themselves have had their impact on design training as well).

One of the most important points made in Tufte's texts are his clear articulation of the importance of visual design for understanding data, noting that "graphical excellence consists of complex ideas communicated with clarity, precision, and efficiency," a divergence from the expectation that somehow visual design serves to make data pretty or entertaining.



The example above from [Tufte, 1997] shows the redesign of a visualization of storm cloud formation. While the final image is generally considered more attractive, more importantly it's far easier to read through the addition of actual scale of the units in use (answering the questions "by how much?" and "in comparison to what?"). Grid lines, used to depict size are decreased in their importance so that they no longer dominate the image. A less dramatic use of lighting model is used, meaning that the cloud actually looks like a cloud. The final diagram is engaging and



likely to draw in even viewers who might not be interested in the subject in question.

The books fall short in addressing three important aspects of contemporary information design problems. Notably absent are situations in which the data in question is undergoing continual change. In addition, none of the examples have the complexity of something as vast as the human genome. And finally, the texts have little to say of methods for interacting with the data as a way to learn about it. Information Visualization attempts to deal with some of these aspects, however there is often little regard to the visual design principles espoused by Tufte and the design community.

3.3 INFORMATION VISUALIZATION

Visualization as a sub-field of science, statistics, and graphics has only been recognized as its own entity since the mid- to late-80s. The depth of seminal work is in line with that of a young field, but finds its strength in background drawn from years of statistics and graphic design.

A succinct definition of Visualization is found in [Card *et al*, 1999]

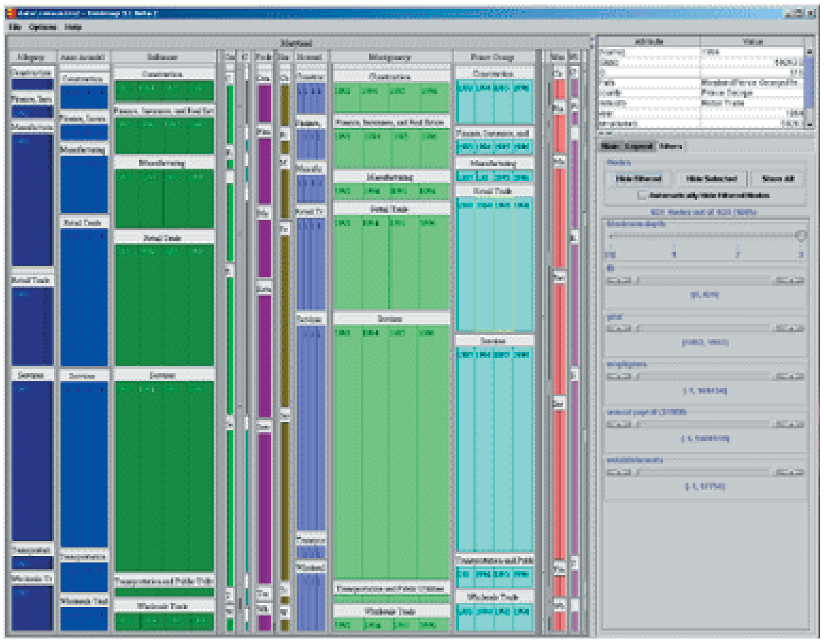
> *Visualization* – the use of computer-supported, interactive, visual representations of data to amplify cognition.

Visualization is concerned with non-abstract data sets, for example imagery from weather data or an animation describing the movement of a fluid. For this kind of data, representations or physical analogues already exist. When studying scientific phenomena, this is sometimes called scientific visualization (although that term tends to be poorly defined, since it can apply to the next case as well).

> *Information Visualization* – the use of computer-supported, interactive, visual representations of abstract data to amplify cognition.

Information Visualization, by contrast, is concerned with making an abstract set of information visible, usually in circumstances where no metaphor exists in the physical world. It is also sometimes called data visualization, referring to the raw material used to create the image. This thesis is concerned solely with information visualization.

An emphasis on the quality of visual design is notably absent from the generally accepted definition above, and this is born out in the majority of work in the field. Perhaps visual design is generally considered to be of lower importance, as it is often perceived as a non-quantifiable and endeavor to make the representation more subjectively attractive. While that may be a side effect, the issue is not that the visual design should be "prettier". Rather, that the approach of the visual designer solves many common problems in typical information visualization. In addition, what are commonly considered cosmetic tweaks may be less important for a simple diagram (tens or hundreds of elements), but for a complex data set (thousands of elements) they become extremely important, because what were minor problems in the diagram of a smaller data set are vastly magnified in a larger one.



A simple example comes from the TreeMap project [Bederson, 2002 and *www.cs.umd.edu/hcil/treemap/*] at the University of Maryland. Ben Shneiderman's Human-Computer Interaction Laboratory (now run by Bederson) has been a standards bearer for the Information Visualization field for having popularized "dynamic queries" [Shneiderman, 1994], tree maps, and others. The image that follows is taken from a recent version of the TreeMap software.

The representation suffers from overall layout issues, as well as minute details combine to create a larger problem. Too much of the screen is taken with a series of sliders on the right, rather than focusing on the data itself (including a large gap of dead space in the bottom right), so instead the narrow sliders could easily be moved to the bottom of the screen for better space economy—meaning that the data itself could take up an additional ten to twenty percent of the screen.

At the micro level, details like the frame around each block of data, and the method used for drawing the heading, cause a great deal of visual noise. The borders around the smaller blocks wind up larger than the data itself, which is confusing for the user. The labels, having the same rectangular shape as the data itself, creates additional confusion. Removing the labels and reversing their text out of the colored blocks would be an improvement in clarity, allowing each block (the data itself, the most important thing in the image) to recover more screen area.

The issue is about diagrams that are accurate, versus those that are understandable (or subjectively, those that viewers prefer to look at). An image under the Information Visualization definition may properly represent all aspects of the data set (such as the example above), while failing to be as effective as possible in helping the viewer understand the data in question.

3.4 DATA ANALYSIS AND STATISTICS

Data Analysis, is something of an extension to Statistics. The statistician John Tukey developed Exploratory Data Analysis largely in response to what he saw as a field that had become too attached to the application of specific methods to data rather than the original intention of the field, which was to come to an understanding about the data in question.

> *For a long time, I have thought that I was a statistician, interested in inferences from the particular to the general. But as I have watched mathematical statistics evolve, I have had cause to wonder and to doubt. And when I have pondered about why such techniques as the spectrum analysis of time series have proved so successful, it has become clear that their 'dealing with fluctuations' aspects are, in many circumstances, of lesser importance than the aspects that would already have been required to deal effectively with the simpler*

case of very extensive data, where fluctuations would no longer be a problem. All in all, I have come to feel that my central interest is in data analysis, which I take to include, among other things: procedures for analyzing data, techniques for interpreting the results of such procedures, ways of planning the gathering of data to make analysis easier, more precise or more accurate, and all the machinery and results of (mathematical) statistics which apply to analyzing data.

TUKEY, 1962

The quote shows Tukey's return to what initially made him curious about statistics—analysis, interpretation, as well as simplification and improvement of the methods therein. This focus on curiosity about the data, rather than the field itself, drove Tukey's many accomplishments (including the co-invention of the fast fourier transform), and laid the groundwork for the field of Exploratory Data Analysis. (He is also credited with coining the terms "hardware", "software", and "bit"). True to form, he introduces his seminal "Exploratory Data Analysis" book as:

...looking at data to see what it seems to say. It concentrates on simple arithmetic and easy-to-draw pictures. It regards whatever appearances we have recognized as partial descriptions, and tries to look beneath them for new insights. Its concern is with appearance, not with confirmation.

TUKEY, 1977

This is perhaps best exemplified by his "stem and leaf" plots, an example taken from the same text is shown at the top of the next page.

Consider 17 used car prices: $250, 150, 795, 895, 695, 1699, 1499, 1099, 1693, 1166, 688, 1333, 895, 1775, 895, 1895, and 795. To see what these data "look" like, a stem and leaf plot can be used. The four steps below show how the plot is constructed. The plot is constructed by first ordering the numbers by their leading digits (signifying 1 for 100 through 18 for 1800). The second image shows those trailing digits being removed from the diagram because they're redundant. Next, the right most digit is removed, since they're less important than the others (the difference in price between $1490 and $1495 is irrelevant to understanding the overall trends in the data set).

Finally, the right most image is the stem & leaf plot, a compact representation of the data set. The left-hand side are the leading digits of the

1	150	1	1**50**	1	1**5**0	1*	**5**
2	250	2	2**50**	2	2**5**0	2	**5**
3		3		3		3	
4		4		4		4	
5		5		5		5*	
6	695, 688	6	6**95**, 6**88**	6	6**9**5, 6**8**8	6	**98**
7	795, 795	7	7**95**, 7**95**	7	7**9**5, 7**9**5	7	**99**
8	895, 895, 895	8	8**95**, 8**95**, 8**95**	8	8**9**5, 8**9**5, 8**9**5	8	**999**
9		9		9		9*	
10	1099	10	10**99**	10	10**9**9	10	**9**
11	1166	11	11**66**	11	11**6**6	11	**6**
12		12		12		12	
13	1333	13	13**33**	13	13**3**3	13*	**3**
14	1499	14	14**99**	14	14**9**9	14	**9**
15		15		15		15	
16	1693, 1699	16	16**93**, 16**99**	16	16**9**3, 16**9**9	16	**99**
17	1775	17	17**75**	17	17**7**5	17*	**7**
18	1895	18	18**95**	18	18**9**5	18	**9**

numbers, the right-hand side is the second-to-last digit, since the plot uses increments of $10. So the first line is for $150, since the leading digit is 1, the next is 5, meaning 15, which is multiplied by the $10 increment (signified by the asterisk) to make $150. While not immediately intuitive, it can be learned in a matter of minutes, and is a powerful tool for understanding numbers by scratching them down on paper.

So for the series of numbers that took two lines, they can now be seen as a distribution, with the digits at right signifying how many of each value are present. Immediately, a clustering can be seen around $600, $700, and $800. In addition, a glance down the column tells us that 9s are prevalent in the pricing (looking at the original prices will verify the fact that there are several prices ending 95 and 99).

Even this simple representation reveals much about the data. This is fascinating for two reasons. First, that an image so basic can be both quickly created and rapidly understood. The method is perhaps non-obvious on initial observation, but the learning curve is brief and worth the payoff. Second, the diagram itself shows that the representation of the data need not be visually intricate or over-designed in order to convey the data. The latter point is important to be consider with the emphasis on graphic design—that the design is not a matter of fancy graphics or visual tricks, rather that it is a means to an end for creating the cleanest, most understandable diagram possible.

3.5 DATA MINING

Data mining is essentially statistics mixed with computational methods for handling large data sets, in particular, it is a combination of theory and application between statistics and databases. "The science of extracting useful information from large data sets or databases is known as data mining." [Hand, 2001] The field concerns itself with a large-scale version of the type of exploratory data analysis pioneered by Tukey, and backed with databases and additional software. However, in the process it often loses the visual aspect of Tukey's foraging through numbers. For instance, when confronted with a database, one might want the computational equivalent of an interactive "stem and leaf" diagram, so that the user could quickly become familiar with the contents of the database before engaging in more specific analysis. An offshoot of Tukey's work in "Exploratory Data Analysis" is centered on such ideas, but falls short of the simplicity of what can be observed from his common-sense focus to statistics and data.

Computational Information Design takes the strength of data mining for handling large data sets and mixes the analysis methods back in, while also placing an emphasis on information visualization and graphic design.

3.6 INFORMATION VISUALIZATION AND DATA MINING

In one of the rare crossover events between the fields, a collection of articles "Information Visualization in Data Mining and Knowledge Discovery" [Fayyad, Grinstein, & Ward, 2002] was published. One of its editors, Georges Grinstein, describe it as "the result of two workshop whose goals were to open up the dialog between researchers in visualization and data mining, two key areas involved in data exploration." The text itself is "a collection of papers, some of which are tutorial, some idealized positions, some seminal in nature, and some provocative." It is notable that this sort of collaboration between the members of the two fields was considered "new" or "different" from regular practice, as both fields are at the forefront of data understanding, yet in their respective disciplines.

In one of the introductory papers [Grinstein & Ward, 2002], Grinstein describes a set of steps that are quite similar to the process I was developing as Computational Information Design:

> *data select* – get a subset of the data
>
> *data manipulate* – smooth, filter, interpolate
>
> *representation* – choose what dimension is assigned to what variable, or change the colormap
>
> *image operations* -– changing orientation and viewing
>
> *visual interactions* – navigate & manipulate elements

This covers much of what is considered in this thesis, however notably absent is the aspect of visual refinement, as espoused for instance by Tufte, that proper tuning of the graphical aspects of the image are essential to understanding data.

3.7 CARTOGRAPHY

Additional background can be found in mapping and cartography. It also encompasses several of the fields already discussed. Data for geographic features is notoriously voluminous, and the resulting representation extremely dense. Yet cartographers have mastered the ability to successfully organize geographic data in a manner that communicates effectively. Cartography is a useful model because it synthesizes illustration, information design, statistics, and most often employs technological tools for implementation.

Images from Microsoft MapPoint 2002

There are two general categories of maps. One is used to gain an understanding of high-level geographic features like the location of mountains in the United States or comparing the size of New England to the rest of the country. Conceivably, this is the goal of the current genomic maps, to understand a broad overview of the data, and perhaps pick up a few interesting parts. For instance, in the recently published mouse genome, one of the findings was a remarkable similarity to the human genome, with a few exceptions that were mostly related to 'mouse like' features, i.e. an improved sense of smell. The goal would be well made maps of the human and mouse genomes that could be visually compared and such differences quickly revealed.

A second type of map, such as a street atlas, has a more specific purpose, in being useful to provide navigation. Genome maps could also be treated in such a way, having a more directed purpose than simply showing all the data. It is necessary here to consider what kind of task, if not navigation, is being supported. This directs the pursuit of a more relevant picture.

Extending cartography in the direction of Computational Information Design, one might look at how would relate to genetic data. In such a case, four aspects require additional consideration, which fall along the lines of the process outlined in the previous chapter. First, the amount of data necessitates computational methods for handling it. To take an example from genomics, a file listing a set of features currently known for human chromosome 20 (one of the shortest) contains 62,000 lines. It would be impossible for a designer to manually handle this much data, much less to have the resulting image quickly become irrelevant, as more information becomes known.

That situation leads to the second consideration: that the input data is undergoing continual change. Genomic data is in a process of discovery, where new features are continually added and refined. In the months following the publication of the fruitfly genome, a series of contaminated data was discovered, causing the re-release of the information, and rendering its published map somewhat inaccurate. The speed of discovery is accelerating, with no expectation of any kind of 'completion' in the near future. This expectation of change suggests that rather than focusing on a final outcome, genomic cartography should be a flexible process around a dynamic diagram that can handle change. To attain this flexibility, software presentation methods must be grounded in variability—the positioning of elements must be semi-automatic, based on rules set by the cartographer. Rather than simply applying rules of graphic design, the designer must be able to abstract some of these rules and implement them as active software elements.

Third, while it is useful to create large format printed images like traditional maps, the more common rendition of this work will be in interactive software. In such software, the end-user might select feature layers based on what is most important for their research, or use viewing techniques to zoom into a section, or pan along a data set.

The fourth, and perhaps most difficult issue, is that the process needs to be accessible to a wider audience (i.e. designers, or computational biologists, for the example in question). In additional to the lack of process, few tools exist that can cover most of the issues presented.

3.8 THE VISIBLE LANGUAGE WORKSHOP

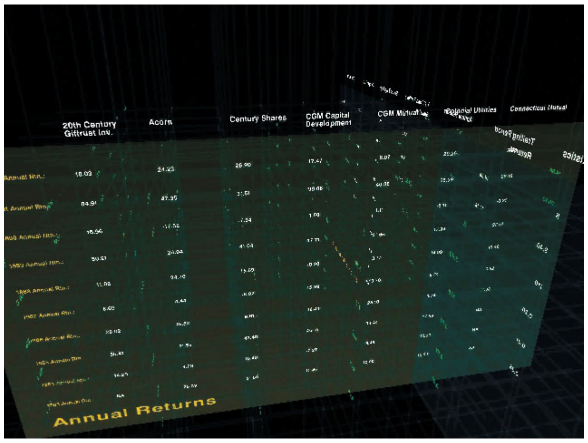
The Visible Language Workshop, founded by Muriel Cooper, operated from roughly 1976 until she passed away in 1994. The group embraced the collision of visual design and computer science and sought out how the combination would change the field of design.

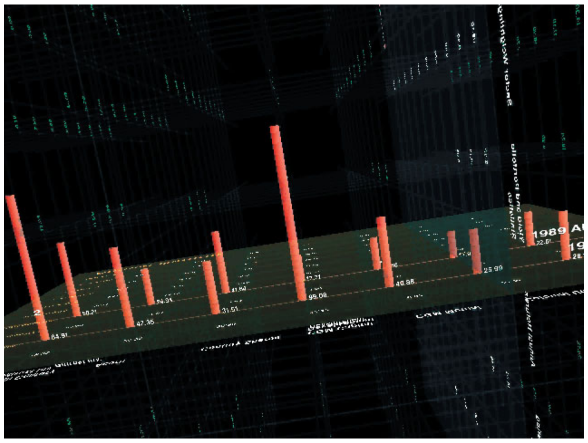
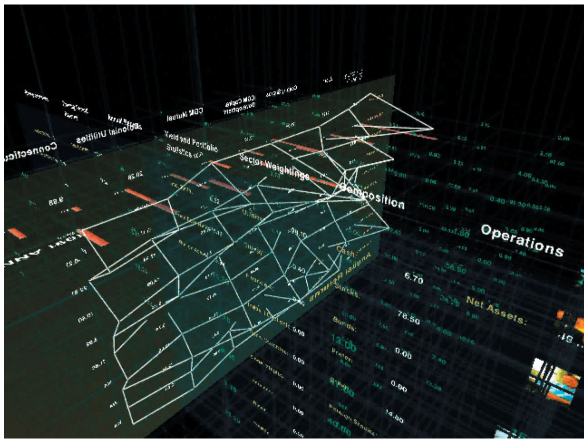
Their work studied themes of automatic layout, programmatic generation of design alternatives, kinetic information display, and a range of other themes. Such work highlights the fact that the idea of a mixture of disciplines as in Computational Information Design is not a new idea, so much as a natural evolution of where the fields are headed.

3.8.1 Financial Viewpoints

Typical of this work was the Financial Viewpoints project by Lisa Strausfeld [Strausfeld, 1995]. The project examined mutual fund data, juxtaposing several spreadsheets together in three dimensional space. The planes were essentially those of a standard two dimensional spreadsheet, intersecting one another where they linked, for comparison and analysis. The project was both a step forward in information graphics, particularly as employed on the computer, but also touched on aspects of interaction and visual refinement in a thorough way. It is an example of what could be considered Computational Information Design, where the many stages of the process are linked together tightly.

One missing factor, however, was the fact that the tools to create this work were expensive graphics workstations and complex programming languages and support libraries. An opening for such work in current day practice is that such a project can now be built on an inexpensive PC, and work is being done (such as the *Processing* system described in chapter six), to bring the necessary programming skills to a wider audience. By widening the field, the focus can switch from the VLW-era questions regarding what does the computer mean for graphic design, to a critical discourse about what type of work is interesting and relevant in the context of the computational media. Three dimensional spreadsheet systems have not yet taken hold as a standard means of understanding multiple layers of data as implemented in Financial Viewpoints, perhaps because while an interesting concept, it diverged too far from standard modes of understanding data employed by the



Trailing Period Returns
Annual Returns
Yield and Portfolio Statistics
Sector Weightings
Composition
Ytd. Total Rtn.: 14.35
Yr. Total Rtn.: 17.18
1992 Annual Rtn: 20.26
1991 Annual Rtn: 25.99
1990 Annual Rtn: -5.11
1989 Annual Rtn: 17.94
1988 Annual Rtn: 14.09
1987 Annual Rtn: -6.24
1986 Annual Rtn: 9.74
1985 Annual Rtn: 17.59



target audience. Where at the time the interesting aspect of the project was the implementation (previously no such thing existed), rather than the practicality of use, more contemporary discourse can instead push forward based on such lessons because the implementation is far less of an issue.

3.8.2 *Improvisational Design*

Another notable product of the Visible Language Workshop is the work of Suguru Ishizaki, whose Ph.D. thesis [Ishizaki, 1995, later published as Ishizaki, 2003] described a process of design for dynamic media that used the metaphor of improvisational performance. Ishizaki describes the need for such a model because a designer cannot address every single design task in an era of always-on internet connections and ever-changing information.

The thesis describes the computational designer as more of a choreographer, whose task is to order and move the elements of a continually changing space. He addressed the process because:

> *...in digital media, designers often find it impossible to design a solution to a particular problem. Instead, they must design a way of designing, or a process, in the form of a computer program that can generate design solutions at run time.*
>
> ISHIZAKI, 2003

But such solutions are fraught with problems, because of a tendency for a breakdown between artifacts generated by machines versus humans: "*automatic* is a bad word for design, implying average quality products." To clarify, he notes that the computer is acting "on behalf of," not "instead of" the designer. He balances the automation of the computer against the reasoned hand of the designer to build systems that, rather than *replacing* designers, augment the abilities of the designer or open possibilities where there were none before. This distinction is important, because it provides more a realistic attribution to the abilities of machines (primarily to reduce repetitive tasks) versus those of human beings (ability to reason and make complex decisions). Without a major breakthrough in artificial intelligence, this distinction will hold for many years to come.

4 Advanced Example

This chapter describes a study of methods to understand the differences in the genomes of multiple people. The text first covers the scientific background of research in genetic variation, along with a survey of previous approaches for representation of the data, and then describes how to use Computational Information Design to develop a more in-depth approach for analysis and visualization.

4.1 INTRODUCTION TO GENETIC DATA

Inside the nucleus of very cell of the human body, 3.1 billion letters of genetic code can be found. The information is stored on 23 pairs of tightly wound chromosomes (shown above), long chains of the nucleotide *bases* adenine, cytosine, thymine, and guanine. These four bases are commonly depicted using long strings of A, C, G, and T letters, and contain the entire set of instructions for the biological construction of that individual. This blueprint is known as *the human genome*.

Amongst the billions of letters are some 35,000 *genes* [International Genome Sequencing Consortium, 2001], which are subsections known to contain a specific set of instructions for building proteins (or other materials that handle how proteins are used). A typical gene, approximately 1600 letters in length, is shown below:

ACATTTGCTTCTGACACAACTGTGTTCACTAGCAACCTCAAACAGACACC**ATGGTGCATCTGACTCCTGAGGAGAAGTCTG**

CCGTTACTGCCCTGTGGGGCAAGGTGAACGTGGATGAAGTTGGTGGTGAGGCCCTGGGCAGGTTGGTATCAAGGTTACAAG

ACAGGTTTAAGGAGACCAATAGAAACTGGGCATGTGGAGACAGAGAAGACTCTTGGGTTTCTGATAGGCACTGACTCTCTC

TGCCTATTGGTCTATTTTCCCACCCTTAGG**CTGCTGGTGGTCTACCCTTGGACCCAGAGGTTCTTTGAGTCCTTTGGGGAT**

CTGTCCACTCCTGATGCTGTTATGGGCAACCCTAAGGTGAAGGCTCATGGCAAGAAAGTGCTCGGTGCCTTTAGTGATGGC

CTGGCTCACCTGGACAACCTCAAGGGCACCTTTGCCACACTGAGTGAGCTGCACTGTGACAAGCTGCACGTGGATCCTGAG

AACTTCAGGGTGAGTCTATGGGACGCTTGATGTTTTCTTTCCCCTTCTTTTCTATGGTTAAGTTCATGTCATAGGAAGGGG

ATAAGTAACAGGGTACAGTTTAGAATGGGAAACAGACGAATGATTGCATCAGTGTGGAAGTCTCAGGATCGTTTTAGTTTC

TTTTATTTGCTGTTCATAACAATTGTTTTCTTTTGTTTAATTCTTGCTTTCTTTTTTTTTCTTCTCCGCAATTTTTACTAT

TATACTTAATGCCTTAACATTGTGTATAACAAAAGGAAATATCTCTGAGATACATTAAGTAACTTAAAAAAAAACTTTACA

CAGTCTGCCTAGTACATTACTATTTGGAATATATGTGTGCTTATTTGCATATTCATAATCTCCCTACTTTATTTTCTTTTA

TTTTTAATTGATACATAATCATTATACATATTTATGGGTTAAAGTGTAATGTTTTAATATGTGTACACATATTGACCAAAT

CAGGGTAATTTTGCATTTGTAATTTTAAAAAATGCTTTCTTCTTTTAATATACTTTTTTGTTTATCTTATTTCTAATACTT

TCCCTAATCTCTTTCTTTCAGGGCAATAATGATACAATGTATCATGCCTCTTTGCACCATTCTAAAGAATAACAGTGATAA

TTTCTGGGTTAAGGCAATAGCAATATCTCTGCATATAAATATTTCTGCATATAAATTGTAACTGATGTAAGAGGTTTCATA

TTGCTAATAGCAGCTACAATCCAGCTACCATTCTGCTTTTATTTTATGGTTGGGATAAGGCTGGATTATTCTGAGTCCAAG

CTAGGCCCTTTTGCTAATCATGTTCATACCTCTTATCTTCCTCCCACAG**CTCCTGGGCAACGTGCTGGTCTGTGTGCTGGC**

CCATCACTTTGGCAAAGAATTCACCCCACCAGTGCAGGCTGCCTATCAGAAAGTGGTGGCTGGTGTGGCTAATGCCCTGGC

CCACAAGTATCACTAAGCTCGCTTTCTTGCTGTCCAATTTCTATTAAAGGTTCCTTTGTTCCCTAAGTCCAACTACTAAAC

TGGGGGATATTATGAAGGGCCTTGAGCATCTGGATTCTGCCTAATAAAAAACATTTATTTTCATTGC

Even within the gene, commonly all not all the letters are used to code for the protein. The coding regions, shown here in a darker color, are known as *exons*, and are mixed with other material that might regulate or provide no additional function, called *introns*.

4.2 CODING SEQUENCES AND TRANSCRIPTION

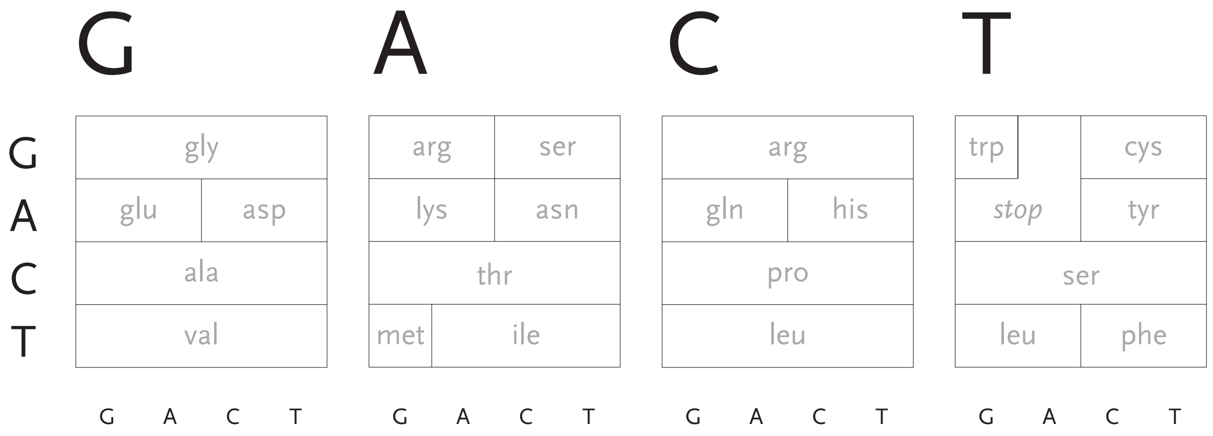
In fact, the transcription process uses uracil (denoted by a U) instead of thymine (T), but T is used in the description here for sake of simplicity.

During the process of *transcription*, a gene is copied by the cell's machinery, and everything but the exons are clipped out of the copy. The result for this gene is that the three exons are placed in succession to one another. The transcription process reads the A, C, G, and T letters, every three of which specify a particular *amino acid*. The table shown here shows the amino acid produced for every possible three letter set (called *codons*). For instance, when the cell reads A, C, and then G, the amino acid "threonine" will be produced.

The amino acid that is chosen depends on each of the letters in succession, a three-dimensional problem where the outcome is dependent on the successive refinement of possible amino acids, based on the order in which the letters are used. Each letter affects the number of possible choices for the next, each having slightly less importance than the letter before it, and the final letter has the least importance because the amino acid is often chosen based on the first two, a situation known as *four-fold degenerate*. This nuanced ordering of the letters is a difficult concept to describe in words, but a diagram makes the patterns immediately obvious.

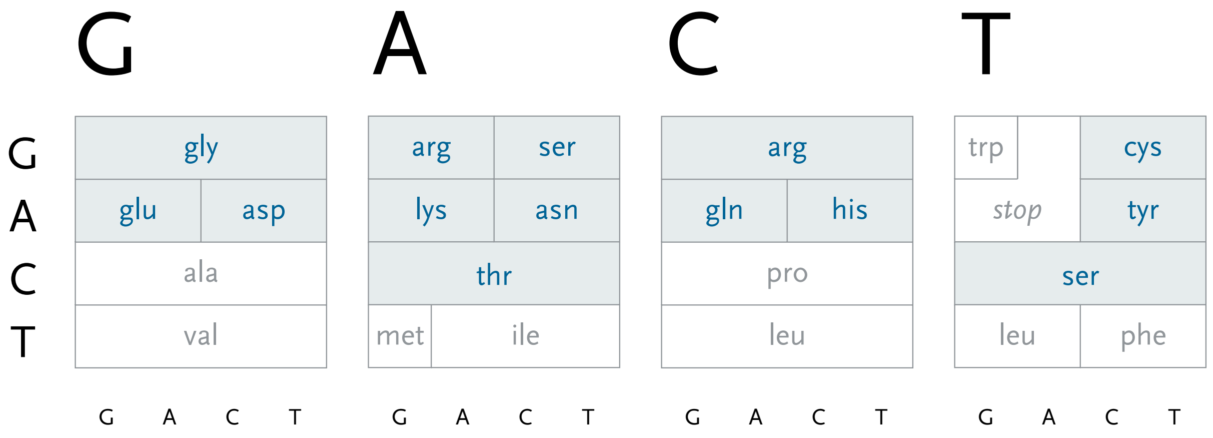
This new arrangement of the genetic code diagram is read counter-clockwise, following the letters from largest (at the top) to smallest (at the bottom). The size indicates which letter is to be read first, while also helping to place the most emphasis on the first two letters which have the greatest impact on the selection of the amino acid.

With that in mind, the design of the diagram should be focused on revealing the pattern of how the letters affect one another, because the pattern will be easier to discern (and remember) than simply memorizing the 64 possible mappings to the 20 amino acids. In the design below, the viewer selects the first letter from the top, which will restrict the remaining choices among one of the four blocks.



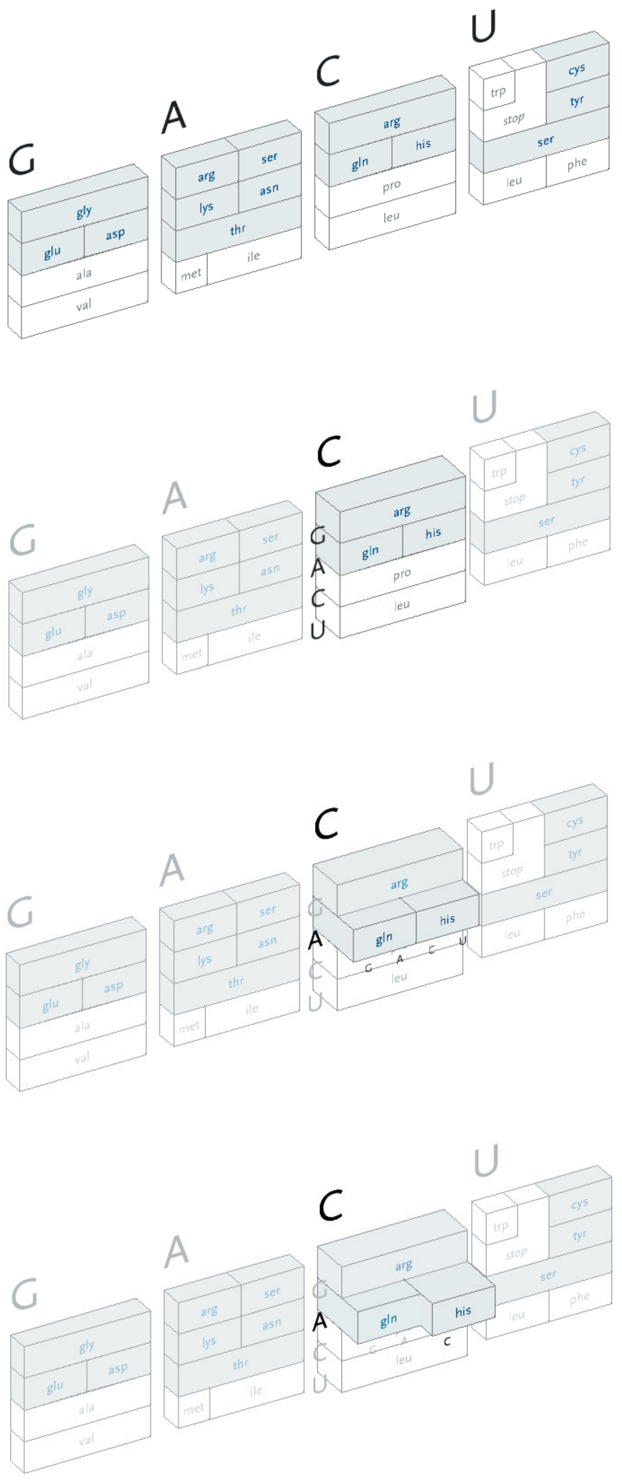
The second letter is chosen from the left-hand side, which limits the remaining choices to a specific row. The third letter is selected from the bottom, which determines the column within the row. However, as can be seen above, the row often extends across all four letters (a four-fold degenerate), meaning the third letter won't have an effect in determining the amino acid used. In other cases, only two possibilities exist (such pairs are called two-fold degenerates). This subtle rearrangement attempts to be simpler than common versions of this table, which tend to be cluttered with all choices for each letter or can be disorienting for a viewer because they require forward and backwards reading. Most important, it focuses on revealing what's interesting about the data—the weighted importance of each letter, and employs that to create a clear diagram. Few data sets are rigid and generic, there is almost always something interesting that can be pulled from them, so the role of the designer is to find that pattern and reveal it.

In addition, a small amount of color can be employed in the diagram to reveal the hydrophilic amino acids being grouped at the top, with the hydrophobic found primarily on the bottom. If not using color, the same could be achieved with a gray pattern or other visual device.



Adding the chemical structures of the amino acids would expose more patterns that would go even further to explain why amino acid translation works the way it does.

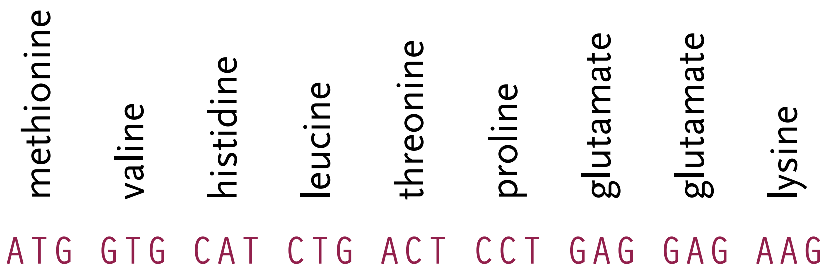
An interactive version of the diagram allows the user to select letters in succession, and observe the selections that remain. A three dimensional drawing is used to emphasize the selection.



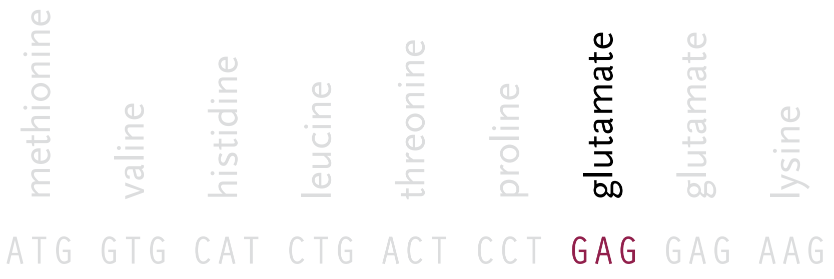
This version only shows the letters in question at each stage, extending the block of each sub-selection forward until the final amino acid is chosen. The starting position is at the left, the second image is just after *C* has been chosen as the first letter, followed by *A* and in the final frame, *C* is chosen for the third letter, bringing forward the amino acid histidine. The interactivity helps the user to learn how the relationships work through experimentation.

4.3 CHANGES TO CODING SEQUENCES

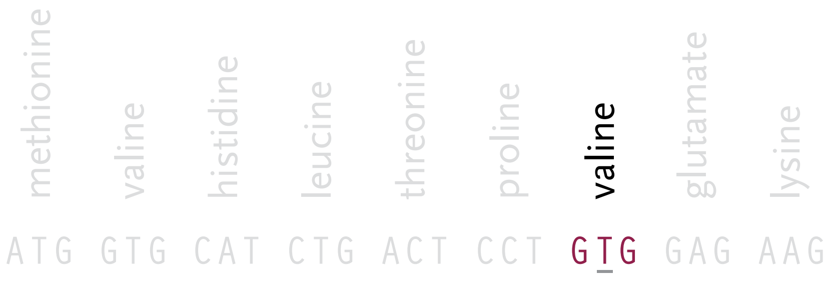
Changes can occur in the genetic sequence, either due to copy errors or other outside effects, resulting in a letter that changes which may in turn alter the amino acid produced. In the case of the HBB gene shown a few pages previous, the first few amino acids are as follows:



Sickle-cell disorder involves a change in the first glutamate in the sequence:



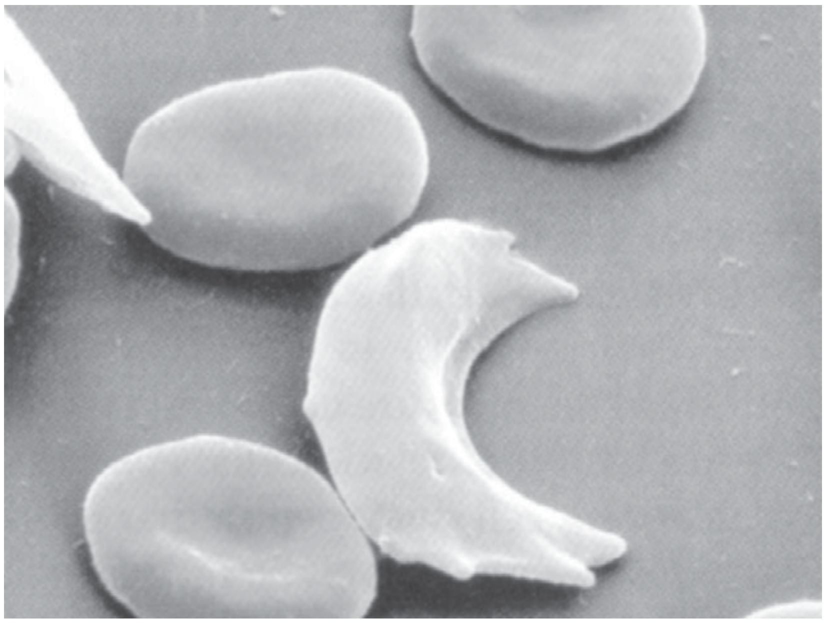
The second letter of the codon changes from an A to a T, which in turn produces the amino acid valine, instead of glutamate:



The change disrupts the amino acid chain by changing its shape, causing detrimental effects to the body:

> *When the mutated hemoglobin delivers oxygen to the tissues, the red blood cell collapses, resulting in a long, flat sickle-shaped cell. These cells clog blood flow, resulting in a variety of symptoms including pain, increased infections, lung blockage, kidney damage, delayed growth and anemia (low blood cell count).*
>
> GSLC.GENETICS.UTAH.EDU/UNITS/NEWBORN/INFOSHEETS/SICKLECELLDISORDER.CFM

The characteristic shape of sickle cell can be seen in this image, where the collapsed cell forms a "sickle" shape:

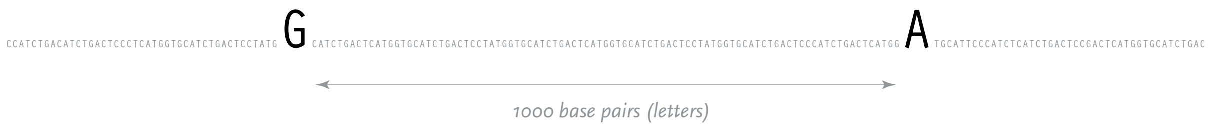


Original image source is unknown.

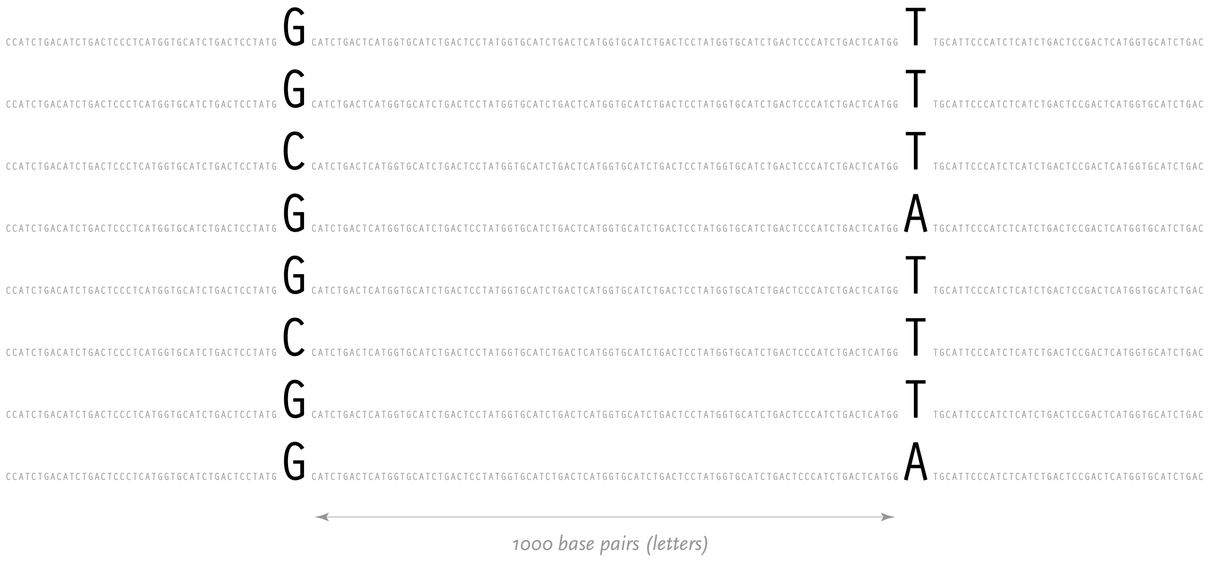
As noted in the amino acid table, not all changes will alter the character of the instruction set. Not all changes cause deleterious effects like susceptibility for a disease, they might simply modify something basic like one's eye or skin color.

4.4 TRACKING GENETIC VARIATION

While it's rare that a single letter change will be responsible for something as dramatic as a disorder like sickle cell, they are still a powerful means for understanding and tracking change in individuals and populations.

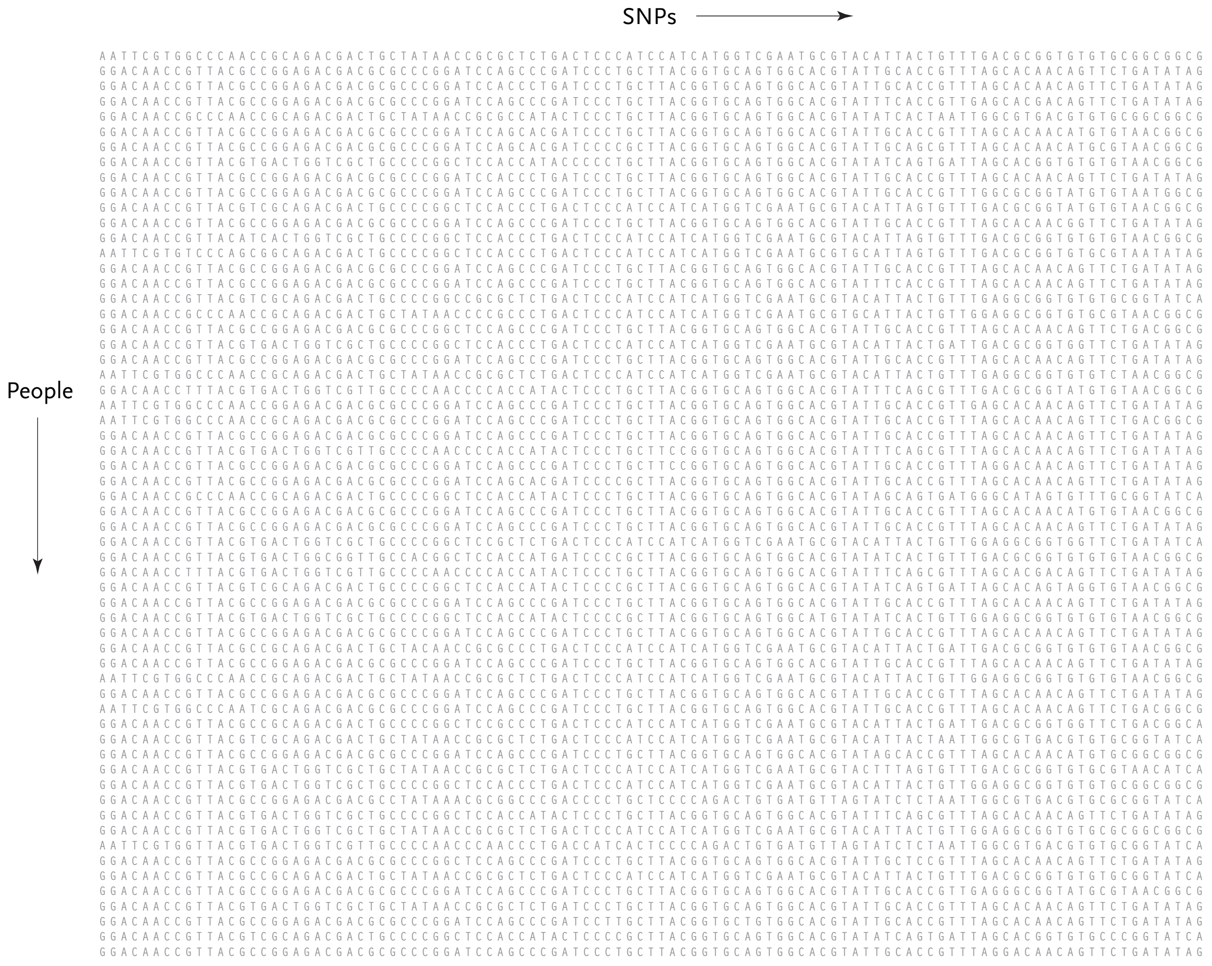


Such a change is called a SNP (pronounced "snip", and an acronym for *single nucleotide polymorphism*). When the genomes for two unrelated people are compared, a SNP can be observed roughly every thousand bases. It is possible to find other single-letter changes in the code that are simply random, but SNPs are unique in that they are seen in an identical position across multiple people. A SNP is a result of some random effect, such as an error during *meiosis*, where the genetic data of a cell is copied during the process of creating a second identical cell.



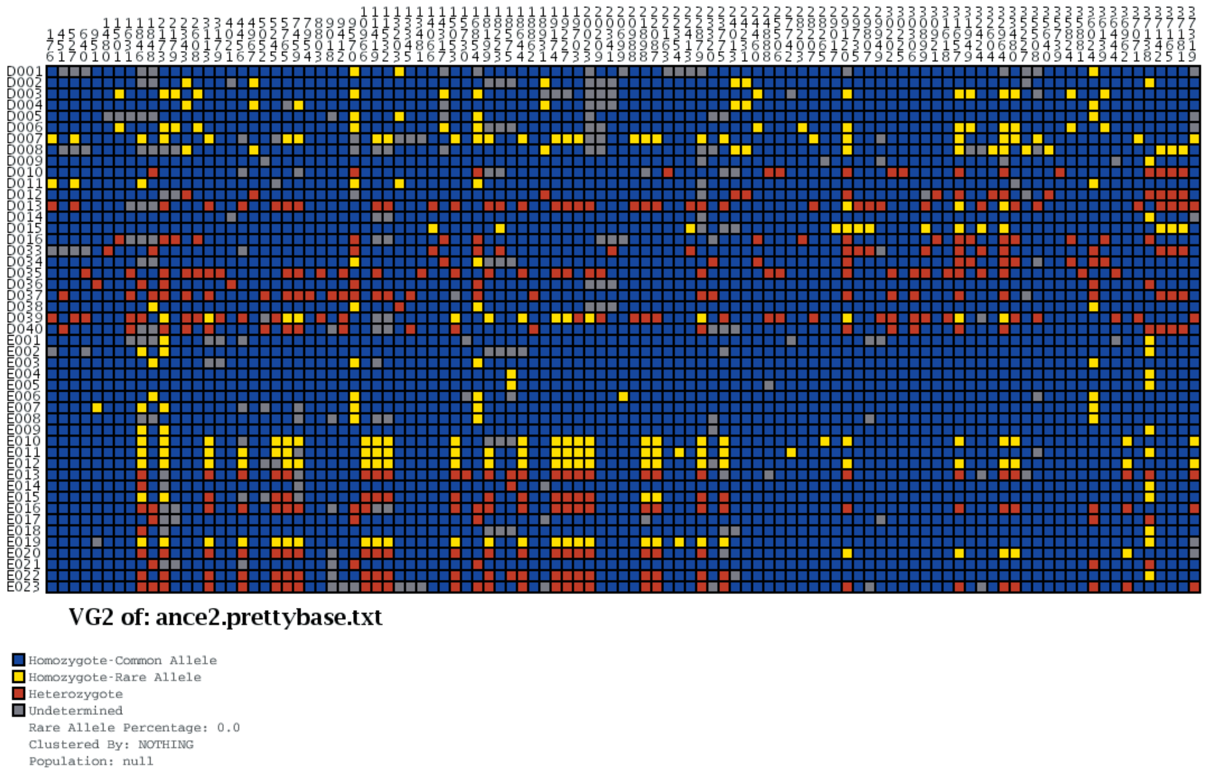
Each SNP is one of two letters (*variations*), and several projects have both taken place [*snp.cshl.org*] and are underway to produce a comprehensive catalog of all the variations that are found in the human genome [*www.hapmap.org*], along with a representative percentage of how often each of the two variations are found. It is believed that a full catalog of all such variations would show one every 300 letters.

Typically, this data will be studied with a few dozen or hundreds of individuals representing a *population*. A set of a few dozen to a hundred or more SNPs are observed across the group, forming a large table of the raw data:



Because every individual has two sets of chromosomes, the SNP on each chromosome may be different from the SNP in the identical position in the second chromosome. This situation is called *heterozygous*, as opposed to *homozygous*, where the same SNP is found on both chromosomes.

As a way of representing this raw data for populations, the Visual Genotype program has been developed by the Nickerson Lab at University of Washington. It colors each entry in the table of people versus SNPs based on whether it is heterozygous, homozygous for the most commonly occurring SNP, or homozygous for the rare variation of the SNP.

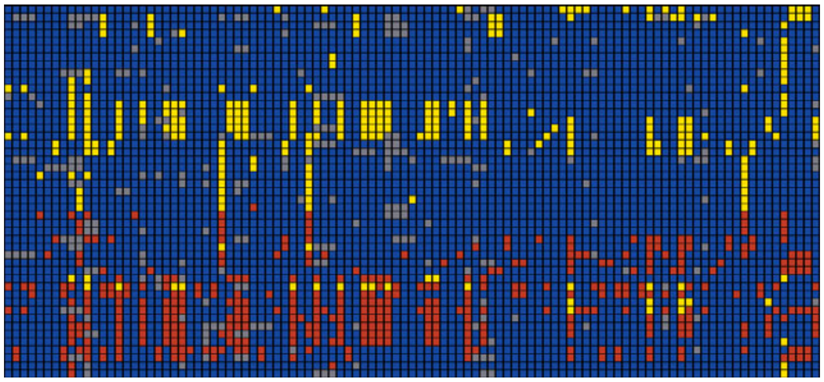


Above, the image output from Visual Genotype shows a series of data for the ACE2 Angiotensin I converting enzyme [Nickerson *et al*, 1998 and Rieder *et al*, 1999]. The image seems to convey a great deal of variation in the data set, yet makes no attempt to clarify it or highlight what kind of variations actually exist. When considered in terms of the Computational Information Design process, it's clear that the data is

acquire parse lter mine represent re ne interact

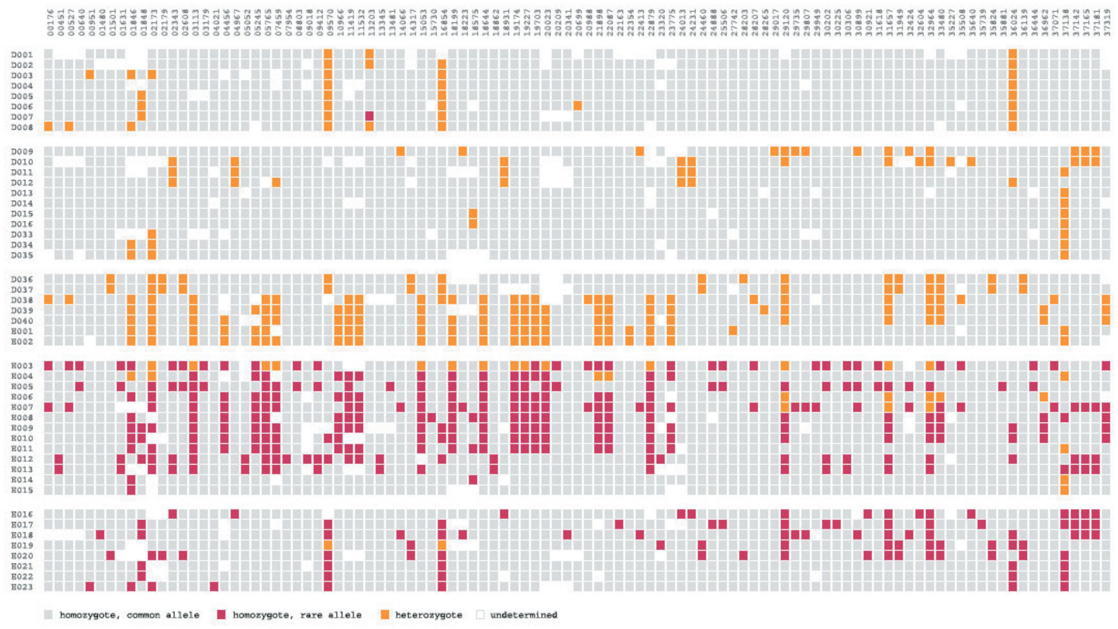
being acquired, parsed and represented in a very basic way, without regard for filtering or mining the data, much less given care of visual refinement or adding the use of interaction to improve how the data set can be explored.

To address these issues, a 'mining' step is added, where the data is clustered into useful groupings based on the rows that are most similar. This way the viewer can see both the rows that are similar, while it highlights the differences between those similar rows by placing them adjacent to one another, aiding the eye to understand what's inside the data set:



For an image to be useful, the relevant or key results should be immediately obvious, so that the viewer need not spend a great deal of time first trying to understand what might be important. Diagrams can themselves be complex, requiring some time of study before they're understood, but the basic idea being expressed needs to immediately apparent. It is similar to hearing someone speak in a foreign language, where even if it's not possible to understand the language (the specifics of the diagram) it's generally possible to tell whether they're angry or happy (what parts of the diagram are relevant and require further inspection). The original image fails such a test, partly due to the deep coloring of the image, creating an overall muddy appearance. Taking this into account, a new set of colors were chosen for the image seen on the next page.

In the previous version of the image, the dark blue boxes, which stood for 'homozygous common' are given a lighter gray color, to place them further in the background. The gray color that was previously used to denote 'missing' data is simply set to white, so that the data is simply missing from the diagram, rather than calling attention to itself. The two colors for the 'rare' genotypes are set to a pair of opposing but equal colors, and given greater importance than the background information in the diagram.



Small breaks are made in the table to further the sense of visual grouping between the 'clusters' of data rows. The goal of a diagram like this one is to help reveal patterns in a data set, so the representation should also emphasize those patterns. The groupings help the viewer break the large data into smaller pieces mentally, making it simpler to learn.

As a minor point, as compared to the original, the text at the top of the redesigned version has been oriented at a rotation, and the shorter numbers made to the same as the others so that they don't attract additional attention to themselves. In addition, as the user moves the mouse around the diagram, the row and column heading is highlighted for better clarity, and the associated data type is highlighted in the legend. This simple interaction helps associate each entry with its label.

The resulting product makes the case for Computational Information Design through a more complete depiction of the process used:

acquire   parse   lter   mine   represent   re ne   interact

mine → clustering similar data

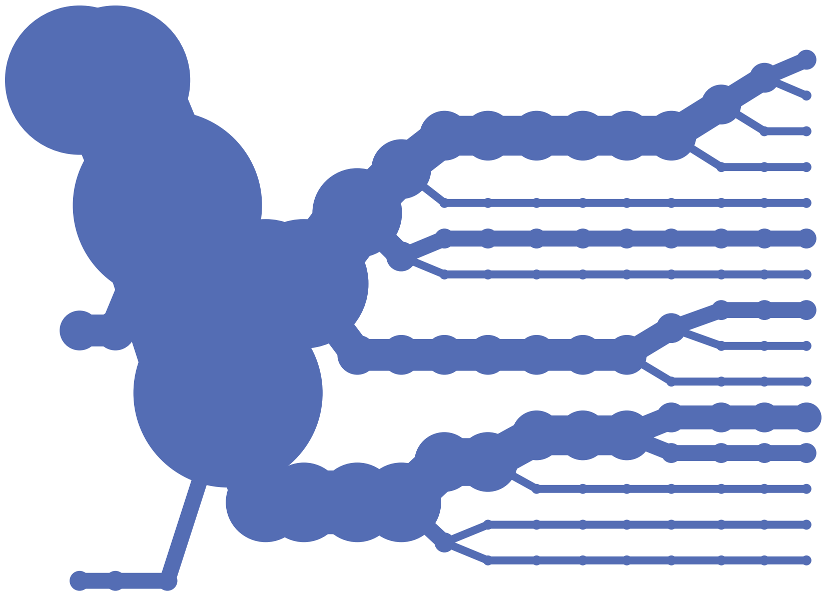
re ne → use of color to improve clarity

interact → mouse to highlight location

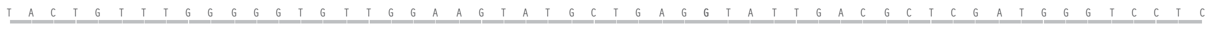
4.5 BIFURCATION PLOTS

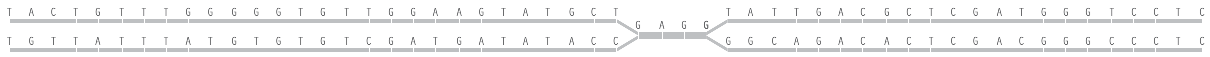
Another method of looking at variation data is to examine how the data differs between individuals, as it *bifurcates* from a center point of one or several SNPs.

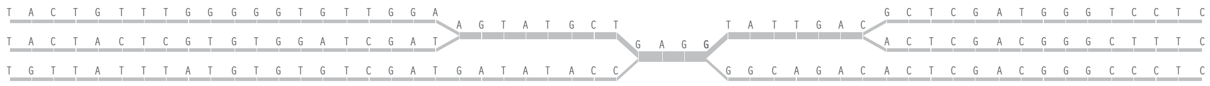
This diagram plots bifurcation and was introduced in [Sabeti *et al*, 2002] as a way to depict stretches of particular regions were conserved, often a sign of *natural selection*, meaning that the region is perhaps advantageous to the individual and is therefore more likely to survive and pass that set of variations on to its offspring (because of the region having impact on the survival rate of the organism).

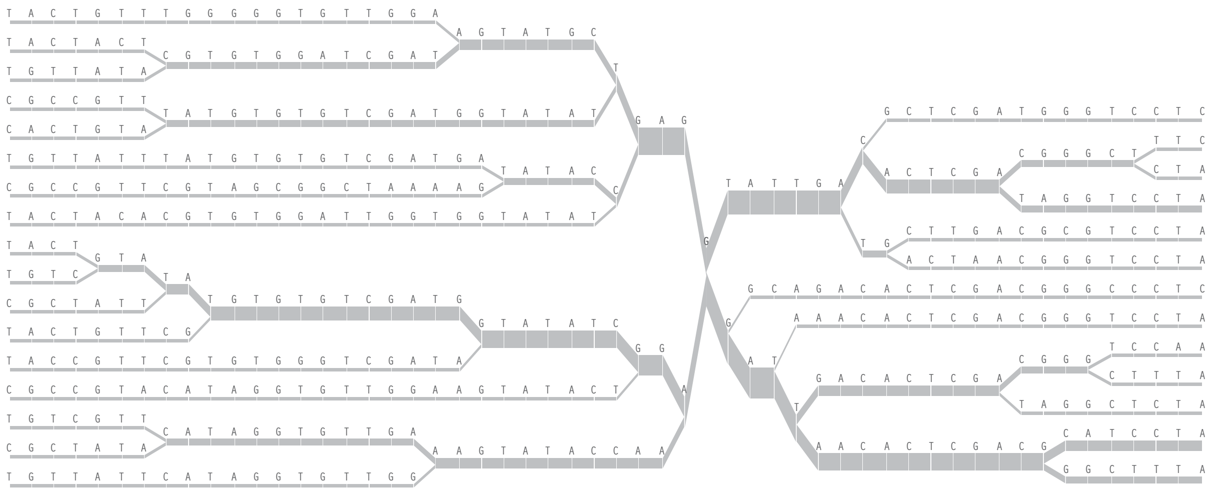


A redesign of this diagram seeks to clarify it further by eliminating the blobby effect that can obscure the exact data. In addition, interaction is used to allow the user to add or remove individuals from the image, so that it's easier to understand where the shifts happen and the origins of the shapes found in the diagram. The following sequence shows just one individual, then the separation as a second individual is added. Changes are shown as they happen from a SNP roughly in the center. This sequence shows one, two, three, then several individuals having been added:

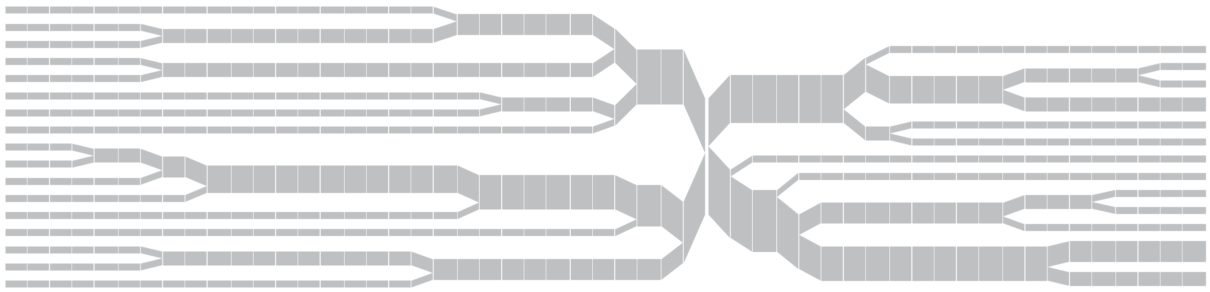

T A C T G T T T G G G G G T G T T G G A A G T A T G C T G A G G T A T T G A C G C T C G A T G G G T C C T C

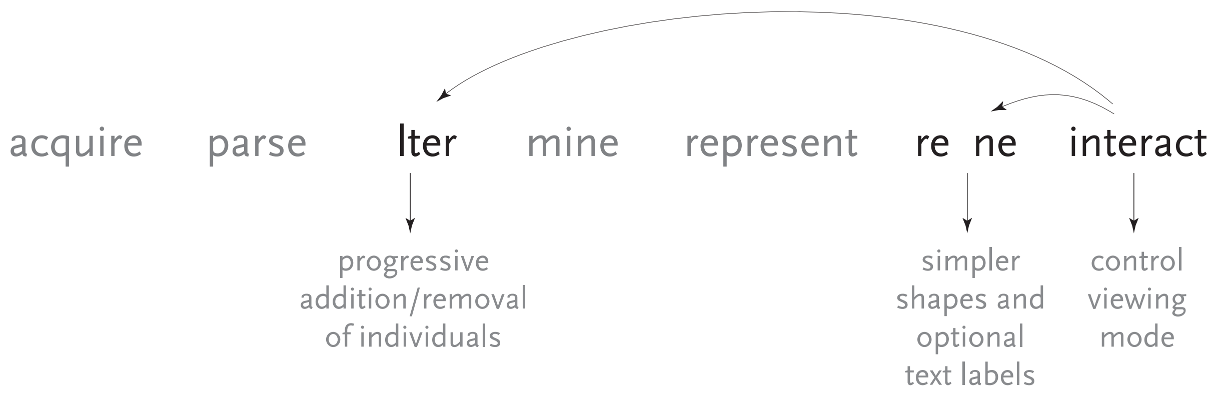

T A C T G T T T G G G G G T G T T G G A A G T A T G C T
G A G G
T A T T G A C G C T C G A T G G G T C C T C
T G T T A T T T A T G T G T G T C G A T G A T A T A C C
G G C A G A C A C T C G A C G G G C C C T C


T A C T G T T T G G G G G T G T T G G A
A G T A T G C T
G A G G
T A T T G A C
G C T C G A T G G G T C C T C
T A C T A C T C G T G T G G A T C G A T
A C T C G A C G G G C T T T C
T G T T A T T T A T G T G T G T C G A T G A T A T A C C
G G C A G A C A C T C G A C G G G C C C T C


T A C T G T T T G G G G G T G T T G G A
A G T A T G C
T A C T A C T
C G T G T G G A T C G A T
T G T T A T A
T
C G C C G T T
T A T G T G T G T C G A T G G T A T A T
C A C T G T A
G A G
T G T T A T T T A T G T G T G T C G A T G A
T A T A C
C G C C G T T C G T A G C G G C T A A A A G
C
T A C T A C A C G T G T G G A T T G G T G G T A T A T
T A C T
G T A
T G T C
T A
C G C T A T T
T G T G T G T C G A T G
T A C T G T T C G
G T A T A T C
T A C C G T T C G T G T G G G T C G A T A
G G
C G C C G T A C A T A G G T G T T G G A A G T A T A C T
A
T G T C G T T
C A T A G G T G T T G A
C G C T A T A
A A G T A T A C C A A
T G T T A T T C A T A G G T G T T G G
G
T A T T G A
G C T C G A T G G G T C C T C
C
T T C
C G G G C T
C T A
A C T C G A
T A G G T C C T A
C T T G A C G C G T C C T A
T G
A C T A A C G G G T C C T A
G C A G A C A C T C G A C G G G C C C T C
G
A A A C A C T C G A C G G G T C C T A
A T
T C C A A
C G G G
C T T T A
G A C A C T C G A
T
T A G G C T C T A
C A T C C T A
A A C A C T C G A C G
G G C T T T A

Once the diagram has reached many individuals, it might be appropriate to remove the text, so that a more compact, overall picture can be seen of the data:




acquire
parse
lter
mine
represent
re ne
interact
progressive addition/removal of individuals
simpler shapes and optional text labels
control viewing mode

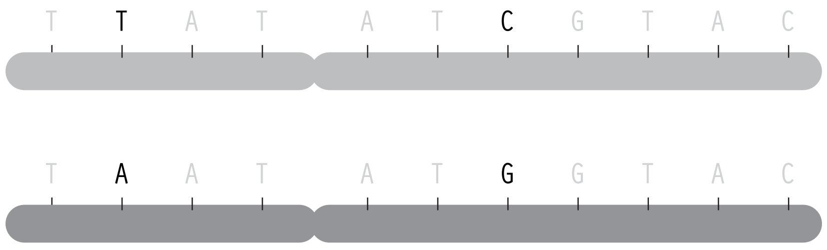
The process for the redesigned version is shown above, and highlights some of the inter-working between the how interaction can be used to affect other parts of the process.

By pressing keys in the program, the user can add or remove individuals, altering the filtering used for the data. The user can also control the viewing mode, offering a way to refine the representation dynamically, optionally disabling the text to avoid a cluttered appearance when a large amount of data is in use.

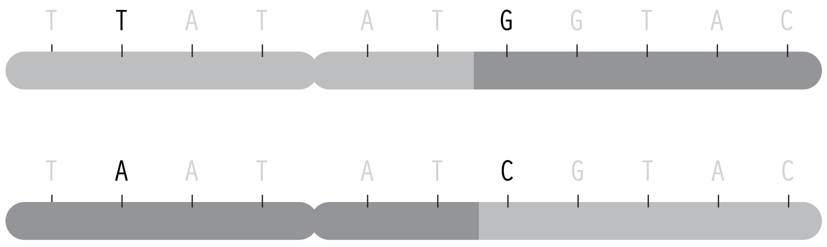
4.6 HAPLOTYPES

Many diseases (i.e. diabetes) are at least influenced by genetics, so by tracking differences in the genetic data of those who are and are not affected by a disease, it's possible to figure out what area of the genome might be influencing that disease. Simply put, if one, or a group of related SNPs (called a "haplotype") is more common in a group of people with the disease in question, then there is a likelihood that those changes contribute to the cause or associated effects of the disease.

SNPs are often found in groups, which are called *haplotypes*. The haplotypes are groups of SNPs that most commonly change together, because of how the chromosomes are inherited from one's parents.



The letters of genetic code are found along 23 pairs of chromosomes. Through conception, one set are passed along from a person's mother, the other half from the father. Each parent has two sets of chromosomes from their own parents. As the chromosomes are copied to be passed on, *crossover* occurs, where the beginning of one chromosome is linked to the end of another. The myriad of possible combinations drive the diversity that comes from genetic variation.



But in spite of the many variables involved, the predictable elements of this process mean that one can determine, with some degree of certainty, what traits came from one parent or the other through combinatorics and probability. This becomes useful for disease studies because given the letters read from several randomly placed SNPs from mother, father, and child, a researcher must be able to reconstruct the two chromosomes passed on to the child. The process of reading SNPs from each individual happens only in terms of position—it is not physically possible (with current methods) to separate one chromosome from the other and read them individually. Instead, the mathematical methods (called *phasing*) are used, because it is necessary to know which variation for the SNPs are found on each chromosome. The ordering is important in determining how closely related SNPs are to one another, so that a general structure (a set of haplotypes) can be determined from each chromosome.

The goal of the human genome project was to produce a consensus version of the 3.1 billion letters of code, though less focus was placed on understanding how the genome varies between people. With that structure in place, the penultimate project of population genetics is the *hapmap* project [*www.hapmap.org*], which is an attempt to catalog the variation data from several people across multiple populations. The hope is that this data set will provide insight into how the genome has evolved over time (is it random? or are we remarkably similar?), as well as patterns of inheritance for many types of diseases (why does this disease occur? why has it not disappeared through natural selection?). The hapmap catalog will include thirty child/mother/father sets (called *trios)*, across three distinct populations (Europeans, Africans, and Asians) that can be used as a reference set for answering such research questions.

The methods described here form the background for *population genetics* [Hartl & Clark, 1997] which is the study of how genetic data varies between groups of people and larger populations. And although this chapter will focus on SNPs, it is only one type of variation in the genome.

What follows are some of the methods used to visualize data from Haplotype and Linkage Disequilibrium testing. The techniques themselves have emerged out of necessity during the last few years, but have yet to be addressed with regards to the effectiveness of their representations in communicating the nature of the data they seek to describe.

4.6.1 Linkage Disequilibrium

A powerful tool for understanding variation in genetic data is *linkage disequilibrium (LD)* testing, where genetic data is scanned for statistically relevant patterns. Linkage is the connection between the occurrence of a feature, in this case SNPs. Disequilibrium simply refers to an upward or downward change that deviates from what might normally be expected if the SNPs were to appear at random (no correlation to one another).

One such method for testing LD is the D′ statistic, which originated from Leowontin's D statistic [Leowontin, 1964]. D is calculated given SNPs from two locations (or *loci*), multiplies the probability that both SNPs are observed together (P_{AB}), and subtracts the frequency of each (p_A and p_B) multiplied by one another. The function reads as:

$$D = P_{AB} - p_A p_B$$

For instance, to calculate D for the data set at right, one first counts the how often C appears in the first column, and divide by 20, the number of . The result will be p_A, the probability that C will appear. In this case, there are ten Cs, meaning:

$$p_A = 10 \;/\; 20 = 0.50$$

On the other hand, pb is the probability for C, the major allele, to appear in the second column:

$$p_B = 11 \;/\; 20 = 0.55$$

Finally, P_{AB} is the frequency with which the *major allele* appears in both columns, meaning the number of times C appears in the first and C also appears in the second:

$$P_{AB} = 9 \;/\; 20 = 0.45$$

So to calculate D for this data set, the result would be:

$$D = P_{AB} - p_A p_B$$

$$D = 0.45 - (0.50 \times 0.55)$$

$$D = 0.45 - 0.275$$

$$D = 0.175$$

Unsorted

C	C
A	G
A	C
A	G
C	C
A	G
A	G
C	C
A	C
A	G
C	C
A	G
C	C
C	C
A	G
A	G
C	C
C	C
C	G
C	C

Two locations from the first twenty individuals used in the data set in the next section.

Sorted
(for easier counting)

C	C
C	C
C	C
C	C
C	C
C	C
C	C
C	C
C	C
C	G
A	G
A	G
A	G
A	G
A	G
A	G
A	G
A	G
A	C
A	C

The D´ statistic is calculated by dividing D by its maximum value, which is calculated as:

$$\min(p_A p_b,\ p_a p_B) \text{ if } D > 0 \text{, or}$$

$$\max(p_A p_B,\ p_a p_b) \text{ if } D < 0$$

The value of p_a is the probability that the opposite SNP (A instead of C) appears in the first column. In the same manner, p_b will be the probability of G appearing in the second column:

$$p_a = 10 / 20 = 0.50$$

$$p_b = 9 / 20 = 0.45$$

Because D is greater than zero, $\min(p_A p_b,\ p_a p_B)$ will be used:

$$D' = 0.175 / \min(0.50 \times 0.45,\ 0.55 \times 0.50)$$

$$D' = 0.175 / \min(0.225,\ 0.275)$$

$$D' = 0.175 / 0.225$$

$$D' = 0.777\ldots$$

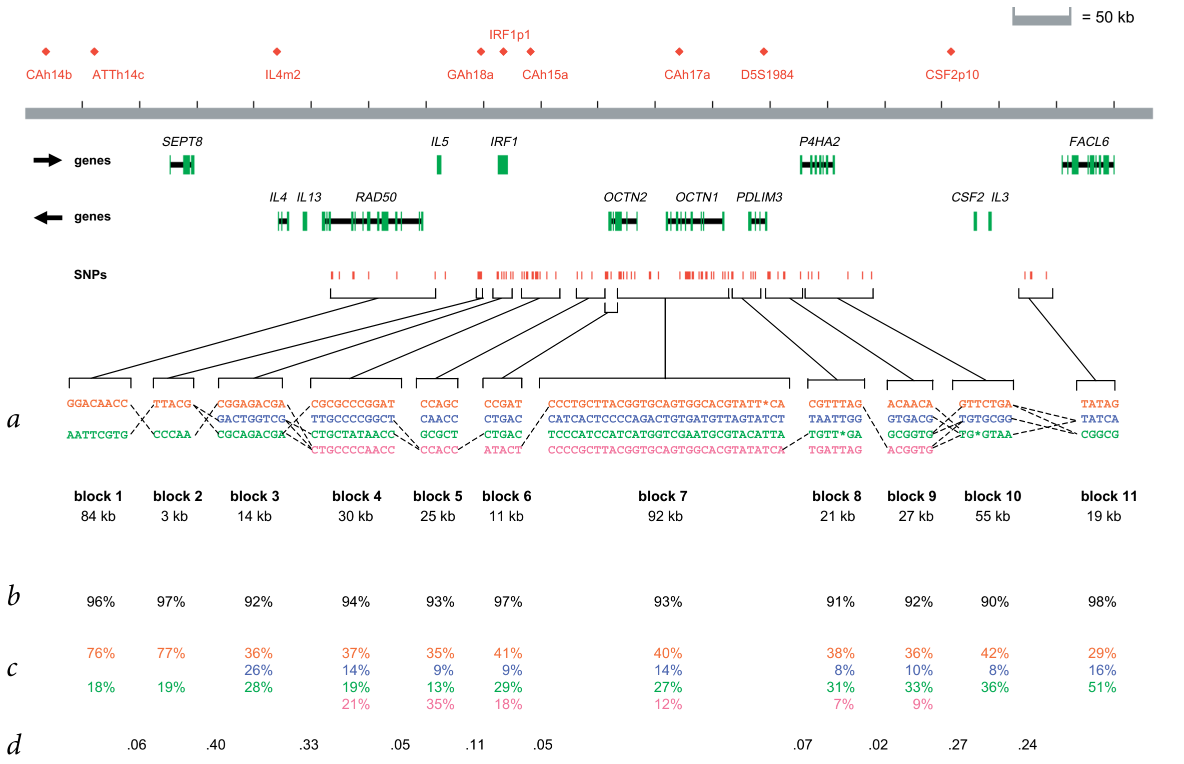
This value is not very high, and would fail the “block” definition found in the next section.

4.6.2 *Haplotype Blocks*

Studying populations tends to be difficult because of the amount of diversity found in the data. In the course of studying LD data from a group of individuals for connections to Crohn's disease [Daly *et al*, 2001 and Rioux *et al*, 2001], the data was found to be more structured than previously thought, showing a distinct pattern:

> *The results show a picture of discrete haplotype blocks (of tens to hundreds of kilobases), each with limited diversity punctuated by apparent sites of recombination.*
>
> DALY, 2001

The paper was included the following diagram (reproduced here in its entirety) to demonstrate how the block structure worked with the IBD5 data set studied for the Crohn's disease research:

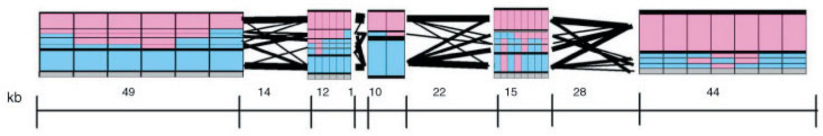


Block-like haplotype diversity at 5q31. ***a***, Common haplotype patterns in each block of low diversity. Dashed lines indicate locations where more than 2% of all chromosomes are observed to transition from one common haplotype to a different one. ***b***, Percentage of observed chromosomes that match one of the common patterns exactly. ***c***, Percentage of each of the common patterns among untransmitted chromosomes. ***d***, Rate of haplotype exchange between the blocks as estimated by the HMM. We excluded several markers at each end of the map as they provided evidence that the blocks did not continue but were not adequate to build a first or last block. In addition, four markers fell between blocks, which suggests that the recombinational clustering may not take place at a specific base-pair position, but rather in small regions.

The first column shows a block made of GGACAACC, and a second variation of exactly the opposite letters, AATTCGTG. By matching the red color of this text against the percentages shown lower, it can be seen that the first combination appears in 76% and 18% (respectively) of the individuals studied.

Simply put, the blocks are constructed by finding spans of markers that are in "strong" LD, meaning a series of snps where values of D´ are consistently above a threshold, in the original case, values above 0.8. Series of markers with D´ greater than 0.8 were added to a list, then sorted to find subgroups. This method, referred to as finding a "strong spine of LD" provided a means to find areas that where the signal was especially strong, but proved too rigid for minor fluctuations that might cause D´ to dip slightly lower.

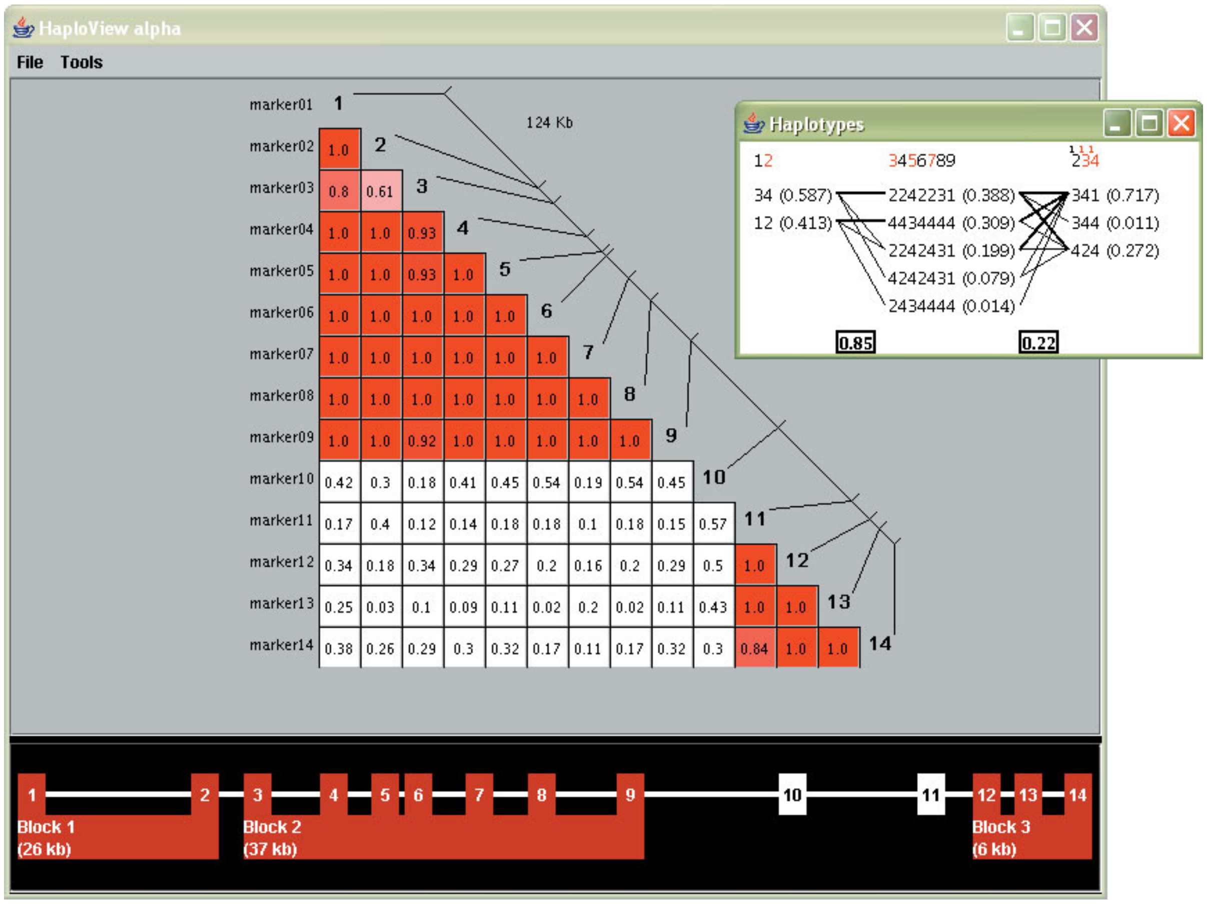
The block method was completed in greater detail for [Gabriel *et al*, 2002], which presented a slightly more refined version of the block algorithm developed by S. Schaffner. This took into additional parameters of a *confidence interval* and a *LOD* score. This definition uses a combination of the parameters to produce a more robust characterization of the block structure in a region.



In addition, a draft version of the Gabriel paper included another possible representation of block data (shown above), which used a series of rectangles that spanned along the block boundaries, where the height of each correlated to the percentage of individuals with that block structure. This presents an interesting starting point for a more qualitative representation of the data, where a glance provides the viewer with an immediate overall picture of the data set, without requiring additional study of percentages or other notation in a table.

4.6.3 HaploView

The study of haplotype blocks were the basis for the development of a software project from Daly's lab titled "HaploView," which can be used to examine haplotype data in a similar manner. A screen shot from its initial release appears below:



	1	2	3	4	5	6	7	8	9	10	11	12	13
marker01													
marker02	1.0												
marker03	0.8	0.61											
marker04	1.0	1.0	0.93										
marker05	1.0	1.0	0.93	1.0									
marker06	1.0	1.0	1.0	1.0	1.0								
marker07	1.0	1.0	1.0	1.0	1.0	1.0							
marker08	1.0	1.0	1.0	1.0	1.0	1.0	1.0						
marker09	1.0	1.0	0.92	1.0	1.0	1.0	1.0	1.0					
marker10	0.42	0.3	0.18	0.41	0.45	0.54	0.19	0.54	0.45				
marker11	0.17	0.4	0.12	0.14	0.18	0.18	0.1	0.18	0.15	0.57			
marker12	0.34	0.18	0.34	0.29	0.27	0.2	0.16	0.2	0.29	0.5	1.0		
marker13	0.25	0.03	0.1	0.09	0.11	0.02	0.2	0.02	0.11	0.43	1.0	1.0	
marker14	0.38	0.26	0.29	0.3	0.32	0.17	0.11	0.17	0.32	0.3	0.84	1.0	1.0

```
12            3456789            234
34 (0.587)    2242231 (0.388)    341 (0.717)
12 (0.413)    4434444 (0.309)    344 (0.011)
              2242431 (0.199)    424 (0.272)
              4242431 (0.079)
              2434444 (0.014)
         0.85               0.22
```

The large triangular plot shows the values for D′, each snp compared against the others. The inset diagram are the haplotype blocks, calculated via a menu option that runs one of the algorithms described in the previous section.

The interface was built more of necessity as a step towards an end-user tool—a means to combine several of the analysis tools that were in use for the two publications and other research and was beginning to influence other current projects.

As a be further enhanced through additional visual refinement that would place more focus on the data and improve its readability.

The coloring of the D´ plot provides a useful qualitative understanding of what's inside the data set. The reading of markers and their individual location is fairly clear (the diagonal line with others connected to it). The individual D´ readings are perhaps the most difficult to understand. Part of this is that such a small font size is required to fit three numbers and a decimal in a small space. This becomes especially problematic when dealing with large data sets, such as the IBD5 data, that includes more than a hundred markers instead of the fourteen shown in the sample image at the left.

The inset block diagram is useful for its simplicity, but could be clearer from an initial glance. Part of the problem is that numbers, mostly of identical size are used for many different kinds of features: percentages, weights, marker numbers, and the 1, 2, 3, 4 (representing A, C, G and T) for the individual genotypes themselves. A well designed diagram provides visual cues for ways to 'learn' the image. Applying such reasoning in this image means that numbers should either be used less or differentiate themselves better so that the intention is clearer.

The red colored numbers at the top are used to signify 'tag' SNPs, markers that can be used to uniquely identify the block, even in absence of the others. This feature is extremely useful because just that small number of such SNPs need to be typed, and the others guessed, with a significant degree of accuracy. A red coloring is poorly chosen as it more commonly represents an error or some other type of problem, as opposed to highlighting the most useful of the information.

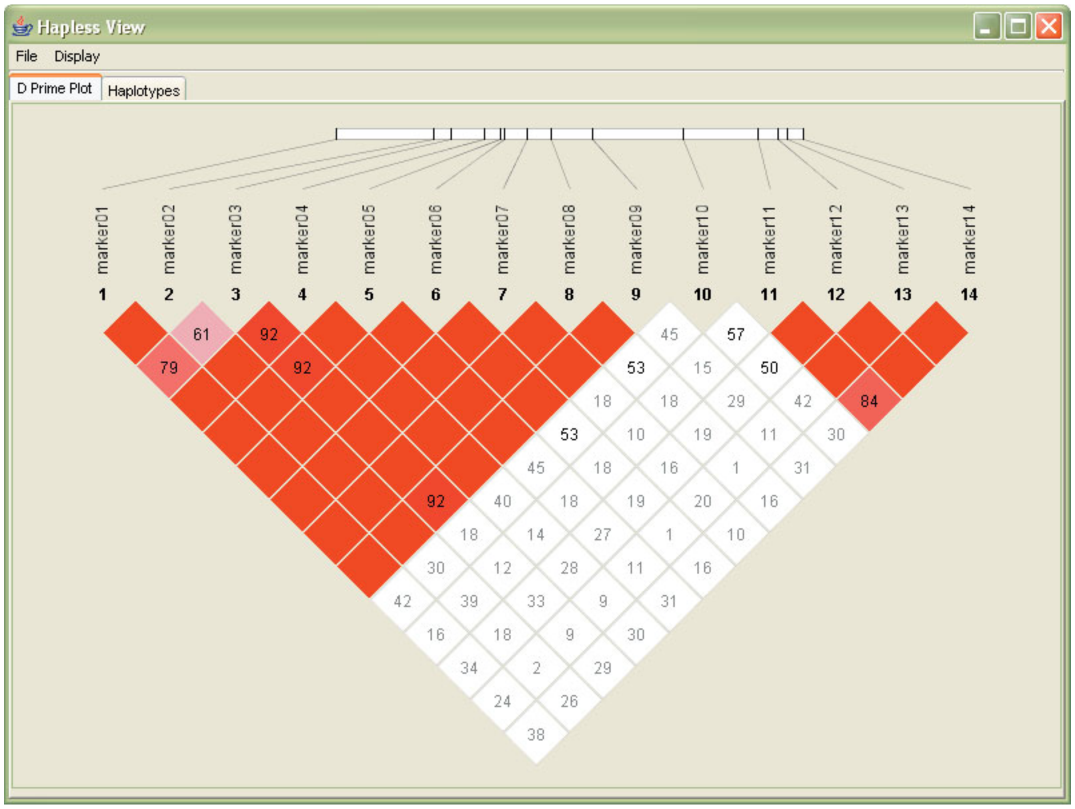
At the bottom of the screen is an additional device to both show the block ranges and manually modify them. This is disengaged from the other parts of the interface, making it confusing to understand what part of the data it relates to; particularly when all aspects of the data being shown have a strong relationship to one another.

4.6.4 *Redesign of HaploView*

The design of these diagrams was first developed manually to work out the details, but in interest of seeing them implemented, it was clear that HaploView needed to be modified directly in order to demonstrate the improvements in practice. Images of the redesigned version are seen on this page and the page following. The redesigned version was eventually used as the base for a subsequence 'version 2.0' of the program, which has since been released to the public and is distributed as one of the analysis tools for the HapMap [*www.hapmap.org*] project.

Since the values for D´ range from 0.00 to 1.00, in the following redesign they are multiplied by 100. Through a glance, it's easy to see where numbers are large and small (one, two, or three digits)

Next, values of 100 are not shown, reducing clutter in the image. Very little is lost by not showing the 100s (or 1.00 values) since they already



have the strongest coloring, and the absence of a number is sufficiently striking.

The plot is rotated 45°, to better align with the more familiar horizontal ordering of sequence data with which it is associated. The modified orientation is commonly employed in other uses of this diagram.

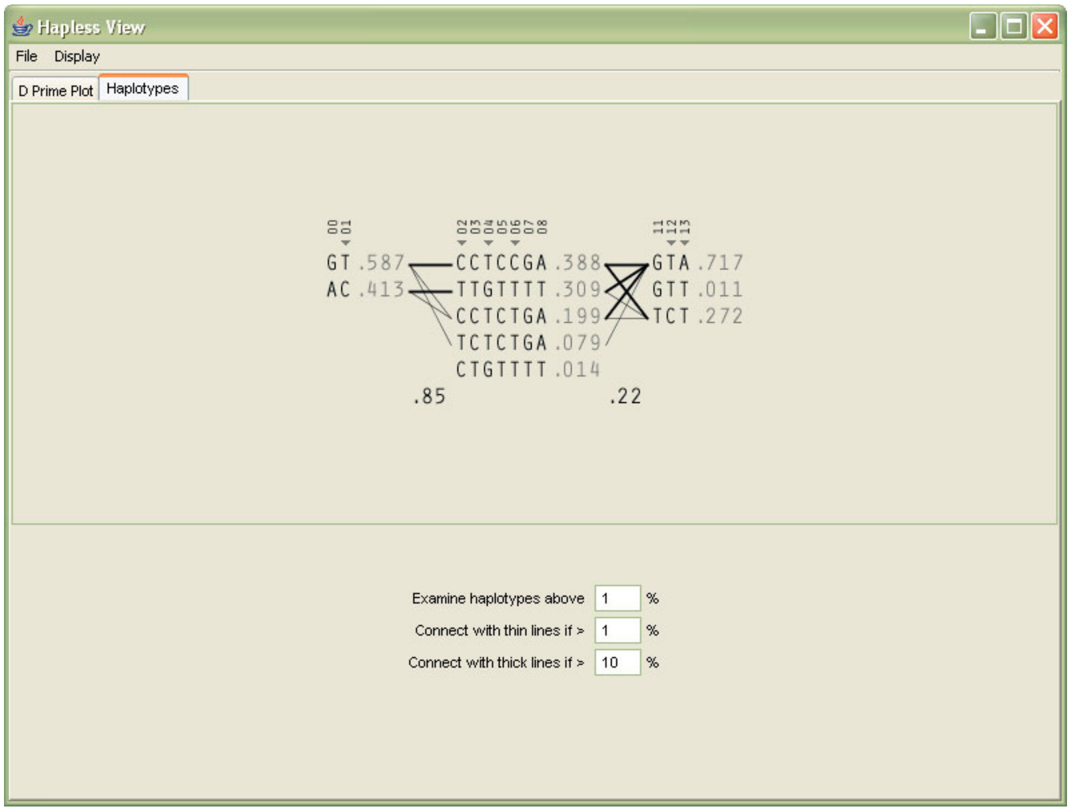
A small band is used to show the row of sequence data, which is slightly smaller than the long line used in the original image. With the markers slightly closer together on the line, it's easier to compare their relative positions.

The rotation also helps the placement of the marker number and titles to be less ambiguous. In the original image, the markers sit at the end of the row, and there is some confusion for an untrained viewer as to whether the markers are connected to either the row or the column.

The coloring in the redesign is toned down slightly, with a slightly deeper red color to make the colors slightly more 'natural' (more on this in the color subsection of the 'process' chapter). More consideration could be given to color in this image, where in some cases, slightly blueish colors are used for outlaying data points. Because the image shows not just D´ but is also mixed with the LOD score.

With the redesigned version, the image's size can be made much smaller than the previous image, while still retaining far more readability. While many of these features are small cosmetic fixes, these become increasingly important when considering a much larger data set. The contribution of the individual fixes would be magnified when considering a data set with several hundred markers.

In addition, the interface was modified to use a series of tabbed panels, to further convey a sense of multiple 'views' of the same data.



Examine haplotypes above 1 %

Connect with thin lines if > 1 %

Connect with thick lines if > 10 %

The second tabbed panel of the interface shows the block diagram.

In order to help the user differentiate between multiple types of data in the image (which were previously all numeric), the 1-4 numbering of the genotypes are replaced instead with the letters they represent, reducing the overall number count.

The percentages, since all are less that 1.0, have their decimal place removed, and are shown in a gray color because they are a secondary level of information in comparison to the genotypes.

The two percentages between the blocks (the *multi-allelic D´*) are shown in black so that it's clear they're different from the other percentages. The box around them is removed because it was unnecessary, and made the numbers more difficult to read.

The markers across the top are shown with a smaller font, and padded with zeroes so that a change from single to multiple digit marker numbers (from 9 to 10) doesn't distract by attracting attention to itself.

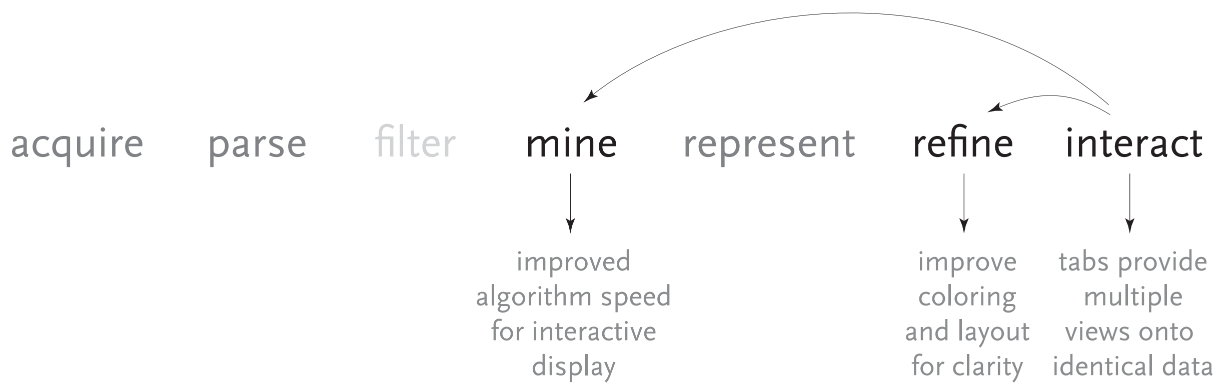
The tag SNPs in the original image were shown in red, and instead shown with a small triangle glyph that gives a better feeling of 'tagging', rather than the red coloring which might be used to imply a 'problem' with the SNPs.

Towards the bottom of the display, the user can modify the parameters for how the image is drawn, in order to change the criteria for what should be drawn as a thick line, or as a thin line, between the blocks themselves.

Relating the new software back to the original process, this redesign covers the mining, refinement, and interaction steps. The changes to the mining step were improvements to the algorithms used by the program for its calculations, to make them run more quickly for the interface that was now more interactive.

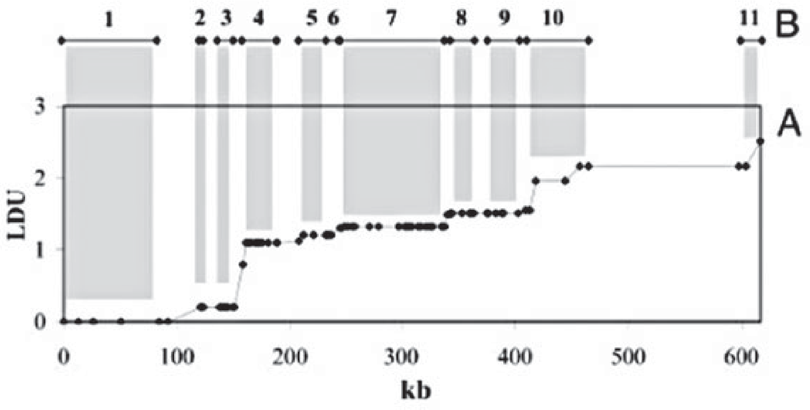
The interaction step involves re-working of the program to clarify how data is used. Through changes to the interface layout (and internal features like automatically loading additional 'support' data files), the number of menu options was decreased as well, in order to make the program easier to learn and therefore its feature set more clear.

The refinement step was discussed at length already, and results in a clearer, more compact way of looking at the data, making it easier to study larger data sets without the appearance becoming too overwhelming.

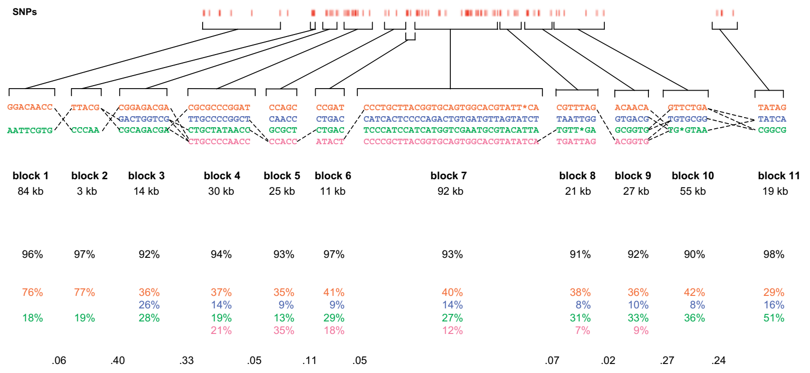

acquire
parse
filter
mine
represent
refine
interact
improved algorithm speed for interactive display
improve coloring and layout for clarity
tabs provide multiple views onto identical data

4.6.5 LD Units

Another means of analysis for identical data is a system of LD "units" proposed in [Zhang *et al*, 2002]. The system relies on similar mathematics as other tests of Linkage Disequilibrium, and provides an alternate perspective that would appear to support the "haplotype block" model described in section 4.6.2.



Shown above is a plot of LD Units versus distance along a chromosome (in kilobases, or thousands of letters of code) for the IBD5 data set used depicted by the diagram in section 4.6.2 and duplicated here:

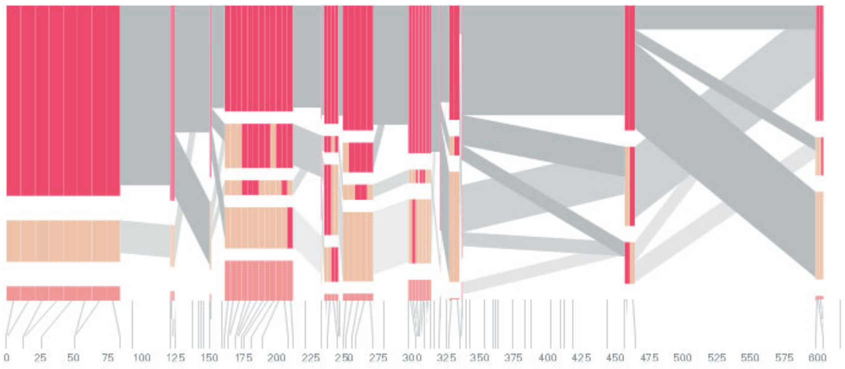


A rough correlation can be seen between the stair-stepping of the LDU plot versus the positions and lengths of the individual blocks. This is examined further in the next section.

4.7 *Combination*

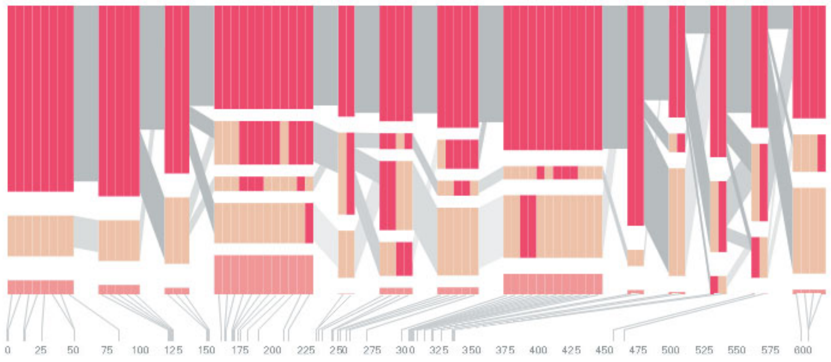
The goal for this project was to develop a single software piece that combines aspects of many of the preceding analysis and visualization tools for haplotype and LD data. By mixing the methods in an interactive application, one can compare how each of them relate to one another in a way that maintains context between each, and serves as a useful tool for comparison.

The diagram below depicts the block structure for a section of 5Q31 from [Daly, 2001] for 103 SNPs in a population of around 500 individuals. The colors in each row depict one of (only) two variations possible for each SNP, the most common in dark red, less common in a paler color. At the bottom of each column, a category for those variations occurring in less than 5% of the population. At a glance, this diagram can be used to quickly discern the general structure of the population in question, with roughly 70% of those studied exhibiting the haplotype block shown in the first column, and others that continue towards the right. Such an image is used in contrast to simply showing a chart with percentages, which requires the viewer to consider the relative importance of each percentage, rather than simply "seeing" it. Because size information can be processed pre-attentively (described in section 3.1), the mind processes the basic structure of the diagram before conscious thought is given to it.



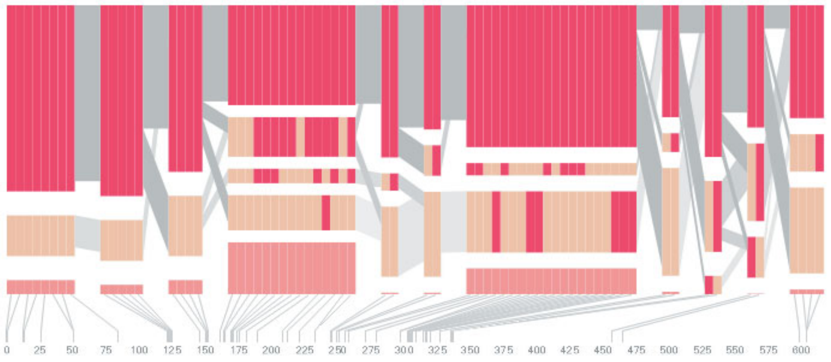
One difficulty, however is that the diagram above is shown with each SNP having a width equal to the distance to the next SNP, causing markers that are close together to be lost, and the frequency of transitions between each block (the gray bars) predominating, when they're only secondary information.

An alternative approach is to give each SNP (and each transition between the blocks) equal spacing, while maintaining a connection to the "real" scale using lines at the bottom of the image:



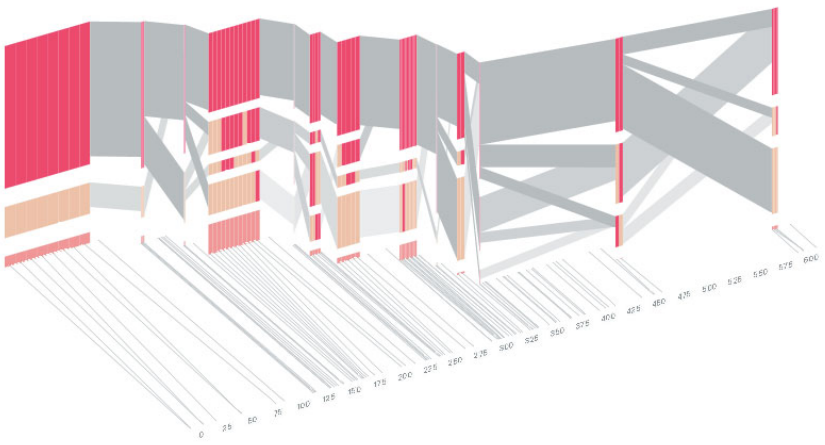
This version of the graphic can be more informative, and via an interactive software, this transition can be made dynamically, triggered by a single command, allowing the viewer to rapidly compare between the two without losing context.

Some find the block definition controversial [Couzin, 2002], so the use of an interactive software program that allows one to modify the parameters of the mathematics used to set boundaries on the blocks helps reinforce the notion that the blocks are themselves not meant as rigidly as might be implied by their name. For instance, an alteration to the parameters of the block definition produces the following:



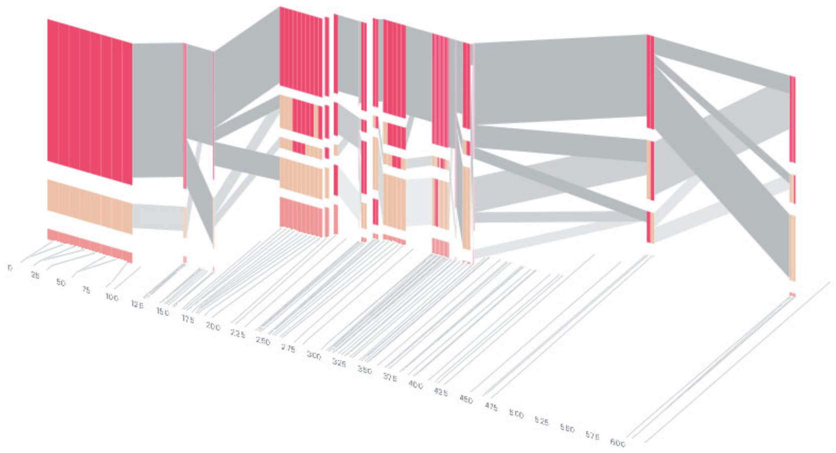
a view that shows wider block groupings. Moving the parameters in the opposite direction would produce a far more mixed picture than the original. This method of directly manipulating the values helps reinforce for the user how the algorithm itself works. The rapid feedback of simply manipulating the cutoff rate as an on-screen slider allows changes to be made in a way that is non-destructive, allowing the viewer to test different values but easily return to a previous state by a 'reset' function.

As another alternative, the block diagram can be shown in 3D, where each block offsets slightly in the z-axis, so that the lines depicting the transitions between blocks can be seen more clearly:

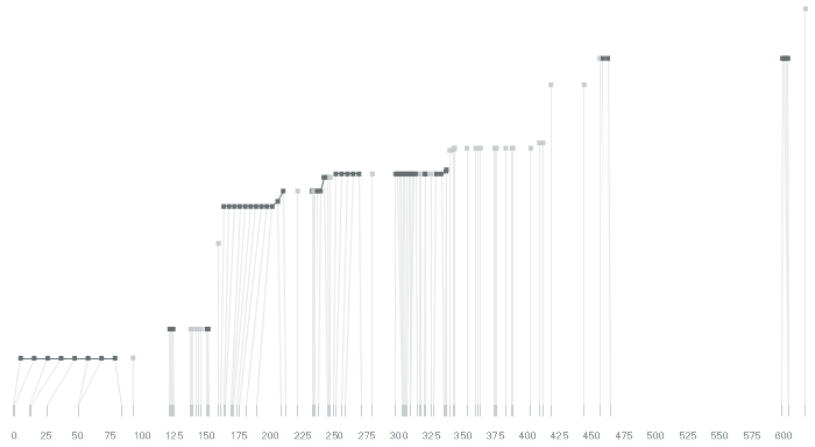


The view helps expose the transitions between blocks that are immediately adjacent one another. A "false" 3D isometric projection is employed that allows the data to be shown while preserving the linear scaling of the nucleotide scale in the horizontal axis.

However, it is likely placing too much emphasis on a few lost transitions to assign an additional spatial dimension to them. To make better use of the z-axis, the software can instead superimpose the LDU plot from the previous section, mixing the block diagram with an additional level of confirmation for the block structure. This works well because the stair stepping seen in the LDU map is reminiscent of the block structure shown above. When the user enables this mode, the software slowly moves each bar to its new position, so that the transition can be seen. An additional keypress moves back to the original layout so that the two views can quickly be compared.



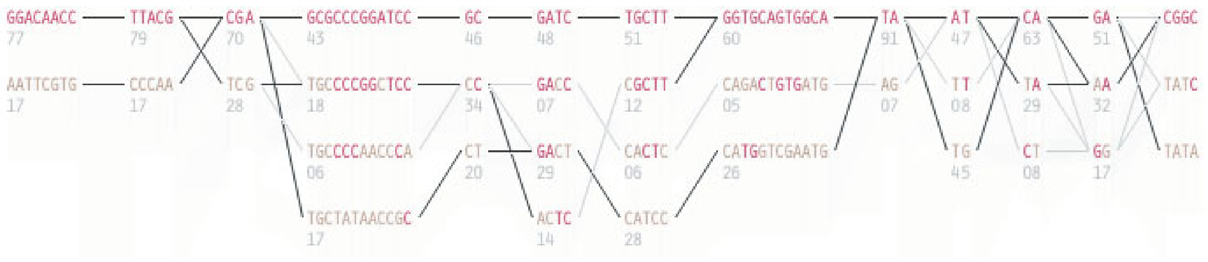
The user can easily transition between each type of view, enabling or disabling the three dimensional view if not needed, or modifying the block definition as appropriate, to see how it might match the LD map. Viewing the block image from the top will correspond exactly to the LDU plot, which can be seen by rotating the diagram in the software:



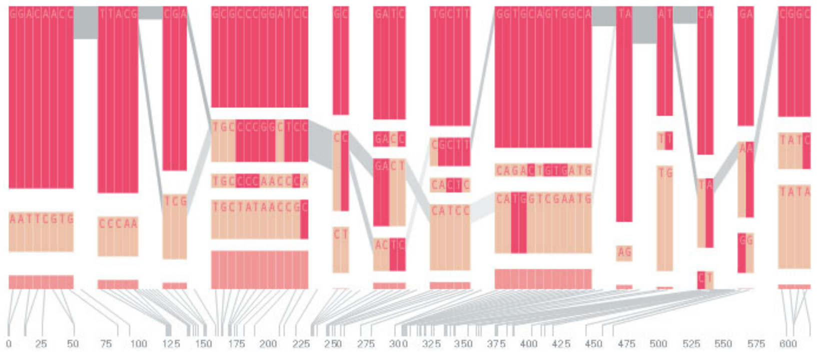
This type of exploration provides many different perspectives into the data, relying on the fact that users may have different goals in mind when observing the data, and personal preferences as to how they prefer to see the data represented.

To this end, an additional display can be added which simply shows the raw letters of the data and the percentage for each, the quantitative description to what's seen in the more qualitative visual block diagram:

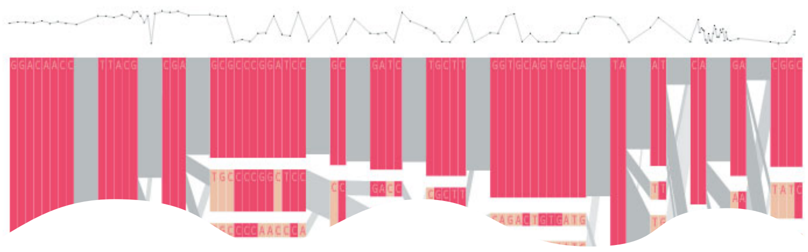
The software is built to morph between the representations, providing a tight coupling between the qualitative—useful for an initial impression and getting a “feel” for that data, with the quantitative—necessary for determining specific frequencies of haplotypes of interest for specific study.



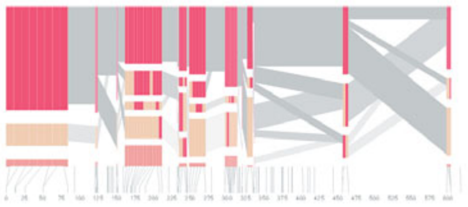
Additional perspectives on this data can be achieved through other user interaction, clicking on a particular block varies the line weights of the transitions to show how many of the transitions are related to the highlighted block, a dynamic version of the bifurcation plots that were discussed back in section 4.5.

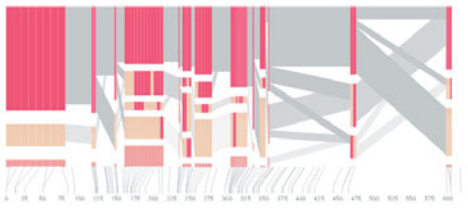


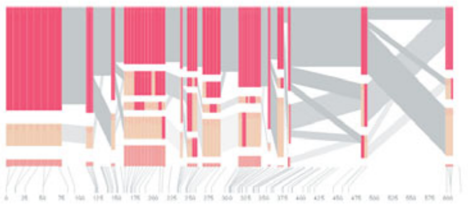
Clicking an individual SNP shows the degree of linkage (D´) relative to every other SNP as a line graph across the top:

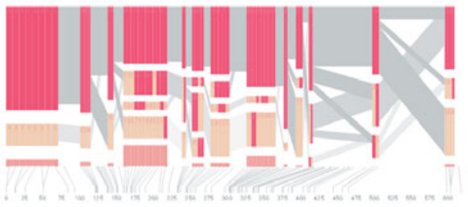


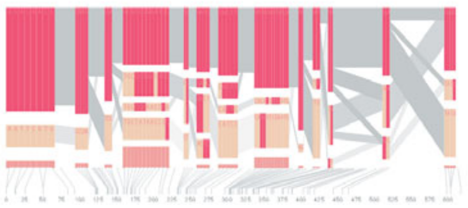
From actual SNP positions to evenly-spaced positions

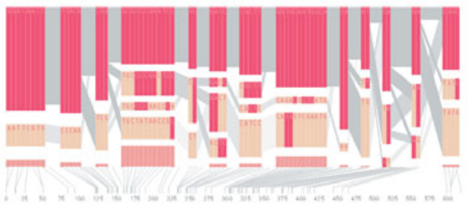




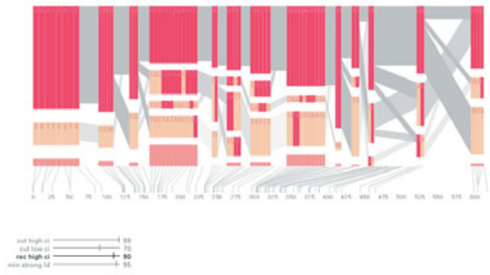


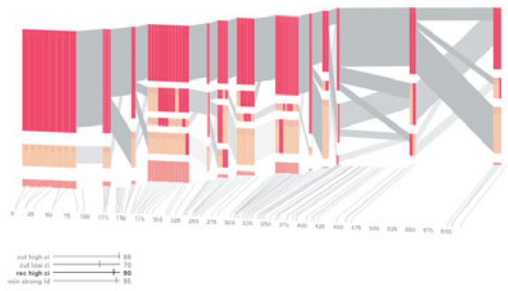


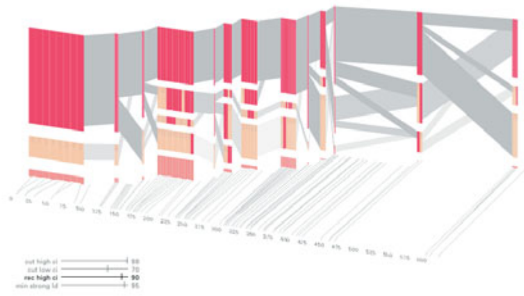


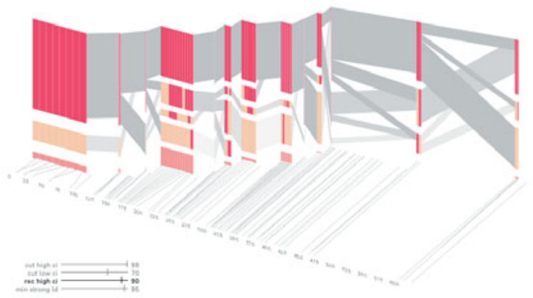


From 2D to 3D, to emphasize transitions betweeen blocks



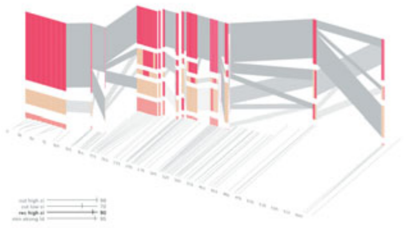


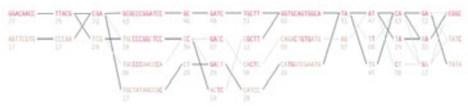


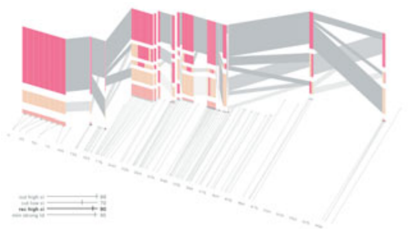


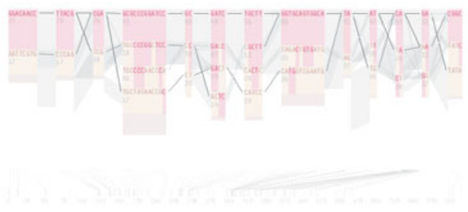
View from above reveals LDU plot in 2D

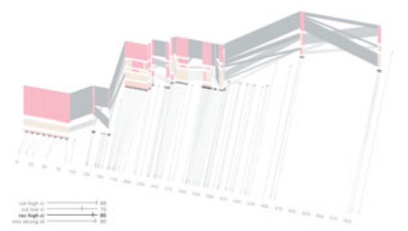
Quantitative view with exact values and the transition back

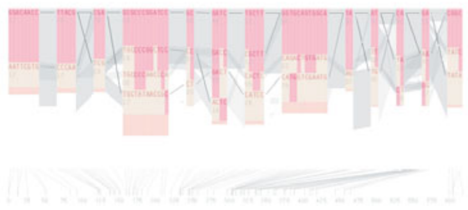


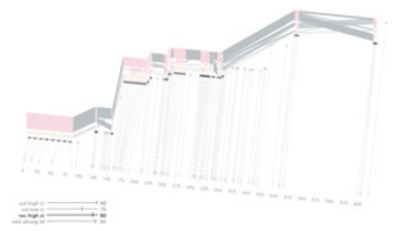


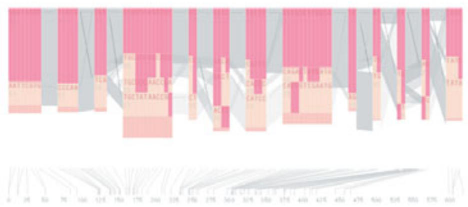


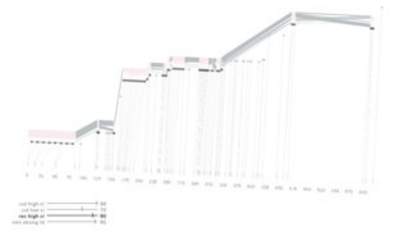


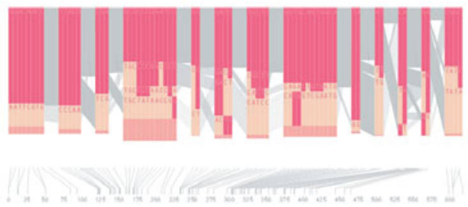


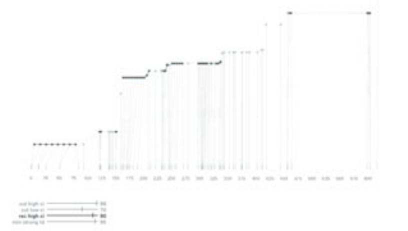


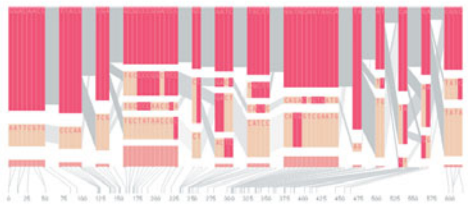










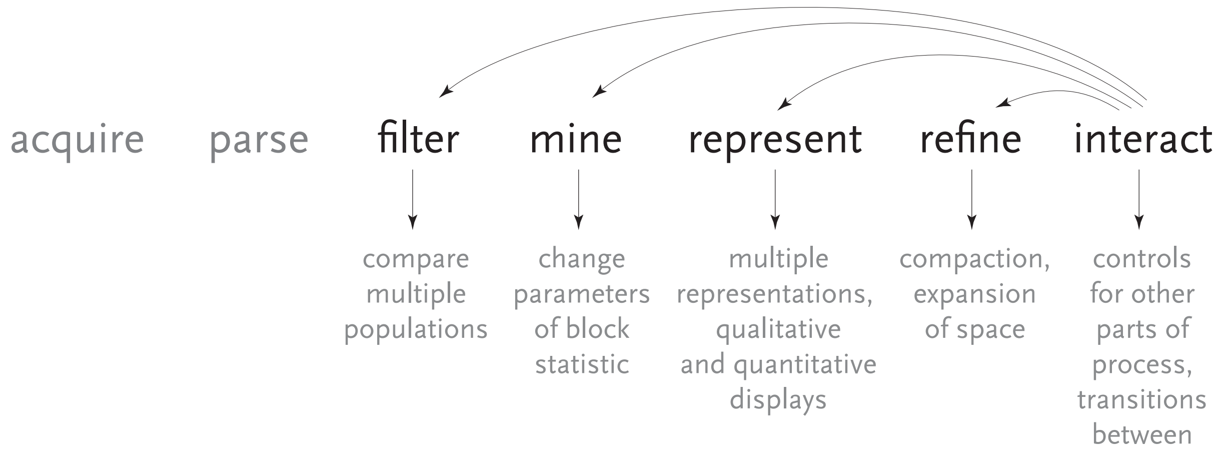


This provides a far more compact view than a full D´plot, which takes up far too much space for the amount of information that can usefully be extracted from it.

Another option allows the user to switch between populations (i.e. affected versus unaffected individuals), showing how the frequency of particular haplotypes increases or decreases.

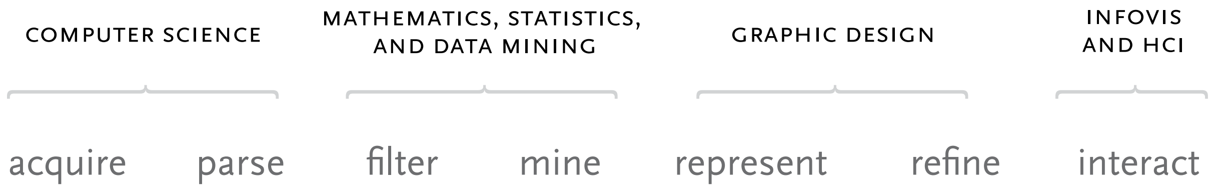
The many dimensions in genetic variation data necessitate multiple perspectives for how it is viewed, and an interactive software visualization provides a means to transition between these views in an informative manner, showing how the many views are related, yet at the same time highlight different aspects of the data set.

Relating back to the process diagram, this project shows a broader set of the steps in use. For interaction, the application makes considerable use of transitions between its multiple states. In terms of refinement, a number of means with which to expand and compress data in spatial terms are employed. The representation provides several modes of display. The mining and filtering steps allow different data sets to be read, or parameters of the algorithm to be modified, presenting the results of the modification to the user in real time.


acquire
parse
filter
mine
represent
refine
interact
compare multiple populations
change parameters of block statistic
multiple representations, qualitative and quantitative displays
compaction, expansion of space
controls for other parts of process, transitions between

5 Process

This chapter describes the process of Computational Information Design in greater detail. For each step, the goal is to provide an introductory (though hardly exhaustive) reference to the field behind it, as each step has its own considerable background. The intent of Computational Information Design is not to make a inter-disciplinary field, but rather that single individuals should be able to address the spectrum of issues presented—the goal is to understand data, so one needs to be familiar with the issues in handling data from start to finish.



Consolidation often occurs when an established set of fields with their own considerable background meet a technological shift. The combination of an affordable laser printer, scalable font technology, and page layout software gave rise to desktop publishing in the mid 1980s. This collapsed the role of the many practitioners required in publishing, making it possible that a single individual could (from their “desktop”) write, layout, print and distribute a printed artifact such as a flyer or a book. Previously such tasks were left to a writer to produce the text, a graphic designer to arrange it on a page, a person to set the type, another to photograph it for printing, produce negatives, then positives, make printing plates, and finally print the finished product on a printing press. The process was tedious and costly.

The combination of technology and tools that resulted in desktop publishing subsequently created an opening that led to an entirely separate process that focussed on the act of *publishing*, rather than on the individual tasks (layout, typesetting, printing) involved. It also filled a significant need by making information easier to disseminate and publish. Of course desktop publishing did not replace traditional publishing, rather it both brought publishing to a wider audience on the low end, and revolutionized the process on the high end, making it far more efficient and flexible.

For Computational Information Design, a history has been established (as set out in the third chapter of this thesis) amongst a range of fields, some more traditional like mathematics and statistics, others more recent like Information Visualization and Human-Computer Interaction. The range of work, by the VLW, and other research laboratories and institutions, point to a next step for the handling of data.

As a technological enabler, the availability of extremely powerful, and surprisingly inexpensive computer hardware with advanced graphic capabilities (driven by the consumer push for internet access and the gaming industry, respectively) puts the ability to build this work in the hands of a far wider audience.

As a further enabler, the *Processing* software development tool has been created as a step to bring development of visually-oriented software to a wider audience. While the requirement to write software is still considerable (and thus is not necessarily expected to lead to the sort of "revolution" characterized by desktop publishing), it provides an initial bridge that has shown early signs of success—thousands of developers and hundreds of projects being developed by the community surrounding it.

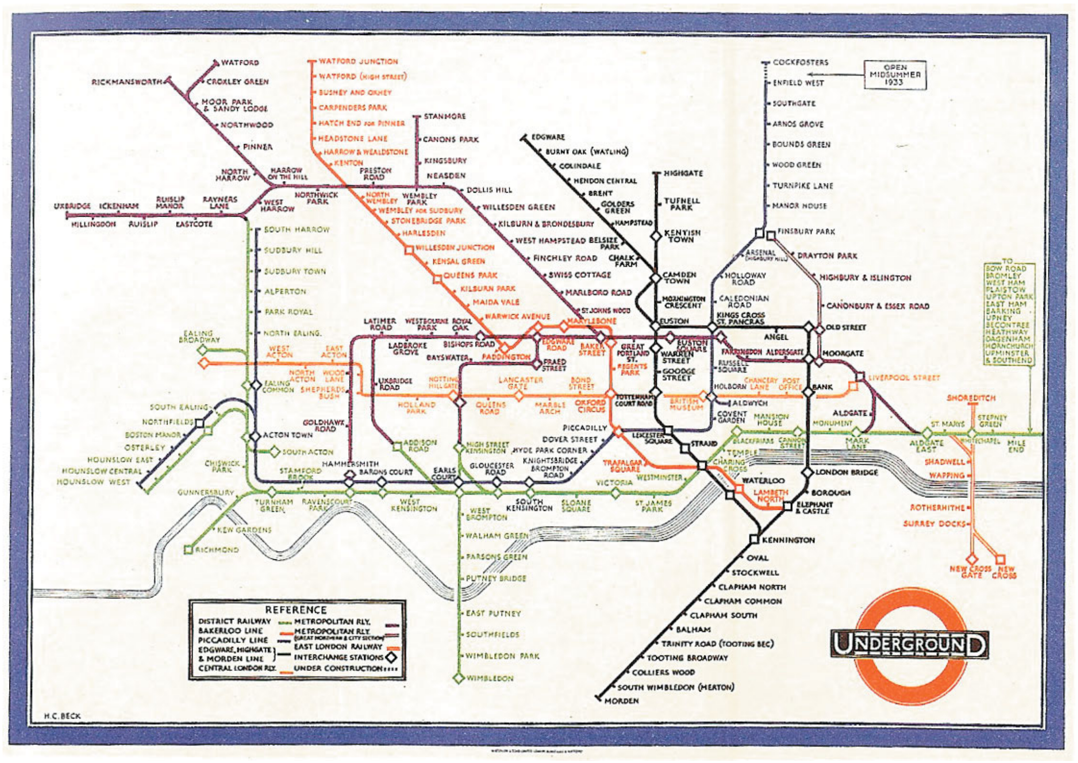
Not unique to this author, the process is also largely an attempt to codify what is being addressed by many dynamic information visualization projects, while addressing the gaps that exist in many of these implementations. Individuals and laboratories have addressed all or part of this process in these works, but as yet, little attempt has been made to teach it as a whole. This leaves the ability to build such works in the hands of a small number of practitioners who have figured it out on their own, while individuals in the larger design or computer science communities are often left perplexed at how to begin.

For these reasons, the goal of this thesis has been to demonstrate the use of such a process in basic (chapter two) and advanced (chapter four) settings, explain it for a wider audience (this chapter) and then make available a tool to make the process easier (chapter six).

5.1 WHAT IS THE QUESTION?

As machines have ever-increased the capacity with which we can create (through measurements and sampling) and store data, it becomes easier to disassociate the data from the reason for which it was originally collected. This leads to the all too often situation where visualization problems are approached from the standpoint of "there is so much data, how do we understand it?"

As a contrast, one might consider subway maps, which are abstracted from the complex shape of the city, and are instead focussed on the goal of the rider, to get from one place to the next. By limiting the detail of each shape, turn, and geographical formation, this complex data set is reduced to answering the question of the rider: "How do I get from point A to point B?"



The system of subway maps was invented by Harry Beck in the 1930s who redesigned the map of the London Underground [Garland, 1994]. Inspired by the layout of circuit boards, the map simplified the complicated tube system to a series of vertical, horizontal, and 45° diagonal lines. While preserving as much of the terrain as possible, the map shows only the connections between stations, as that is the only information usable for the rider in making the decision of their path.

In addressing data problems, the more specific the question can be made, the more specific and clear the visual result. When questions are scoped broadly, as in "Exploratory Data Analysis" tasks, the answers themselves will be broad and often geared towards those who are themselves versed in the data. But too many data problems are placed in this category simply because the data collected is overwhelming, even though the types of results being sought, or questions being answered, are themselves far more specific.

Often, less detail will actually convey *more* information, because the inclusion of overly-specific details cause the viewer to disregard the image because of its complexity. Consider a weather map, with curved bands of temperatures across the country. Rather than each of these bands having a detailed edge, their specificity is tied to conveying a broader pattern in data that is itself often subject to later be found inaccurate.

It is a somewhat jocular endeavor to boast of how many gigabytes, terabytes or petabytes of data one has collected and how difficult the analysis will be. But the question needs to be asked, why have they been collected? More data is not implicitly better, and often serves to simply confuse the situation. Just because it can be measured doesn't mean it should.

The same holds for the many "dimensions" that are found in data sets. Web site traffic statistics have many dimensions: ip address, date, time of day, page visited, previous page visited, result code, browser, machine type, and so on. While each of these might be examined in turn, they relate to distinct questions. Because the question might be "how many people visited page x over the last three months, and how has that changed month over month," only a few of the variables are required to answer that question, rather than a burdensome multi-dimensional space that maps the many points of information.

The focus should be on what is the smallest amount of data that can be collected, or represented in the final image, to convey something meaningful about the contents of the data set. Like a clear narrative structure in a novel or a well orated lecture, this type of abstraction would be something to boast about. A focus on the question helps define what that minimum requirements are.

5.2 ACQUIRE

The first step of the process is about how the data is first retrieved (where does it come from?) and the most basic aspects of how it is initially filtered. Some typical kinds of data sources:

analog signal

analog signal – an analog signal is simply a changing voltage, which is "digitized" by an analog-to-digital converter (ADC) into a series of discrete numbers. For instance, the waveforms that make up the audio on a compact disc are sampled into values that range from 0 to 65535, meaning that 65535 is the highest peak of the wave, and zero is the lowest. A value halfway between those two numbers is silence. There are 44,100 such levels for each second of audio, and the numbers are stored as binary on the disc.

The numbering of 0 to 65535 is because the data is stored as two bytes, or 16 bits that can be either zero or one. Sixteen bits can represented 2^16 numbers, which is 65536 total values (when counting from one, but 65535 because electronics count from zero).

file on a disk

file on a disk – this is perhaps the simplest type of data to acquire, since files exist only to be read and written. Files are either text (such as plain text documents or HTML files) and human readable, or binary (such as JPEG and GIF image files) and generally only machine readable. Making sense of such data is described in the section that follows.

stream from network

stream from a network – it is common for live data to be "streamed" directly from a network server. Common examples are the raw data from news feeds (generally text data) or live video and audio. The type of acquisition shifts because it is often unclear how much data is available, or it represents a continuously updating set of values, rather than something more fixed like a file on a disk.

relational database

relational database – A relational database is simply a large number of 'tables', which are themselves just rows of data with column headings. For instance, a table for addresses might have columns for first and last name, street, city, state, and zip. The database is accessed via a "driver" that handles the "stream from a network" style data. The database is first given a *query*, typically a command in a language called SQL, (short for *Structured Query Language*) for example:

```
SELECT * FROM addresses WHERE firstname IS joe
```

This query would grab a list of all the rows in the “addresses” table where “Joe” was found in the column named “firstname.” Despite their important sounding name, at a fundamental level, databases are extraordinarily simple. The role of a database is to make queries like this one, or others that can easily get far more complicated run exceptionally fast when run on enormous tables with many, many rows of data (i.e. a list of 10 million customers).

The pitfall of databases in interactive environments is that it’s not always easy to have data from them update automatically. Because they are fundamentally a means to “pull” data (rather than having it pushed continuously, like a network stream) software that uses a database will have to hit the database with a new query each time more or a different set of data is needed, which can easily cause a lag in an interface (several steps later in the process) or be prohibitively cumbersome for the system (i.e. thousands of people using an interactive application that’s connected to a database and running a new query each time the mouse is dragged to change values in a scroll bar).

an entire field – The data acquisition quickly becomes its own field, regarding how information can be obtained and gleaned from a wide variety of sources, before they’re even codified into a manageable digital format. This is a frequent problem concerning how relevant data is recorded and used. For example, how does one quantify the ‘data’ from an hour-long meeting, that involved a verbal discussion, drawings on a whiteboard, and note-taking done by individual participants?

...an entire problem area

In terms of Computational Information Design, the acquisition step is perhaps the most straightforward, however it’s important to understand how sources of data work (such as the distinction between text and binary data) , and their limitations (how common relational database can and cannot be used).

5.3 PARSE

This step looks at converting a raw stream of data into useful portions of content. The data might first be pre-filtered, and is later parsed into structures usable by a program. Almost always, data boils down to just lists (one dimensional sets), matrices (two dimensional tables, like a spreadsheet), or graphs (sometimes trees, or individual 'nodes' of data and sets of 'edges' that describe connections between them). Strictly speaking, a matrix can be used to represent a list, or graphs can represent the previous two, but the over-abstraction isn't useful.

5.3.1 *Pre-filter*

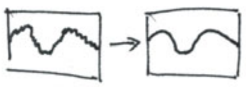
Pre-filtering are the most basic operations on the binary data of the previous step. It is not the more complicated content-oriented filtering that comes from in the "mine" section of the next step of the process, but rather involves issues of synchronization to find the content in a data set.

offset

offset – finding appropriate offset and subset of data in a binary stream. Often, a stream of data will have a "synchronization" signal that signifies the beginning of a new set of data.

For instance, a television signal is a one-dimensional stream of information, an analog wave whose intensities determine the intensity of each pixel on the television screen from left to right, top to bottom. For each line of the image, the signal has a specific up/down change that determines the end of one line of the image and the beginning of the next. Another signal marks the beginning of a new screen of data. When first turned on, a television receiver will wait until the first time it sees the signal's wave make the pattern for a new screen, and then begin reading and drawing to the screen from that point.

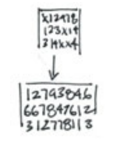
An offset might also be used in a similar way for a binary stream, so that some initial data is ignored, perhaps until a flag or marker is seen that quantifies a beginning point.



lter

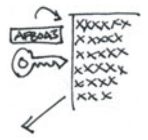
filter – a "low-pass" filter might be placed on the waveform to steady a bad signal, removing small deviations from the signal ("high frequency" data, passing only the "low") to clean it.

unpack/decompress – unpacking involves expanding data that may have been compressed with regards to its information content. For instance, it's more compact to store an image on disk by saying that there are a stream of 500 pixels of red, than to write out "red" 500 times. This is the method of simple compression algorithms like RLE, which stands for run-length encoding.


unpack/decompress

Others, such as *gzip*, pack data using slightly more complicated methods [*www.gzip.org*]. Unlike gzip which can act on a stream of data as it is emitted, table-based compression algorithms like LZW, (used for GIF images) first create a table of features in the data set that appear most often, rank them, and then assign the smallest possible amount of bits to the most common entries in the table.

decrypt – decryption involves using a mathematical key as a way to decode a data set, usually a very large number multiplied or divided by another.


decrypt

Perhaps the simplest form of encryption/decryption is ROT13 [demonstration at *www.rot13.com*], which simply rotates letters by 13 places (i.e. A becomes N, B becomes O, ... , and so on until N also becomes A).

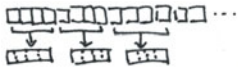
More feasible forms of encryption include SSL, widely used on the web to keep credit card information safe when making online purchases [*www.netscape.com/eng/ssl3*]. It relies on large number 'keys' (arbitrated between the customer and merchant's computers) to encode data.

5.3.2 *Parsing Tasks*

The parsing step involves breaking the data into known structures that can be manipulated by software. The result of this step is a set of data structures in memory. For this step in the zipdecode example in chapter two, this distilled the 45,000 entries in the zip code table into individual "records" stored in memory.

5.3.2.1 Bits with periodicity

This category is for situations where bits are broken into a linear stream at periodic intervals.


binary data

binary data – list of binary values that encode numbers. For instance, a temperature sensor connected to a computer might send a byte for its current reading each time it changes.

On the other hand, this could be a file filled with four byte numbers that encode a series of readings from a more sensitive device. Each byte is made up of a series of bits representing 2^7 down to 2^0, so for the byte 10110001, that looks like:

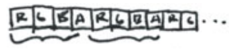
1	0	1	1	0	0	0	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

The second row shows the power of 2 for each bit, and below that is the value for that power of two. Each bit set to one has its value added, so this byte represents:

$$2^7 + 2^5 + 2^4 + 2^0 = 128 + 32 + 16 + 1 = 177$$

Because a byte can only signify a number from 0..255, several bytes can be used together through *bit shifting*. Two bytes used together can represent 2^{16} possible values (0..65535). This is done by making the first byte signify 2^{15} down to 2^8, and then adding that number to the second byte, which represents 2^7 down to 2^0. More bytes can be added in the same manner, on current machines, four bytes will be used to represent an integer number. In addition, the left most bit is sometimes used to signify the 'sign' of the number, so if the bit in the 2^7 place is set, the number is negative. This will provide numbers from -128 to 127.

interleaved binary data – this is often used for image data, where a series of bytes will depict the red, green, and blue values for a pixel. The first byte might be a value from 0..255 for red, followed by one for green, and another for blue. Images are commonly stored in memory as 32-bit integers, where the fourth byte is either a value for alpha, or left empty because it's more efficient to deal with 32 bit numbers than 24 bit numbers (the binary design of computers makes powers of 2 most efficient)


interleaved binary data

windowed or periodic data – a windowed data set might be a signal that has a particular periodicity, and these "frames" might be compared against one another for changes.

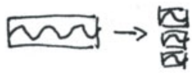

windowed or periodic

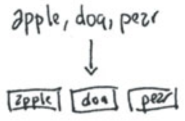
image or scanline – the two dimensional data of an image is unwound into a single stream of data, the width of each line being called the 'scanline', similar to the example of the television signal.

image or scanline

5.3.2.2 Text characters

Text characters can take a variety of forms, whether separated by a simple delimiter like a comma (or a tab, as in the zipdecode example) or must be parsed by a markup language or something more advanced like the *grammars* used to parse the text of a program.

delimited text – plain text lines. One of three types is most prevalent. First, whitespace (a tab or space character) might be used as a separator:



delimeted text

```
APPLE BEAR CAT POTATO TOMATO
```

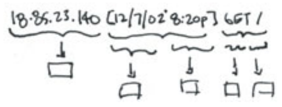
Another common format is CSV, or *comma separated values*, which might be used to save a spreadsheet, and for each row, uses a comma to separate the columns of data (a column that has a comma as data is placed in quotes):

```
APPLE, BEAR, CAT, "WORCESTER, MA"
```

Finally, text files are sometimes fixed width, where each column is separated by padded spacing, so that new columns of data always start at specific positions. This is becoming less common, but was used heavily in older formats and computer systems because programming languages of the time (such as Fortran) were better at splitting data at specific positions, rather than on specific characters:

```
12345678901234567890123456790
apple       1456990    3492
bear           3949      22
cat           33923     619
```

In this example, first column is from character 1 through 12, the second is from 13 through 19, and the final is from 20 through 27. A Fortran programmer couldn't be happier.



varied delimeters

varied delimiters – delimiters are often more complicated, so *regular expressions* are often employed. Regular expressions (or *regexp*) are a way to match text that might vary in length or have delimiters that change. Regexp commingle special sequences like \w, which means match any 'worD´ character (the letters a-z and A-Z) or \d which means any numeric digit (0-9). Next, the modifier + can signify "one or more" or * might signify "zero or more" For instance:

```
\w+\s\w+
```

means to match any number of "word" characters, followed by some kind of whitespace (tab or space char), then another series of one or more word characters. Regexp can become very complex, for instance, the following is a single line of text is taken from a web server activity log:

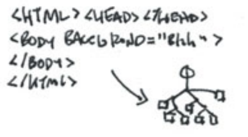
```
mail.kpam.com - - [22/Apr/2004:18:56:49 -0400] "GET /people/pcho/
typemenot/ HTTP/1.1" 200 603 "http://dir.yahoo.com/Arts/Design_
Arts/Graphic_Design/Typography/" "Mozilla/5.0 (Macintosh; U; PPC;
en-US; rv:1.0.2) Gecko/20030208 Netscape/7.02"
```

to parse this into a series of useful pieces (using the Perl programming language), the following regexp is used:

```
^(\S+) (\S+) (\S+) \[(\d+)\/(\w+)\/(\d+):(\d+):(\d+):(\d+)
([^\]]+)\] "(\S+) (.*?) (\S+)" (\S+) (\S+) "(\S+)" "(.*)"$
```

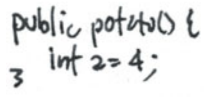
Internal to the computer, regexps are compiled programs and can run surprisingly fast.

text markup languages – while regexps are useful for single phrases or lines of data, tagged formats, such as HTML, are popular for whole documents. As evidenced by the enormous amount of interest in formats like XML, these can be useful way to tag data to make them easy to exchange.



text markup languages

bnf grammars – a step past regexps and markup languages are full BNF grammars, most commonly used for the parsing of programming languages, and also useful in protocol documents (i.e. the documentation for the HTTP standard that describes how communication works between web browsers and web servers) because of their specificity. A BNF grammar, which is human readable and editable, is compiled to program code by a *parser generator*, that makes far less intelligible, but very fast-running code. An example of a (not very strict) grammar:



bnf grammar

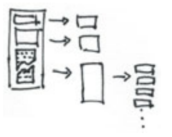
```
WORD           = TOKEN | QUOTED-STRING

TOKEN          = 1*<ANY CHAR EXCEPT CTLs OR TSPECIALS>

TSPECIALS      = "(" | ")" | "<" | ">" | "@"
               | "," | ";" | ":" | "\" | <">
               | "/" | "[" | "]" | "?" | "="
               | "{" | "}" | SP | HT
QUOTED-STRING  = ( <"> *(QDTEXT) <"> )

QDTEXT         = <ANY CHAR EXCEPT <"> AND CTLs,
                 BUT INCLUDING LWS>
```

As can be seen in this example, the grammar is built all the way up from small components like characters, and into more complicated structures like a quoted string.

5.3.2.3 Structured data

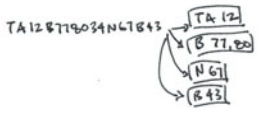
Structures and hierarchies refer to the common situation of a data set having a hierarchical structure of elements within elements, or elements that have pointers to one another.

structured decoder – many kinds of data have a hierarchical format of objects within objects. For instance, a hard disk is made up of a *partition table* that describes its hierarchy, followed by a series of partitions that show up as individual drives on a user's machine. Each partition has a *file system*, which describes how files are laid



structured decoder

out on the disk and where they are located physically on the disk. In the case of a Mac OS file system, the files are made up of two forks, one for actual data and the other for resources, or meta-data. The resource fork itself has a table that describes what kind of resources it contains and where they're located within the file. Each resource has its own structure that provide still more tables within tables. For highly structured data, these can be dealt with in a very generic way (where most things are tables or lists or values, etc).



structured packing

structured packing – data might be packed into a stream to make it more condensed, though it's not necessarily considered compression (described in the 'acquire' section). One case is Postscript fonts, where an encoding called CharStrings is used to convert individual commands for drawing a letter (like lineto and moveto) into specific binary numbers, making the file easier for a program to parse (than plain text) and also more compact.

computational parser

computational parsers – Non-uniform data structures require more logic than a simple description of hierarchy, or unpacking based on indices. These are dealt with by a program designed to understand the higher variability of the structure. No doubt some rules are employed, but they are implemented via a full programming language because they are too complicated to abstract.

5.4 FILTER

The filtering step handles preparing a relevant subset of the data to be considered by the user. It is strongly tied to the later 'interact' step, because the data of interest might change based on

The filter used in the zipdecode example of chapter two was basic, removing cities and towns not a part of the contiguous 48 states. More advanced filtering can be seen in section 4.7, where the user is capable of swapping between a view of the "affected" versus "unaffected" individuals in the study, or a combination of both. Comparing what the data looks like for individuals that exhibit symptoms of the disease in question might reveal clues about

In another case, the initial data set in question might be extremely large, another project, used more than a hundred regions of data (rather than the single ones shown in the chapter four examples) that were available as a single large data file. The filtering step in this case broke the data into its constituent regions so that they could be handled separately, or even re-exported based on a per-region basis, for use in tools that could only handle individual regions.

The filtering step sits in the middle of the process steps, the first step that doesn't handle the data in a completely "blind" fashion, but its methods are not as advanced as the statistics and mining methods of the step that follows.

5.5 MINE

The name of this category is simply meant to cover everything from mathematics, to statistics, to more advanced data mining operations.

5.5.1 *Basic Math & Statistics*

As described by the quote from John Tukey in chapter three, statistics is fundamentally about understanding the contents of data sets, to get an intuitive sense of what is represented by the numbers. Statistics provide an interpretation for the overall "shape" of data. Among the methods are sorting operations, distance metrics, standard deviation, and normalization.

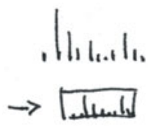
Another important aspect of statistics is understanding the amount of coverage in a data set. A plot of even as many as 500 points across a grid of size 100 × 100 will still only cover 5% of the area. This small amount of coverage should be considered when trying to make the data set as compact as possible – for instance, it might be a clue that the grid of 100 × 100 might be a wasteful representation of the data.

A selection of additional methods and their use:

max/min

max & min – simply calculating the highest and lowest value in a list of numbers (used in the zipdecode example to determine the ranges for latitude and longitude)

median & mean – the median point is the very middle point of the data set. For a set of five numbers, it is the third, regardless of its value. The mean, or *average*, is the sum of all the numbers divided by the count.


normalize

normalization – a basic normalization is to re-map a set of numbers to range from zero to one, making them easier to handle for subsequent calculations. In the zipdecode example, this was employed for latitude (and longitude). The span was calculated from the max and min:

```
span = maxLatitude - minLatitude
```

Then, each latitude was subtracted by the minimum value, and divided by the span, so each number would fall between zero and one:

$$\text{normalizedLat}_n = (\text{latitude}_n - \text{minLatitude}) \ / \ \text{span}$$

Having done this, zooming calculations are simpler, or even without zoom, the y position of each point can be calculated by multiplying the number between zero and one by the height, producing a number between zero and the height.

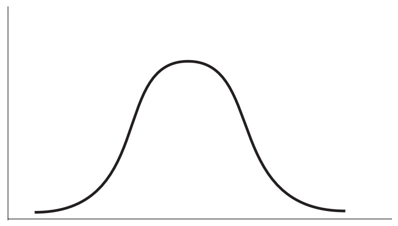
variance – provides a measure of how numbers are distributed, the formula for variance:

$$\sigma^2 = \sum(x_i - \text{mean})^2 \ / \ n$$

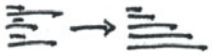
Where n is the count of the numbers to be observed, and x_i is each number. The formula above simply means: 1) subtract a number in the list from the mean, and square the result. 2) divide that value by the count. 3) do this for each number in the list, and add them all together.

standard deviation – this is the square root of the variance, and it helps one to understand where numbers fall in the data set in question. The *normal* distribution looks like:

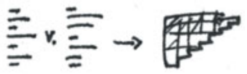
standard deviation



and the standard deviation provides a means to calculate where on the curve the number in question is located.

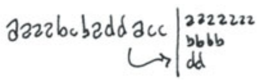

sorting

sorting – generally speaking, types of sorts fall under three categories. First, based on simple numeric criteria (ascending or descending values), next alphabetical sorting of strings, or finally, based on a compare function, that might use each value as a lookup into a dictionary, and the results of the respective lookups are compared against one another. There are many methods for sorting, the most common software algorithm is QuickSort, which, as its name implies, is the fastest method for handling randomized data sets (perhaps oddly, it performs its worst when the data is too ordered).



distance metrics

distance metrics – a distance metric is used to calculate the distance between two data points, or sometimes two *vectors* of data (where each 'point' might have multiple values). For instance, a distance metric is used in section 4.4 to calculate how similar each row of data is so that it can be clustered. A *similarity matrix* is also employed, which is a half-matrix of each row compared to the others (not unlike the D′ table).



count unique instances

count unique instances – if column of data is made up of one of 20 different words, a common task will be to make a list of all the words that are unique in the set, along with a list for how often each word appears.

5.5.2 *Dimensional Measures & Transformations*

The last sections consider methods commonly associated with data mining. Dimensional measures and transformations include methods for handling multidimensional data sets, such as principal components analysis, which is used to determine the most 'important' dimensions in a data set, or the fourier transform, used to break a signal into its component frequencies.

principle components analysis – PCA is a method from linear algebra to decompose matrices that have many dimensions into a smaller number of dimensions by calculating which of the dimensions in question are of greatest importance.

multidimensional scaling – is similar to PCA, but used when the data set looks like a Frank Gehry building. Rather than the single planar dimensions of PCA, multidimensional scaling untangles data while still trying to maintain relevant distances between points.

fourier transform – the fourier transform converts data to frequency space, or the inverse fourier transform (IFT) can be used to transform it back. On a computer, the fast fourier transform, or FFT is most commonly used, because it is far faster (remember QuickSort) than the mathematically intensive fourier. But this leads to a problem with FFT, because the frequencies it calculates are log-based, which is adequate for many situations but not all. Another alternative is the discrete fourier transform, or DFT, in which specific ("discrete") frequency bins are calculated.

autocorrelogram – these are used in particular for audio applications, to show an additive time-lapse plot of how a fourier transform of the data changes.

5.5.3 Classification, Scoring, & Search

Once the data has been fit to a model and dimensionality has been determined, classification might be used to group related elements. Clustering methods (such as k-means or self-organizing maps) exist to break a data set into hierarchies of related elements. Finally, evaluation metrics are considered, to determine how well the model fits the data, and whether it was a sufficient predictor of organization.

clustering – used in section 4.4 to group similar rows of data together for easier comparison, or often in microarray experiments [Eisen, *et al* 1998]. Several clustering methods exist, the method for that section was an algorithm called CAST [Ben-Dor, *et al*. 1999] and another more popular method is called *k-means*.

probability – as demonstrated briefly in the discussion of Linkage Disequilibrium in chapter four, probability can be used to estimate information-rich areas in a data set.

self-organizing maps – Kohenen's self-organizing maps are a another means of classification, based on an iterative process of selection [for an introduction, see p.69 Fayyad, Grinstein, & Wierse. 2001].

dimensional reduction – Sammon dimensional reduction is an additional such method [for an introduction, see p.69 Fayyad, Grinstein, & Wierse. 2001]

scoring methods – a class of data mining methods for determination of "what's interesting" in a data set.

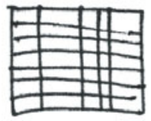
search & optimization methods – another class of data mining methods, which focus on rapid search and analysis.

5.6 REPRESENT

The fourth part of the process considers representation in its most basic forms. This is a laundry list of techniques, some of them older, like the bar graph invented by Playfair, or more recent, like parallel coordinates. The visuals are themselves a catalog of starting points to be used by the designer when considering the data in question. The thesis will describe each of these techniques in turn, and how they apply to a variety of data types. The intent is that by this point, the designer has the appropriate means with which to choose what type of representation might be most appropriate, given how the data has been pulled apart in the first three steps of the process.

5.6.1 Catalog of Common Representations

The choice of representation is based on what is the simplest possible form that conveys the most relevant aspects of the data set.



table

table – the display of tables from a database or spreadsheet is the standard form for two dimensional quantitative data. Tables are useful for showing all the data, but when too many rows or columns are required they are quickly cumbersome. The invention of the spreadsheet (in the form of VisiCalc, invented by Dan Bricklin) in the very late 70s was a major invention that allowed users to show "sheets" of their data spread across multiple pages of printouts or on-screen.

scatter plot

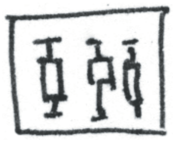
scatter plot – in one dimension, a scatter plot is a disconnected line graph with one axis as the point count. In two dimensions, it is a cloud of points with horizontal and vertical locations based on their values. These can be extended to three and more dimensions by transforming down to the two dimensional plane (same as how 3D graphics map a spatial scene to the two dimensional plane of the monitor). More than two dimensions will require the ability to swap dimensions or rotate across dimensions to show how they relate to one another.

line graph

line graph – a series of points connected by lines *bar graph* – Playfair's invention for showing series data (usually done with a line graph) where values were not connected to one another, or had missing data.

bar graph

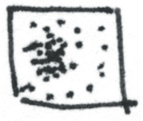
box plot – a two dimensional plot that shows a point and its first (and sometimes second) standard deviation, a useful depiction of the fact that data is often not simply discrete points, but ranges of likelihood. This was invented by John Tukey, more on his role in Exploratory Data Analysis is described in the third chapter.


box plot

physical map – ordering elements by physical location, such as the latitude and longitude points of the zipdecode example.

physical map

heat map – a map that uses color or some other feature to show an additional dimension, for instance a weather map depicting bands of temperature.

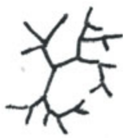

heat map

matrix – any two dimensional set of numbers, colors, intensities, sized dots, or other glyphs.

numeric matrix

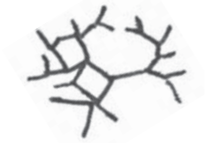
half matrix – where only half a matrix is shown, usually used for similarities, or where two items are being compared against one another (i.e. the D´ table). Only half the matrix is needed because it is the same when reflected across its diagonal.

half matrix



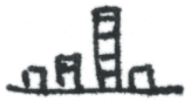
tree

tree – hierarchically ordered data connected by lines of *branches*. Trees are very common because so many data sets have a hiearchic structure. However, even though the data is hieararchic, this is not the proper representation, because the understanding sought from the image is not associated with this hierarchy.



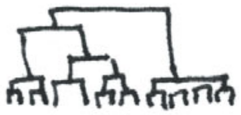
graph

graph – a tree that has less order, and can connect back to itself. Rather than a pure hierarchy, it is a collection of nodes and branches that connect between them.



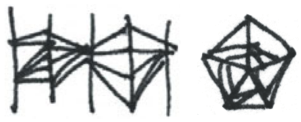
histogram

histogram – a bar chart that displays how many instances of each value on one axis is found. For example, used with a grayscal e image where the horizontal axis are possible color intensities (0..255) and the vertical is the number of times that each color is found in the image.



dendrogram

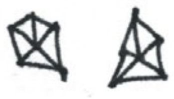
dendrogram – a stacked tree shown connected to points, where the height of the branches show an additional variable. Often used to depict the strength of clustering in a matrix.



linear and radial
parallel coordinates

parallel coordinates – used for multi-dimensional data, where vertical bars represent each dimension. Each element of the data set has values for each dimension, which are shown as points along the vertical axis and then connected together.

radial parallel coordinates – like several superimposed star plots that show multiple records of data.



star plot

star plots –similar to parallel coordinates, but with a single record of data where its points are shown radially.

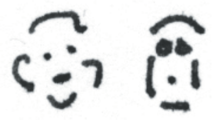
permutation matrix – Bertin's sortable bar charts for the display of multi-dimensional data.

permutation matrix

survey plot – popularized in the "Table Lens" project from Xerox PARC [Rao and Card, 1994], these resemble series of bar graphs that can be sorted independently.

survey plot/table lens

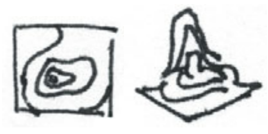
chernoff faces – a method for diagramming multi-dimensional data through the use of facial features. Chernoff's idea [Chernoff, 1977] was that because our visual system is particularly tuned to understanding and remembering human faces, that people would be able to more readily understand many more dimensions as mapped to a face than might be possible with other types of glyphs or diagrams.


chernoff faces

rubber sheet – like a heat map, but used to map four or more dimensions, through the use of a colored, three dimensional surface.

rubber sheet

isosurfaces – maps of data that resemble topographic maps.


2D and 3D isosurfaces

tree maps – first popularized by Shneiderman in [Shneiderman, 1992], and later used for Wattenberg's successful "Map of the Market" that depicts a hierarchically ordered set of boxes within boxes for the sectors, and largest stocks in the market.

tree map

visual diff – a common differencing representation that shows two columns of text connected by lines to show where changes have been made between the two versions.

visual diff

5.6.2 Regarding 3D and 2D Representations

When presented with a problem of many dimensions, an initial instinct is often to employ a three-dimensional view of the data. This is problematic for many reasons:

> The plane of the computer screen is inherently 2D. The advantage of truly three-dimensional objects is being able to look around them, see inside, etc. Most of these affordances are missing in screen-based representations.
>
> Users tend to have a difficult time with understanding data in 3D, because of their unfamiliarity with such representations. This was reportedly a problem that held back the wider adoption of Strausfeld's *Financial Viewpoints* project. While it might be engaging, we're limited by our familiarity with two-dimensional computer screens and printouts.
>
> Because the number of dimensions being looked at is more like six, it's often the case that additional confusions coming from a three-dimensional representation serve to only detract further.
>
> Sometimes it makes more sense to make it easier to navigate between multiple two-dimensional planes of data that show multiple aspects. For instance, a *Financial Viewpoints* style representation that "locked" to each particular view.

Considerations for using three-dimensional representations:

> Understand that the screen is not a 3D space, so the additional dimension should show data that can be occluded by other points that might be "in front" of it.
>
> The use of subtle perpetual motion to rotate the representation can help the viewer identify the contents of the data set because this tends to construct a 3D image in one's own mind.
>
> Try to flatten the dimensionality wherever possible. If the data is mostly two dimensions, with a couple of spikes in another dimension, those might be shown differently, but needn't make the entire representation multiple dimensions to meet the lowest common denominator in the data set.

5.7 REFINE

Even though data sets can be boiled down to simply lists, matrices (multi-dimensional lists), and trees (hierarchic lists), it is not sufficient to simply assign a representation to a data set that it fits. Each of the tools from the previous step can be applied depending on what category the data falls into, but it is the task of the designer to then pull out the interesting or relevant features from the data set by refining the basic representation through the use of visual design techniques.

Graphic design skills provide useful fundamentals for the type of questions to be asked when seeking to communicate the content of a complex data set. The thought processes ingrained in the visual designer, everything from "How can this be shown most cleanly, most clearly?" to more subjective "Why is that ugly and confusing?" are best suited to the goals of clearer communication.

Perhaps the best reference on visual refinement as regards to data are Tufte's series of books [Tufte 1983, 1990, 1997] that clearly make the case for a broad audience.

5.7.1 Contrast and Differentiation

The goal of information design is to show comparisons between elements. The comparisons can be highlighted through:

> *Contrast* – Contrast is perhaps the most fundamental visual property. This is because our visual system is tuned to finding contrasts, whether to understand where one object ends and another begins, or at a higher level, for decision-making processes. The pre-attentive features shown in section 3.1 are all means of contrast between visual elements.
>
> *Hierarchy* – Related to contrast is hierarchy, the means with which an order of importance between elements is established on a page. As is the case for writing, for instance, an information graphic should provide narration about what is being represented, what it means, and reveal its details. Just as a topic sentence must be found in every paragraph, so should a graphic place emphasis on a list of its larger points, while de-emphasizing minor supporting points.

Grouping – "Grouping is represented by objects that cluster or gravitate to each other, implying likeness or some common value or meaning. Grouping creates patterns." [Christopher Pullman, private discussion] This can be as simple as two colors being used to show what objects are part of what group. A more sophisticated example is the redesign solution in section 4.4 that used a visual grouping of elements to highlight the similarities between objects, by providing a clearly delineated set of groups found within the data. At the same time, it also helps to highlight the differences, by placing broadly similar elements together, it also helps to highlight the differences, the minute changes between rows of data.

5.7.2 *Visual Features for Differentiation*

Size, color, and placement are some of the devices used by graphic designers to differentiate and provide contrast. This section describes their use and misuse.

5.7.2.1 Size & Weight

It is important to pay careful attention to how the relative sizes of elements work with one another. Size is a powerful pre-attentive feature (see section 3.1) that can highlight or distract.

The size or thickness of lines can often be used to show relative importance between two sets of values, or a thick line might be used to differentiate from thinner lines beneath it that denote a grid used to show the scale of the figure. Using such a method to decrease the impact of the gridlines is important to placing the most prominence on "showing the data" [Tufte, 1990], rather than allowing the viewer to become distracted with data versus the design of the diagram itself. This can also be applied to lines used as borders around shapes, which are almost always extraneous additions. This is discussed at length in Tufte's text, described as "reducing the amount of non-data ink," a useful phrase that addresses a remarkably common issue when evaluating diagrams.

5.7.2.2 Color

On the computer, color is a mixture of proportions of red, green, and blue components, collectively referred to as RGB. This is an artifact of how the color is generated on screen, where tiny elements of each are placed closely to give the illusion of color to the eye. A more intuitive, traditional model of color uses ranges of hue (or chroma), saturation, and value. When referring to color, what is usually meant is its hue, for instance orange or blue or violet. Saturation is the intensity or richness of the color, and value is the brightness, a range of white to black.

Color is often employed as the most obvious means to denote between categories of data. Yet this quickly becomes problematic for several reasons. First, the human brain can only distinguish between and recall roughly five separate colors [via Ware 2002], where the number of categories of data will more often than not exceed five, leading to sets of ten or even twenty colors that are hardly distinguishable from one another.

Second, overuse of color often leads to diagrams that use identical colors to depict values that are unrelated, for instance combining a full-spectrum scale that shows “temperature” while also using five colors (that can be found in the scale) to delineate physical boundaries to which the temperatures apply.

Often, the output media may not support color, the majority of printing and copying devices still black and white.

And finally there is the issue of color blindness, such as dichromatism, affecting approximately 8% of men and 1% of women, meaning that the colors are often likely to be perceived differently across the target audience. In the end, color is best used sparingly, and most diagrams should be made to work on their own in black and white, using color only to punctuate or support another feature that might be optional.

A paradox to the use of color is that in software, which makes available millions of colors to the user, the tendency, rather than a variety of sensible colors, is to use the maximum value of “red” to produce a red, and zeroes for the other values. This produces a jarring non-natural looking color that easily distracts a viewer. A better method is to consider how colors are found in the natural world, where any colors are rarely if ever pure shades of color components, and instead subtle mixtures of the red, green, and blue (or hue, saturation and value under the alternate model). Sampling colors from a photograph, for instance, can produce a set of colors that are already demonstrated to work together, along with many shades of each for highlights or secondary information.

When many categories of data are to be considered, one might employ numbered annotations, or small symbols known as “glyphs” or “icons” instead of color. Even small pieces of text are often better than a dozen colors competing for the viewer’s attention. More often, consideration should be given to whether it is important enough to show every single category in the diagram, because the extreme variety will likely dominate the representation, clouding the communicative effect of other features that have greater importance (even if they are larger or more colorful). The quantity of identical elements is another signifier of importance, because the brain is inclined to search for patterns and groupings in an image to quickly draw conclusions.

5.7.2.3 Placement

Placement conveys hierarchy by providing an ordering to elements, for instance, the way that the title is found at the top of every newspaper article. Contrast might be conveyed by placement through the use of a set of similarly placed elements and an outlier that is disconnected from the rest of the group. Grouping is the predominant use of placement.

In addition to contrast, hierarchy, and grouping, the placement of elements also plays a role in how a diagram is read, with the majority of audiences reading left-to-right, top-to-bottom either natively, or at least being familiar with that mode for the context of scientific publications. Combined with size changes, the designer can “lead” the viewer’s eye around the diagram, not unlike how a narrative thread runs through a paper. The eye will begin with visually strong (large, colored, with prominent placement) elements and then search for details among those less prominent.

5.8 INTERACT

Interaction methods involve either how the data interacts with itself on-screen (the type of interactions investigated by the VLW), or how users can interact with and control the data representation, the latter being the field of HCI, or Human-Computer Interaction.

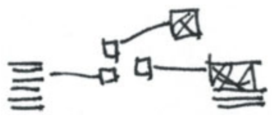
5.8.1 *Data Self-Interaction*

Basic layout rules might be simple connections between elements in the diagram (a rule to maintain left alignment between two objects) through more complicated design rules that use data tagged by its "importance" to achieve a continuously updating composition.

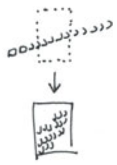
5.8.1.1 Basic layout rules

Some of the layout rules used in common practice, most based on straightforward algorithms:

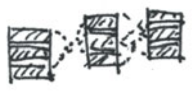
connected elements – graphics that connect one element to another, i.e. a line connecting a box to a magnified version of the same

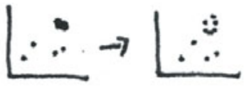

connected elements

spatial rules – items must fit within a certain overall area – scale all elements to fit a predefined space. For instance, *line and page breaks*, where at a certain width or height, a new line or new page is started.


spatial rules

proportional rules – this might control the even distribution of spacing between elements.


proportion rules



jittering

jittering points – slight perturbation of data points to prevent occlusion, and show when points are stacked on top one another.

weighted scale & zoom – consideration of the optical effects of scaling. Through *hinting*, or other more advanced techniques like MetaFont or Multiple-Masters, typefaces at larger sizes typically show more detail and subtlety than when rendered at small sizes. This can be a powerful means to add more detail to a representation as the viewer looks closer.

label stacking – without adequate room for labels that would otherwise occlude one another, simply showing them as a list.

cartographic labeling – on a map with many cities and towns, non-overlapping labels are needed

gridlines – major and minor axes, weighted towards 'pleasant' numbering that doesn't interrupt reading of the data.

5.8.1.2 Physics-based placement

physics rules

It is common for contemporary information visualization projects to use physics rules for movement and placement. These rules are generally based on a series of masses (data points) and springs (connections between them, visible or invisible). A simple Hooke's Law spring can be used to set a target length between two objects and over time these can be drawn together. For instance:

attraction – move elements closer based on a criteria

deflection – push elements away based on some criteria

Both can affect either single elements, or overall groups, e.g.

single element – move item closer to another item based on their relatedness

multiple elements – pull all parts of a group together around a center point

These rules were discussed at length in the author's Master's Thesis [Fry, 2000], where a physics based model was used to introduce a kind of behavior to a data set for data sets such as web traffic or the content of bodies of text.

5.8.1.3 Smart graphics, design knowledge

There are limits to the ability of designers, because we cannot singularly specify the attributes of hundreds of thousands of elements, tweaking each for perfection. The computational process becomes a way to extend the power of the designer to address far larger and more difficult problems than previously possible.

Instead of specific layouts, the practitioner instead designs a set of rules (sometimes called *metadesign*) over the system to determine behaviors or to make basic design decisions that fall within a set of criteria.

Another common instantiation is that of *knowledge-based design systems* [Coyne *et al*, 1990], this research area is often treated separately from the other components of the task (everything from data acquisition to final color choices), rather than as an integral part of the design process itself, where some basic building blocks of rules are provided while others are developed by the metadesigner or visual programmer.

Live elements need to be part of the building blocks, but it needs to be done in a way that includes a well-chosen set of building blocks (and a means to add new ones). The failure of other attempts at this approach is that they were considered an entire means to an end, existing in an academic vacuum away from the other parts of the process. The components need to be made abstract in such a way as they can be applied in a straightforward manner, without the fact of their being live distracting from the other essential parts of the design process.

5.8.1.4 Calibration

Rule based, dynamic systems are notoriously difficult to properly calibrate. The lure of rule-based systems is in the simplicity of the individual rules and their ability to cover complex behaviors. However, these rules also become subject to oscillations (as rules conflict one another) or at worse, explosions, where an errant rule might corrupt the state of one element, and because of the inter-relationship between elements, that corruption is quickly passed to the others in the system, breaking its overall state.

As a result a significant portion of the time in using rule-based systems is in their proper calibration and testing of many data input scenarios, not just in determining the rules themselves, the latter of which may require only negligible time when compared to the other steps.

5.8.2 User Interaction

5.8.2.1 Movement & Viewing Controls

The final step to consider is how to interact with the data. Perhaps the representation is dynamic, actively modifying itself to call attention to important features. Viewing controls can be used to get an appropriate camera angle or warped view of the space of the representation, such as using a fisheye lense to view a large data set in a very compact space.

Sampling controls are interactive dipsticks, allowing the data to be sampled at specific locations. Other user interface elements such as sliders can be used to set proportions or ranges, all the while dynamically updating the view of the data set.

5.8.2.2 Restraint of choices/options

In software, it's easy to provide more options to the user. What's more difficult, and more important, is figuring out how to limit the number of choices to that which is most relevant to the majority of the tasks executed by the majority of users.

When approaching analysis of a complex data set, it's useful to provide a 'default' view to the user, so that it's clear how the functionality of the system works. Instead of first displaying a complicated set of knobs and sliders that control every aspect of how the data might be queried, an initial state makes it easy to move laterally (modifying parameters of the basic state) and see quick updates to what's happening. This way, the user can get a feel for the system first, then as specific needs arise (How do I limit this to situation x? How do I increase the value of y?), they'll be inclined to learn those features one-by-one, rather than learning two dozen options up front before they see anything useful.

In short, the design should make an educated guess about what the user will want to do, and then make the options for continuing further accessible just underneath.

5.8.2.3 Scale and Zoom

The *zipdecode* example from chapter two scales to discrete levels—showing intermediate steps to maintain context. This is specifically chosen over a sliding zoom control that would allow the user to move to any particular magnification they choose. The latter type of control has led to a class of "zoomable" UIs, but these often present awkward user interaction, a bit like using the zoom on a video camera: One wants to zoom *at* something, but the camera gives a continuous forward/backward, which makes it too easy to overshoot, inspiring seasickness in viewers because of a series of rapid zooming, overshooting, over-correction, and re-correction.

By first choosing the relevant levels of zoom in an interface, the zoom remains a useful means for moving between viewpoints while maintaining context because of the animation of moving between levels.

Consider the *Powers of Ten* video from the office of Charles and Ray Eames, narrated and written by Philip & Phylis Morrison [Eames, 1978]. While the zoom moves at a continuous speed throughout, the narration provides a punctuation to the relevant levels in the film, highlighting sections of importance. In addition, for a section where successive orders of magnitude have very little occurring, the narration notes this fact as an interesting point to be observed (thus making it relevant to have those magnitudes visible).

The latter point helps set criteria for when and why zooming is appropriate as well. Is it an interesting fact that for several layers there will be nothing happening. Consider a genome browser done with continuous zoom. It would move from the chromosome level (approximately 100 million letters) down to the next relevant section, being individual genes (or even groups of genes). A gene is roughly one thousand letters of code, meaning that the user will be taken through a 100,000 : 1 zoom. This might be interesting for novices learning the relationship between chromosomes, genes, and letters of DNA, but the interface will likely still require discrete levels, so that users don't become lost on the long trip from a the chromosome scale down to the gene scale.

5.9 NEXT STEPS

Some of the more fascinating future directions for this work come from what follows the process as it is described.

5.9.1 Iterate

The ability for a designer to rapidly iterate is key to Computational Information Design. The projects in this thesis underwent anywhere from six to eighty distinct iterations during their development. This might include the addition or removal of features, modification of the visual design, or improving the interaction. Each step feeds into the others, where a change in the 'refine' step might influence how the data is acquired.

5.9.2 Using Results to Modify the System

As pointed out by colleague Axel Kilian, a step not covered in this dissertation is the use of a finalized project to actually provide a redesign for the system that it is representing. For instance, The zipdecode piece shows how the postal codes work throughout the U.S., but could also be used as a starting point for a tool to redesign the system which might soon (ten to twenty years) become saturated and run out of numbers.

5.9.3 Decision Support, Analysis, Prediction

The output of this process might be used for decision support, analysis, or prediction. In this thesis, computation is a means for handling the repetitive aspects of representation (since machines are best suited for such repetitive acts), and for humans to apply their own thinking to decipher the meaning of what is produced, a task that, despite years of AI research, machines are still very poor at executing.

5.10 INDIVIDUAL PRACTITIONERS

The process begins with a stream of data, and aims to produce a 'live' diagram of its meaning. An understanding at one point affects choices that are made further along. Ideally, all parts of the process are handled by a single individual who has a sufficiently deep understanding for each area that allows them to bring all the necessary knowledge to the task. This breadth offers the designer a better approach to understanding data representation and analysis.

Consider these scenarios for why this becomes a necessity:

- The workings of the mathematical model used to describe the data might affect the visual design, because the model itself may have features, such as the degree of certainty in aspects of the data that should be exposed in the visual design.

- Parameters of the data model might need to be modifiable via user interaction. This might help clarify its qualities such as whether it acts in a linear or nonlinear fashion.

- Knowledge of the audience is essential, for knowing what might be appropriate at each step. For instance, a "powers of 10" style of visualization applied to the human genome would likely become tedious for a lab worker, because the animation itself conveys only the well-known structure of the data.

- In other cases, a designer may only become involved near the end of the process, who is tasked with making decisions about superficial features like color. This can be problematic because the designer has to 1) understand the system well enough to make appropriate decisions (made difficult by usual time constraints of most projects, or simply the interest level of the designer) 2) communicate their intentions to the person who will be implementing them, and then 3) convince that person to follow the design closely, lest an incomplete implementation leave the final solution inadequate. As this author knows from experience in the software industry, this is rarely successful even in cases of the best possible working relationships between developers and designers.

6 Tool

Processing is a tool for developing visually-oriented software. It was conceived of as a way to introduce programming concepts to designers, and design concepts to programmers. While it goes back further, on-screen digital graphics have in the past twenty years emerged from the laboratory and made their way into consumer applications like Macromedia Flash-based web sites or motion graphics used for film and television. Drawing on this history, particularly with regards to interactive graphics, it is possible to distill the central themes of programming this type of work and thus make it accessible to a wider audience by codifying these ideas and simplifying how they are implemented.

Why is it called *Processing*?

At their core, computers are processing machines. They modify, move, and combine symbols at a low level to construct higher level representations. Our software allows people to control these actions and representations through writing their own programs. The project also focuses on the "process" of creation rather than end results. The design of the software supports and encourages sketching and the website presents fragments of projects and exposes the concepts behind finished software.

From *processing.org/faq.html*

The tool was created in collaboration with Casey Reas, and more recently has continued to evolve with the aid of an international community of developers collaborating over the web. It is not designed as, or intended to be, the ultimate tool for Computational Information Design, however it provides a means to make the process significantly easier, and was used for nearly all of the work presented in this thesis.

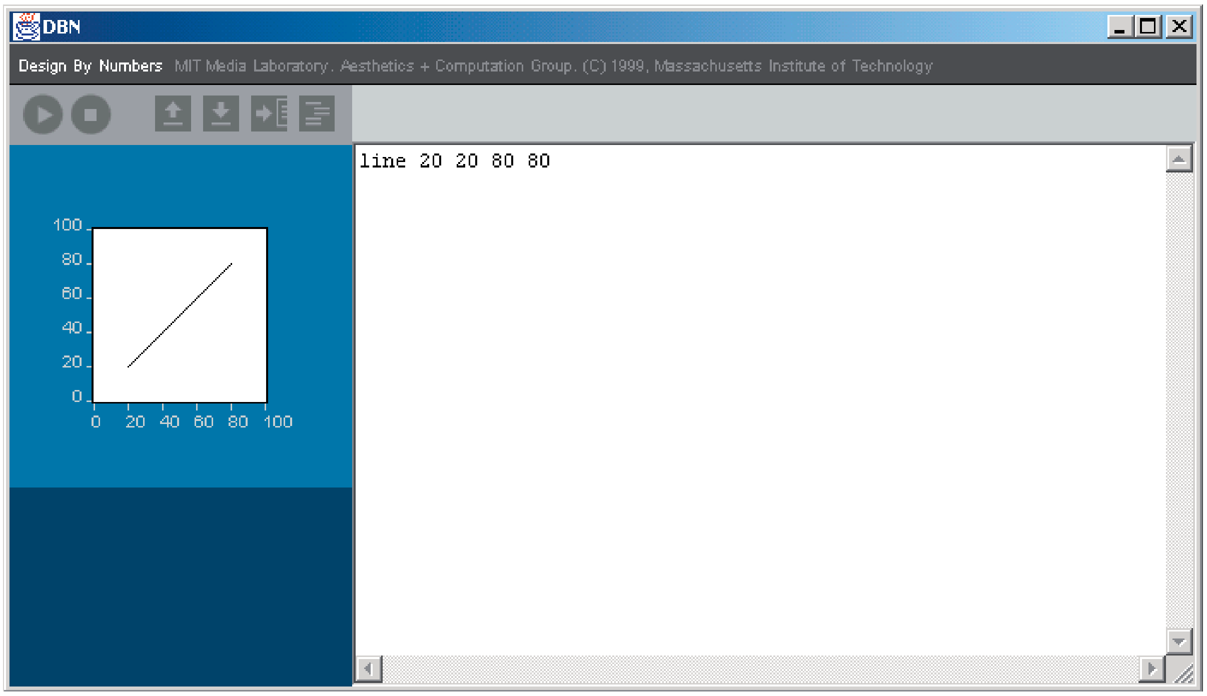
6.1 DESIGN BY NUMBERS

Our research lab at the MIT Media Laboratory has always had a pedagogical focus, inspired in by Professor Maeda's assertion that, with regards to the mix of computation and design:

> *...I believe that these same principles must be applied much more widely, throughout MIT and indeed throughout our university system in general. At least at MIT, there has been for many years an awareness of the need for combining the humanities and sciences at the curriculum level. Despite the best of intentions, however, the model of training is this area remains some form of the humanities wrapped around technology, or vice-versa...It is not enough for us simply to produce a technologist who is aware of the cultural context of technology or a humanities major who can talk fluently about technology...What is needed is a true melding of the artistic sensibility with that of the engineer in a single person.*
>
> MAEDA, 1998

This was part of the inspiration for the development of Design By Numbers [Maeda, 2000], a simple programming language intended to teach the basics of computation to artists and designers.

The strengths of DBN were the simplicity of its distribution, use, and syntax. It was available freely on the web, either for downloading or could be run from a web page with no installation required. This is a notable shift from most programming languages, which require complicated installation and tend to be expensive to purchase.

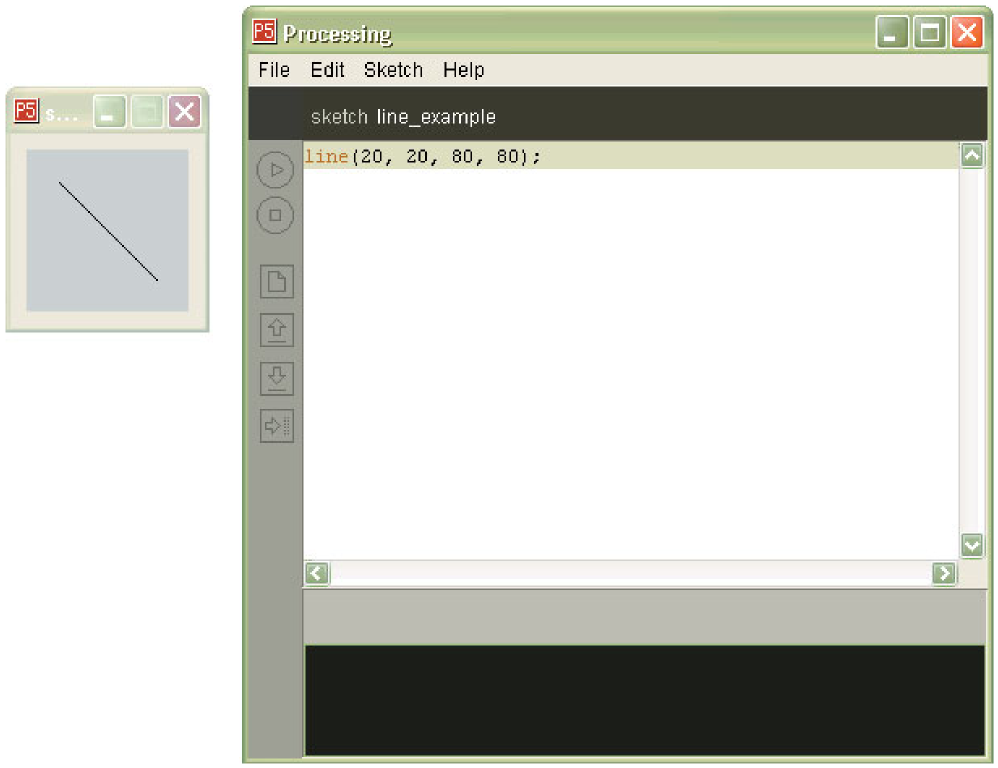

DBN
Design By Numbers MIT Media Laboratory. Aesthetics + Computation Group. (C) 1999, Massachusetts Institute of Technology
100
80
60
40
20
0
0 20 40 60 80 100

```
line 20 20 80 80
```

Typical use of DBN is also far more straightforward than most languages that involve a cycle of "type, compile, run, debug" than can often be cryptic. Code is entered at the right of the window, and run by clicking a button that looks like a familiar 'play' button from consumer electronics.

The DBN system was used for a number of workshops and occasionally for courses at MIT and elsewhere. A supplementary 'courseware' system was developed by Casey Reas that enabled courses to be run online, with projects automatically uploaded to the web and visible via a presentation page that could be used for critiques in a class.

As the primary maintainers of DBN after its inception, we learned a great deal from feedback from users on what was difficult (algorithms and flow of programs), what else they wanted (color, a larger screen, type, ability to distribute programs over the web), and it provided a great deal of insight into the creation of *Processing*, which began as a "next generation" Design By Numbers.



```
line(20, 20, 80, 80);
```

In *Processing* we wanted to move to a more robust programming language, namely Java, but at the same time, to maintain the simplicity of the DBN interface. A system of 'modes' was developed, the first for drawing static drawings, where the user could simply begin typing code without the need to learn functions or other syntax:

The two additional modes get progressively more difficult (but powerful), each essentially designed to act as 'training wheels' for a person learning programming. The final mode is straight Java programming, with a handful of small syntax simplifications, so the skills of learning

Processing are also transferable to Java or even C++ because of their similarity.

6.2 PEDAGOGY

Casey Reas describes *Processing* as part of the shift from command line to GUI finally reaching programming instruction:

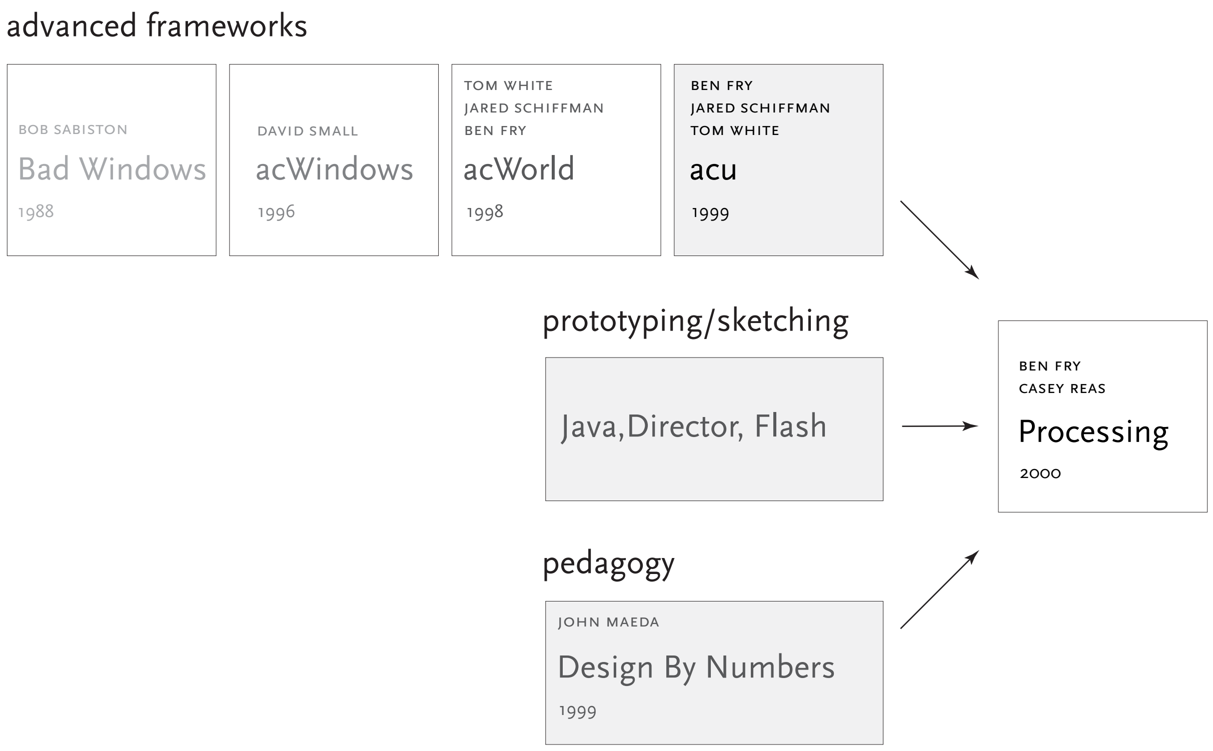
> *Graphical user interfaces became mainstream nearly twenty years ago, but programming fundamentals are still primarily taught through the command line interface. Classes proceed from outputting text to the screen, to GUI, to computer graphics (if at all). It is possible to teach programming in a way that moves graphics and concepts of interaction closer to the surface. The "Hello World" program can be replaced with drawing a line, thus shift the focus of computing from ASCII to images and engaging people with visual and spatial inclinations.*
>
> CASEY REAS [VIA FISHWICK, IN PRODUCTION]

For designers, to make *Processing* more visually engaging from the outset, the idea is to first make things happen, and use that to drive curiosity into learning how it works, and how to do more. Through this alternate approach, our hope is to engage people who might be likely to leave a course on Computer Science "fundamentals" after the second week as they become overburdened in technique and see no clear path to the type of work they're actually pursuing.

For programmers, many have simply not developed graphical software (outside of simply GUI interfaces). This aspect of programming is almost always missing from Computer Science coursework (save for courses specifically in graphics and animation), and the 'getting started' portion of requires a few pages of code before something visual can be presented on-screen.

6.3 ADVANCED FRAMEWORKS

Another ancestor to *Processing* are the many software toolkits built at the Media Laboratory. Its immediate predecessor, called acu, was a C++ and OpenGL-based toolkit that was used to build nearly all the screen-based projects developed in the Aesthetics and Computation Group



between 1999 and 2002. Prior to that were similar implementations, called acWorlds and acJava developed by the same authors. Even earlier, similar projects were underway in the VLW, the earliest known being Bob Sabiston's "BadWindows."

These frameworks (acu in particular) provided a basic structure for interactive graphics research in dynamic typography [Cho, 1999], visual programming systems [Schiffman, 2001], audio-visual environments [Levin, 2000], and methods for information visualization [Fry, 2000]. At the time this work required high-end workstations, but can be reproduced today on nearly any recently purchased PC. This shift has blurred the lines between the requirements for advanced research, and that of sketching and prototyping.

6.4 SKETCHING & PROTOTYPING

The Java programming language enabled web-based distribution of mini-applications, called applets, that had nearly the full functionality of a desktop-based piece of software. The software compiles to a crossplatform byte code, which can be run without modification across any platform implementing the Java Virtual Machine (i.e. Windows, Mac OS, or Linux).

In addition to the ease of distribution, Java tends to make a better prototyping language than C++, both for its simplified syntax (the language designers begin with C++ and removed features that caused common problems) and built-in exception handling, meaning that program errors will be reported, rather than crashing the program, or worse, taking the operating system along with it. These two features lead to Java's usefulness as a basis for the "sketching" environment sought in *Processing*.

Lacking in Java is the fact that two or three pages of standard code are often used across dynamic graphics projects, simply to handle setup of the program and handling basic mouse interaction. By making the assumption that *Processing* was intended for interactive graphics applications, this functionality was built into the base program, so the user could begin creating programs, rather than focussing on infrastructure.

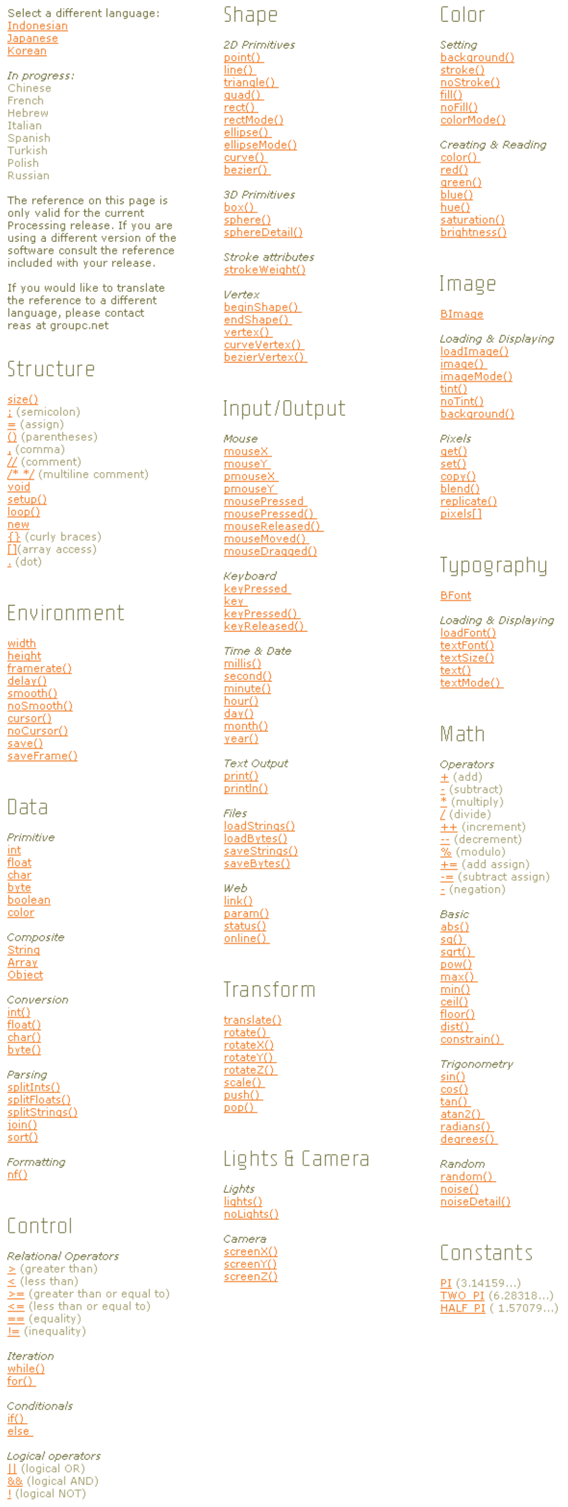
6.5 API

Also lacking in Java were its graphic capabilities. The first most commonly used version of Java, version 1.1, had extremely limited graphics support. Subsequent versions (1.2, 1.3, and so on) addressed the problem, but in their attempt to provide an API (*application programming interface*, a set of features or functions) that could do everything, the most basic functionality (i.e. drawing a thick line on the screen) would now require several lines of code and significant overhead. Worse, these later versions of Java have not received the same wide distribution as the initial 1.1 in part because of legal battles between Sun (proprietor of Java) and Microsoft (accused of violating their contract).

To address graphics for older versions of Java, a new graphics engine was created. For the API issue, this graphics model was designed to be extremely compact, so as to simplify drawing tasks. Because of the more specific audience (people building interactive graphics), this API could be made more compact, which makes it easier for programmers to memorize nearly its entire feature set.

The imaging model also borrowed a concept developed by Tom White for the acu framework, where 2D and 3D graphics could be mixed in

The entire set of functions (the API) for *Processing*, designed to be terse vocabulary that provide only the most relevant features used by the greatest majority of users.



Select a different language:
Indonesian
Japanese
Korean

In progress:
Chinese
French
Hebrew
Italian
Spanish
Turkish
Polish
Russian

The reference on this page is only valid for the current Processing release. If you are using a different version of the software consult the reference included with your release.

If you would like to translate the reference to a different language, please contact reas at groupc.net

Structure

size()
; (semicolon)
= (assign)
() (parentheses)
, (comma)
// (comment)
/* */ (multiline comment)
void
setup()
loop()
new
{} (curly braces)
[](array access)
. (dot)

Environment

width
height
framerate()
delay()
smooth()
noSmooth()
cursor()
noCursor()
save()
saveFrame()

Data

Primitive
int
float
char
byte
boolean
color

Composite
String
Array
Object

Conversion
int()
float()
char()
byte()

Parsing
splitInts()
splitFloats()
splitStrings()
join()
sort()

Formatting
nf()

Control

Relational Operators
> (greater than)
< (less than)
>= (greater than or equal to)
<= (less than or equal to)
== (equality)
!= (inequality)

Iteration
while()
for()

Conditionals
if()
else

Logical operators
|| (logical OR)
&& (logical AND)
! (logical NOT)

Shape

2D Primitives
point()
line()
triangle()
quad()
rect()
rectMode()
ellipse()
ellipseMode()
curve()
bezier()

3D Primitives
box()
sphere()
sphereDetail()

Stroke attributes
strokeWeight()

Vertex
beginShape()
endShape()
vertex()
curveVertex()
bezierVertex()

Input/Output

Mouse
mouseX
mouseY
pmouseX
pmouseY
mousePressed
mousePressed()
mouseReleased()
mouseMoved()
mouseDragged()

Keyboard
keyPressed
key
keyPressed()
keyReleased()

Time & Date
millis()
second()
minute()
hour()
day()
month()
year()

Text Output
print()
println()

Files
loadStrings()
loadBytes()
saveStrings()
saveBytes()

Web
link()
param()
status()
online()

Transform

translate()
rotate()
rotateX()
rotateY()
rotateZ()
scale()
push()
pop()

Lights & Camera

Lights
lights()
noLights()

Camera
screenX()
screenY()
screenZ()

Color

Setting
background()
stroke()
noStroke()
fill()
noFill()
colorMode()

Creating & Reading
color()
red()
green()
blue()
hue()
saturation()
brightness()

Image

BImage

Loading & Displaying
loadImage()
image()
imageMode()
tint()
noTint()
background()

Pixels
get()
set()
copy()
blend()
replicate()
pixels[]

Typography

BFont

Loading & Displaying
loadFont()
textFont()
textSize()
text()
textMode()

Math

Operators
+ (add)
- (subtract)
* (multiply)
/ (divide)
++ (increment)
-- (decrement)
% (modulo)
+= (add assign)
-= (subtract assign)
- (negation)

Basic
abs()
sq()
sqrt()
pow()
max()
min()
ceil()
floor()
dist()
constrain()

Trigonometry
sin()
cos()
tan()
atan2()
radians()
degrees()

Random
random()
noise()
noiseDetail()

Constants

PI (3.14159...)
TWO_PI (6.28318...)
HALF_PI (1.57079...)

the same drawing surface. Most applications are 2D, but this provides a way to mix the two without requiring the developer to learn a 3D programming api, many of which tend to be difficult and counter-intuitive because of the algebra involved. With the added ability to directly manipulate pixels, the set of features provides much flexibility for designers and developers.

6.7 community

Perhaps the greatest strength of *Processing* is that of its community, an international group of users who started as designers or programmers and found the language. In the past year, the number of people to sign up and download the software has jumped from just over 1,000 (in June, 2003) to more than 11,000 (as of April, 2004). Thousands have signed up and use an online discussion area, where new users can get help from more advanced. Users are encouraged to post their latest work and share the code for their sketches as part of a worldwide studio.

About two dozen members of this community are actively involved in developing new, advanced work using *Processing*, which drives further interest because of the number of people who follow their work. Other members are actively involved in developing libraries to be plugged into the *Processing* software, enabling advanced sound synthesis or input devices like drawing tablets. What began as a two person project has fostered many such smaller projects that are included as part of the same support structure.

6.8 elevating practice & critique

An implicit goal of *Processing* is to remove the mystery from programming. People often consider themselves math-averse and therefore incapable of programming, or that programmed works with movement and behaviors that seem complicated are works of complicated mastery. While this is no doubt the case for many works, it is a minority, and the body of example programs included with *Processing*, most of them roughly a single page of code demonstrate many of the algorithms commonly used for motion and interaction.

By making the programming concepts accessible to a wider audience and removing some of the mystery, it places greater emphasis on the creation of meaningful works, rather than technical flourish by those "in the know." The shift from technique to critique is a necessity for moving the field forward. The ability to evaluate is central to the contention of John Maeda's professors in Japan:

> *...my teachers in told me that if I were to do what I did, I would make a lot of money and be the only one doing it, and that I would never know it if I were any good or not.*
>
> JOHN MAEDA [RESNICK, 2000]

It is our hope that the active community of developers producing and sharing works will help elevate the practice, and contribute to making that which is technically "easy" equally as easy for anyone, and that which is "hard" be learnable.

7 Additional Examples

This chapter provides an appendix of several projects developed during the development of the thesis. They represent the evolution of the process behind the work shown in chapters two and four, and helped drive the design decisions that went into the development of *Processing*.

These projects are studies that fall under three categories; genetics, software/computation, and general.

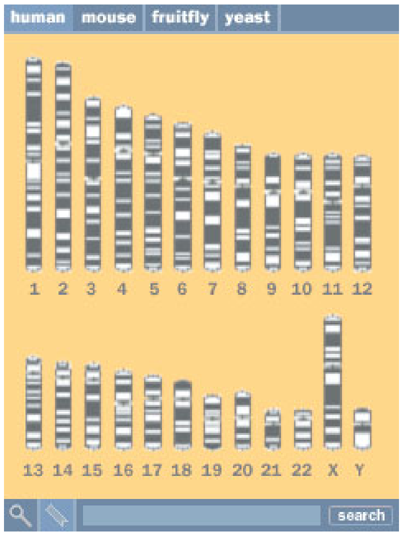
7.1 GENETICS

These projects were developed primarily as studies to pursue aspects of genetic data that were either intriguing, as demonstrations of the way things work, or as a means to demonstrate issues such as scale.

7.1.1 *Handheld Browser Prototype*

ncbi.nlm.nih.gov, ensembl.org, and genome.ucsc.edu

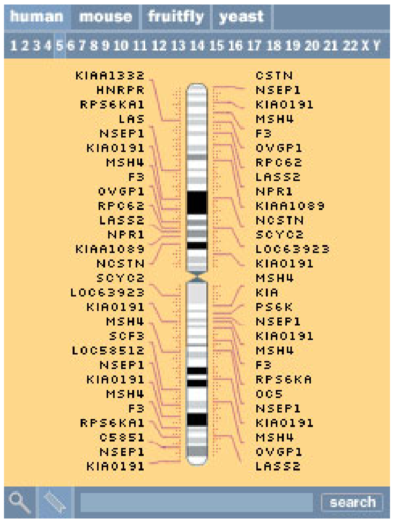
This handheld browser is a simplification of typical online browsers such as those from the NCBI, Ensembl, and UCSC. Focussed in particular on education, it was created in response to the notion that "A CD-ROM containing the entire human genome can be included with every textbook." The data on a compact disc is worthless without a means to interact with it, so the handheld serves as a kind of "biologist's calculator." Similar to an everyday calculator, it can perform basic useful functions, but is seens as a complement to less limited tools that can be run from a desktop workstation.

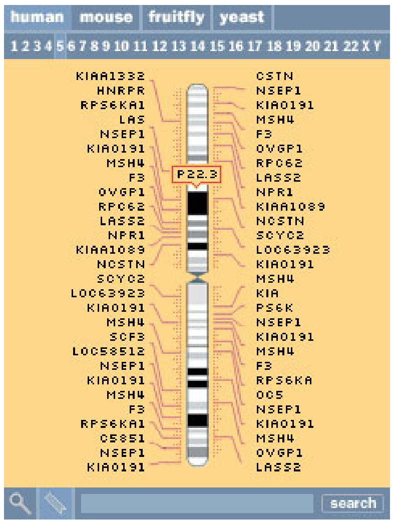


The design presented here is a series of screens that were created as an interaction prototype to consider how such a device would be designed. Much research went into building the backend for such a device, however this was left aside in favor of devloping how the product works as opposed to the actual implementation. Once the design has been worked out, and the backend has been tested for feasability, only software development time is needed for its completion. As such, the screens shown here are not showing live data.

Shown on the previous page, the series of chromosomes, colored according to their cytogenetic banding is shown in the center as a method for selecting a chromosome.

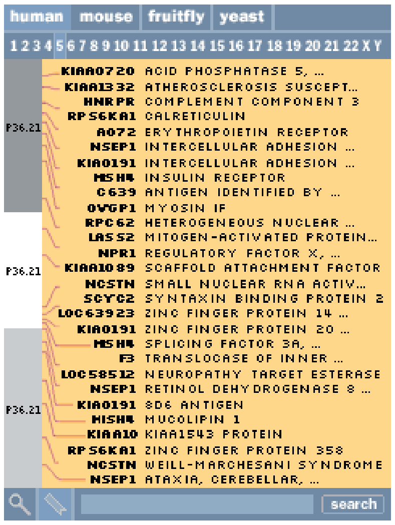
At the top, the user can select between multiple genomes, providing a way to compare and place the data in context. A search function at the bottom provides a method for running a BLAST-style searches or looking for specific search terms.



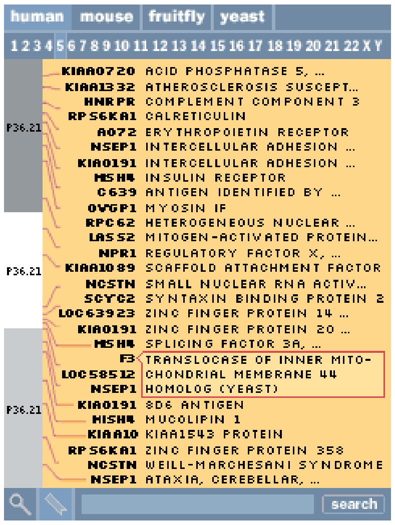

human
mouse
fruitfly
yeast
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 X Y
P22.3
search

Selecting a chromsome (the fifth chromosome is shown here) shows an abbreviated set of the genes along the chromosome. Similar to online browsers, the listed genes are chosen based on size, number of exons, and amount known about each. At right, the result of clicking one of the cytogenetic bands reveals its name. Clicking the numbers along the top allows the user to jump to another chromosome.

Clicking within a particular area takes the user to a zoomed-in view listing all of the genes inside that region. The chromosome band is moved to the lefthand side of the screen.


human
mouse
fruitfly
yeast
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 X Y
P36.21
P36.21
P36.21
search

KIAA0720	ACID PHOSPHATASE 5, …
KIAA1332	ATHEROSCLEROSIS SUSCEPT…
HNRPR	COMPLEMENT COMPONENT 3
RPS6KA1	CALRETICULIN
A072	ERYTHROPOIETIN RECEPTOR
NSEP1	INTERCELLULAR ADHESION …
KIA0191	INTERCELLULAR ADHESION …
MSH4	INSULIN RECEPTOR
C639	ANTIGEN IDENTIFIED BY …
OVGP1	MYOSIN IF
RPC62	HETEROGENEOUS NUCLEAR …
LASS2	MITOGEN-ACTIVATED PROTEIN…
NPR1	REGULATORY FACTOR X, …
KIAA1089	SCAFFOLD ATTACHMENT FACTOR
NCSTN	SMALL NUCLEAR RNA ACTIV…
SCYC2	SYNTAXIN BINDING PROTEIN 2
LOC63923	ZINC FINGER PROTEIN 14 …
KIA0191	ZINC FINGER PROTEIN 20 …
MSH4	SPLICING FACTOR 3A, …
F3	TRANSLOCASE OF INNER …
LOC58512	NEUROPATHY TARGET ESTERASE
NSEP1	RETINOL DEHYDROGENASE 8 …
KIA0191	8D6 ANTIGEN
HISH4	MUCOLIPIN 1
KIAA10	KIAA1543 PROTEIN
RPS6KA1	ZINC FINGER PROTEIN 358
NCSTN	WEILL-MARCHESANI SYNDROME
NSEP1	ATAXIA, CEREBELLAR, …


human
mouse
fruitfly
yeast
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 X Y
P36.21
P36.21
P36.21
search

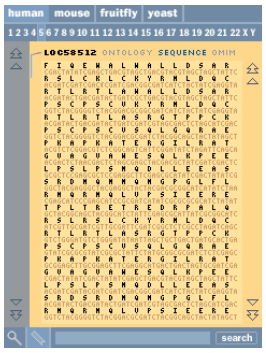
KIAA0720	ACID PHOSPHATASE 5, …
KIAA1332	ATHEROSCLEROSIS SUSCEPT…
HNRPR	COMPLEMENT COMPONENT 3
RPS6KA1	CALRETICULIN
A072	ERYTHROPOIETIN RECEPTOR
NSEP1	INTERCELLULAR ADHESION …
KIA0191	INTERCELLULAR ADHESION …
MSH4	INSULIN RECEPTOR
C639	ANTIGEN IDENTIFIED BY …
OVGP1	MYOSIN IF
RPC62	HETEROGENEOUS NUCLEAR …
LASS2	MITOGEN-ACTIVATED PROTEIN…
NPR1	REGULATORY FACTOR X, …
KIAA1089	SCAFFOLD ATTACHMENT FACTOR
NCSTN	SMALL NUCLEAR RNA ACTIV…
SCYC2	SYNTAXIN BINDING PROTEIN 2
LOC63923	ZINC FINGER PROTEIN 14 …
KIA0191	ZINC FINGER PROTEIN 20 …
MSH4	SPLICING FACTOR 3A, …
F3	TRANSLOCASE OF INNER MITO-
LOC58512	CHONDRIAL MEMBRANE 44
NSEP1	HOMOLOG (YEAST)
KIA0191	8D6 ANTIGEN
HISH4	MUCOLIPIN 1
KIAA10	KIAA1543 PROTEIN
RPS6KA1	ZINC FINGER PROTEIN 358
NCSTN	WEILL-MARCHESANI SYNDROME
NSEP1	ATAXIA, CEREBELLAR, …

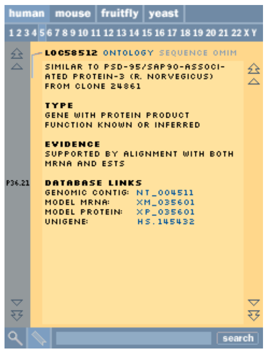
Gene names are shown in bold, and a short description is shown to their right. At the right, clicking on a description reveals the full text in cases where it is too long to fit on the line.

Clicking on a gene shows a basic description of the gene's features. Tabs above the content area allow the user to select between different data views for that gene, in this case the sequence view, the ontology description, or text from the OMIM database.

Below and to the left, the sequence view of the gene. Above the letters of genetic code are its amino acid translation (for exonic regions), and a scrollbar at the right allows the user to page through the data.

At the right, a page from the OMIM database (Online Mendelian Inheritance in Man) is shown, describing the biological & medical significance of what's known about this gene.


human
mouse
fruitfly
yeast
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 X Y
LOC58512 ONTOLOGY SEQUENCE OMIM
search


human
mouse
fruitfly
yeast
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 X Y
LOC58512 ONTOLOGY SEQUENCE OMIM
P36.21
search

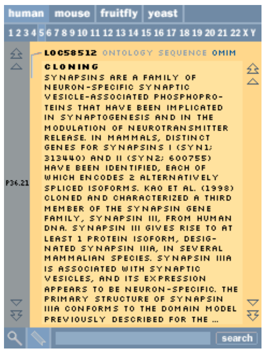
SIMILAR TO PSD-95/SAP90-ASSOCI-
ATED PROTEIN-3 (R. NORVEGICUS)
FROM CLONE 24861

TYPE
GENE WITH PROTEIN PRODUCT
FUNCTION KNOWN OR INFERRED

EVIDENCE
SUPPORTED BY ALIGNMENT WITH BOTH
MRNA AND ESTS

DATABASE LINKS

GENOMIC CONTIG:	NT_004511
MODEL MRNA:	XM_035601
MODEL PROTEIN:	XP_035601
UNIGENE:	HS.145432

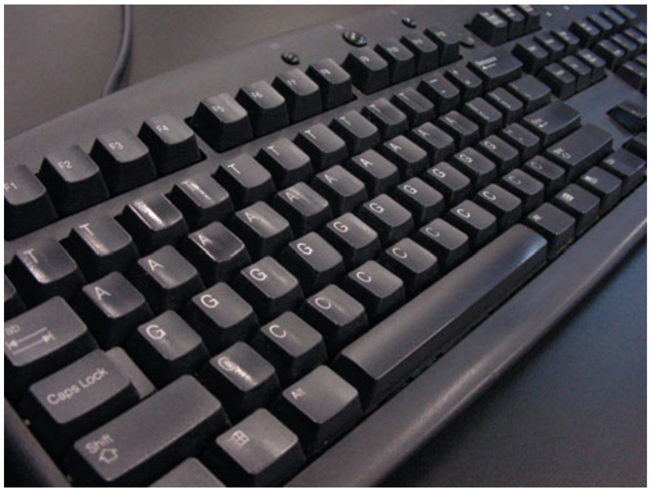

human
mouse
fruitfly
yeast
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 X Y
LOC58512 ONTOLOGY SEQUENCE OMIM
P36.21
search

CLONING
SYNAPSINS ARE A FAMILY OF
NEURON-SPECIFIC SYNAPTIC
VESICLE-ASSOCIATED PHOSPHOPRO-
TEINS THAT HAVE BEEN IMPLICATED
IN SYNAPTOGENESIS AND IN THE
MODULATION OF NEUROTRANSMITTER
RELEASE. IN MAMMALS, DISTINCT
GENES FOR SYNAPSINS I (SYN1;
313440) AND II (SYN2; 600755)
HAVE BEEN IDENTIFIED, EACH OF
WHICH ENCODES 2 ALTERNATIVELY
SPLICED ISOFORMS. KAO ET AL. (1998)
CLONED AND CHARACTERIZED A THIRD
MEMBER OF THE SYNAPSIN GENE
FAMILY, SYNAPSIN III, FROM HUMAN
DNA. SYNAPSIN III GIVES RISE TO AT
LEAST 1 PROTEIN ISOFORM, DESIG-
NATED SYNAPSIN IIIA, IN SEVERAL
MAMMALIAN SPECIES. SYNAPSIN IIIA
IS ASSOCIATED WITH SYNAPTIC
VESICLES, AND ITS EXPRESSION
APPEARS TO BE NEURON-SPECIFIC. THE
PRIMARY STRUCTURE OF SYNAPSIN
IIIA CONFORMS TO THE DOMAIN MODEL
PREVIOUSLY DESCRIBED FOR THE …

7.1.2 *Genome Valence*

With several genome projects nearing states of completion, a primary use of the data for biologists is to search for a sequence of letters and see if it's found in the genome of another organism. If the sequence is found, it is then possible, based on what's known about the sequence as it's found in the other organism, to guess the function of that sequence of letters.

This piece is a visual representation of the algorithm (called BLAST) most commonly used for genome searches. The genome of an organism is made up of thousands of genes (34,000 for the human, 20,000 for the mouse, and 14,000 for the fruitfly). A gene is made up of a sequence of A, C, G and T letters that average one to two thousand letters apiece. In order to handle this amount of information, the BLAST algorithm breaks each sequence of letters into nine letter parts. Every unique nine letter set is represented as a point on screen. The points are arranged from the center, with the most common sets on the outside, the less common towards the middle.



Across the top, a sequence to be searched for is read from an organism. For each set of 9 letters found in the sequence, an arc is drawn between its point in the space and the point representing the next set of nine letters.

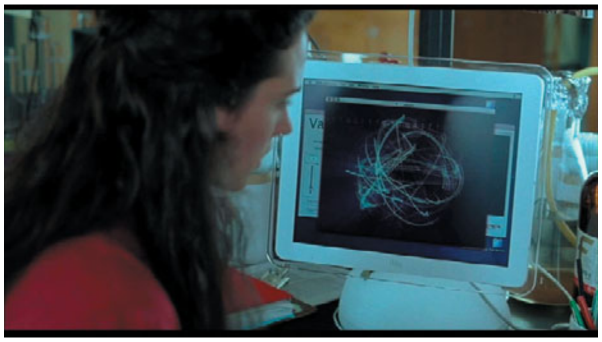
Meanwhile, the same sequence as above can be seen moving through the space as a ribbon of text, wrapping itself between the points that it connects.

For most nine letter sets, there are three points, corresponding to the three organisms and how frequently that set is found in each. The three points are connected by the three lines on each arc, one for each of the organisms being represented. The outer ring is usually the human, the inner is the fruitfly.

Developed specifically for this project, the ACGT keyboard makes it possible to input a sequence for a search. The keyboard is intended as part joke, part practical means to allow exhibition visitors to be able to hit any key (this is the tendency of the public) and have something happen. The result is another ribbon that weaves through the space to highlight the sequence of selected letters.

Genome Valence originated with the Valence project from my master's thesis [Fry, 2000], and was developed as part of the request for its inclusion in the Whitney Biennial in 2002.

It also made a brief appearance in in Ang Lee's *Hulk*, in 2003, finding use in the laboratory of lead actress Jennifer Connelly.

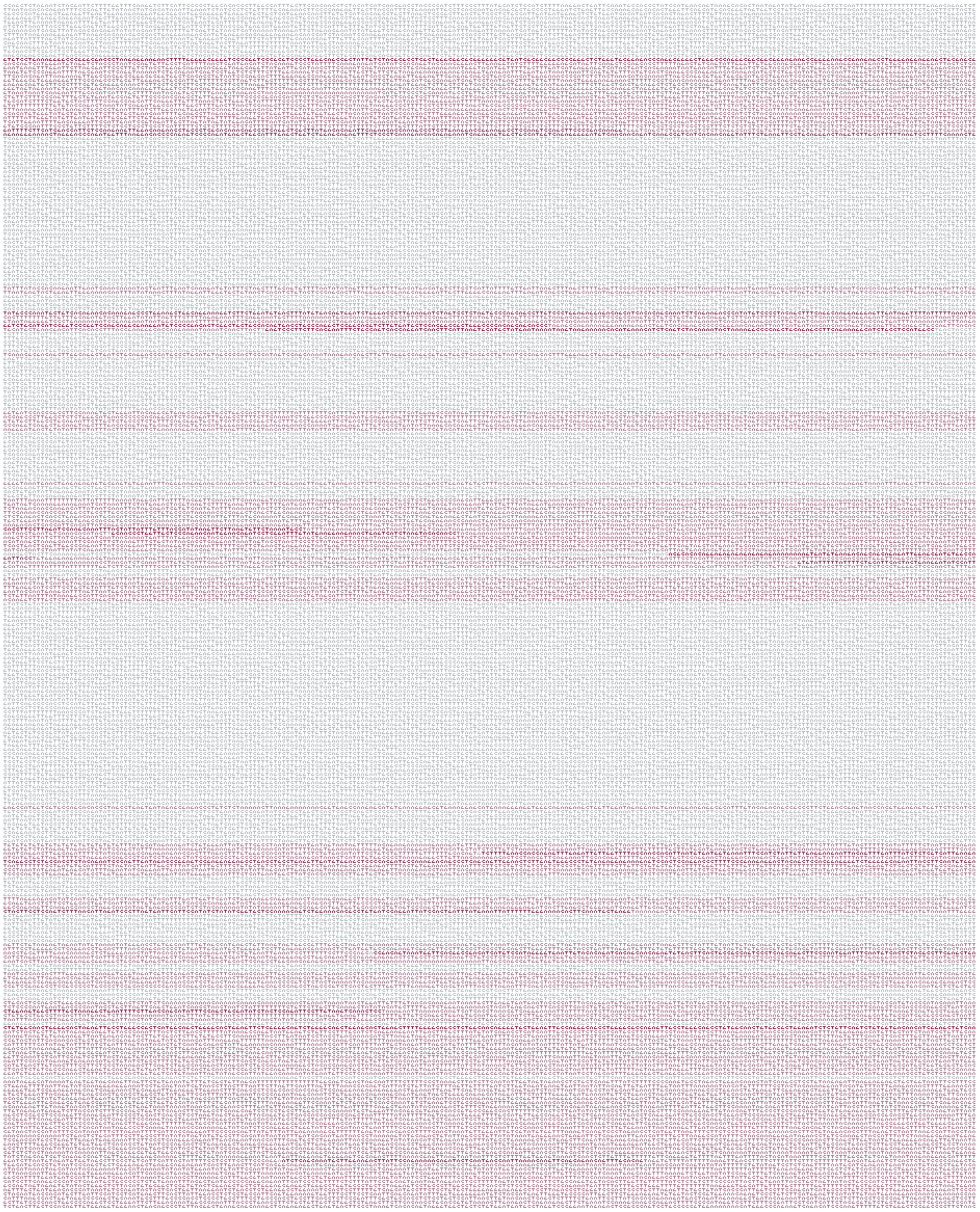


7.1.3 Chromosome 21

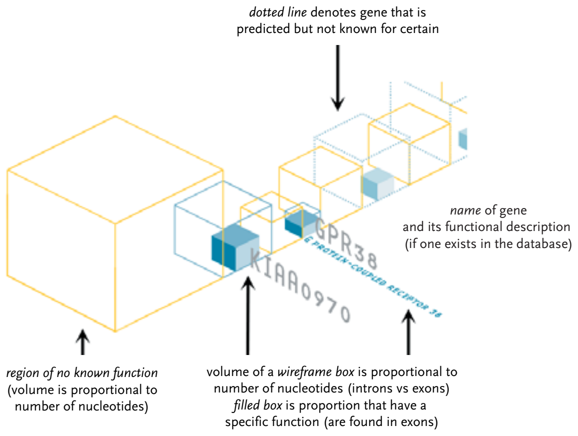
A study in the scale of the human genome, the image below depicts thirteen million letters of genetic code from chromosome 21. Similar to the image a gene in chapter four, the darker color depicts sequences of code (exons) that are known to be used by a cell as the set of instructions for building a protein. These instructions are interrupted by unused pieces of code (introns) which here have a medium coloring. The gray areas that might be regulatory region, or simply have no currently known function.

The image below was part of the exhibition "How Human" at the International Center for Photography in New York City. The size of the installation was eight feet square, and used a specially designed font that occupies only three pixels in either direction, with one pixel of space between each letter. The image is printed at 150 pixels per inch, meaning 37.5 letters to the inch in either direction (same resolution as small subset of the image at right).

Chromosome 21 is one of the shortest human chromosomes, yet this image is only one quarter of the roughly 50 million letters of which it is composed. It would take 250 images of this size to depict the genetic code in the entire human genome, which is 3.1 billion letters.



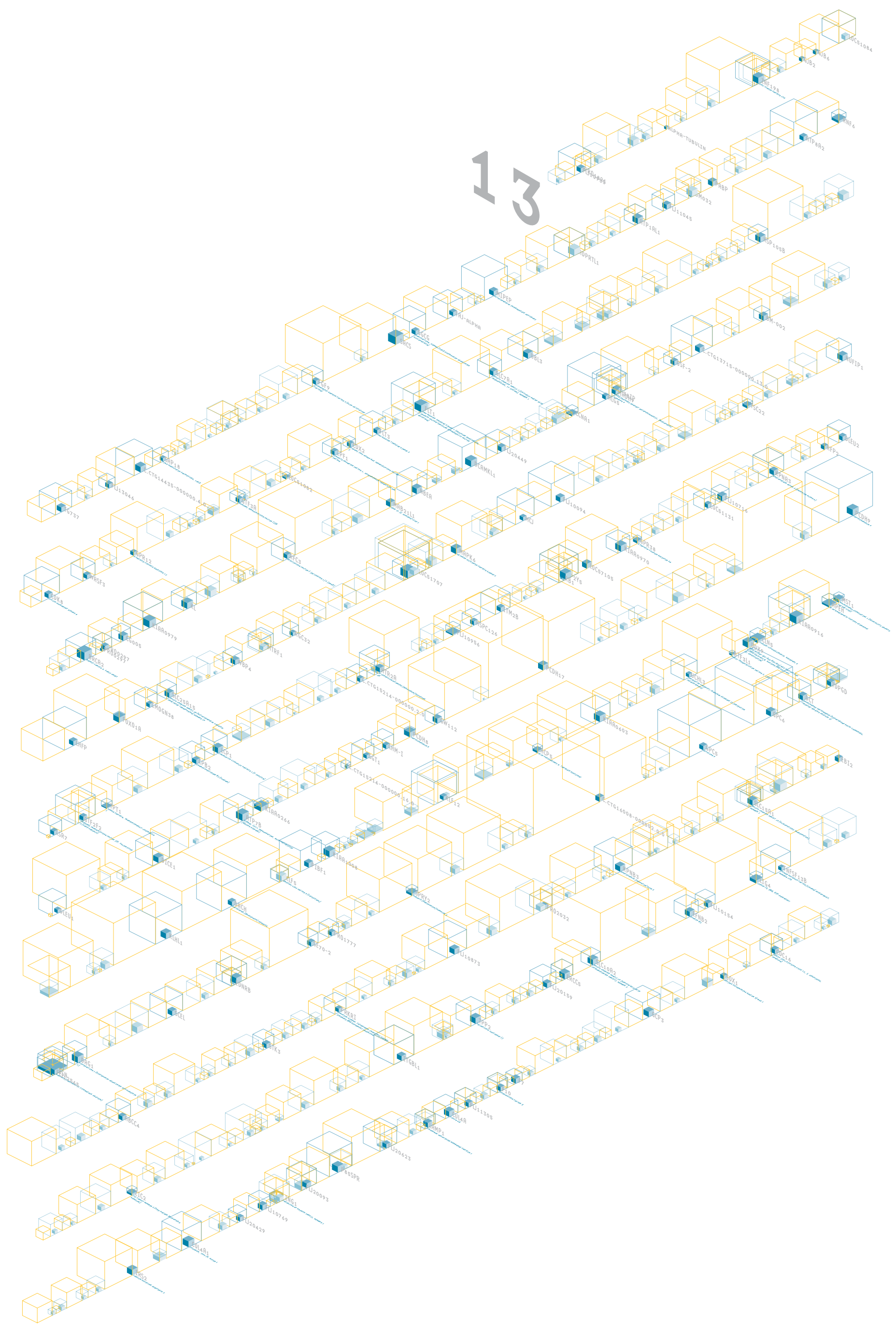
7.1.4 Axonometric Introns & Exons

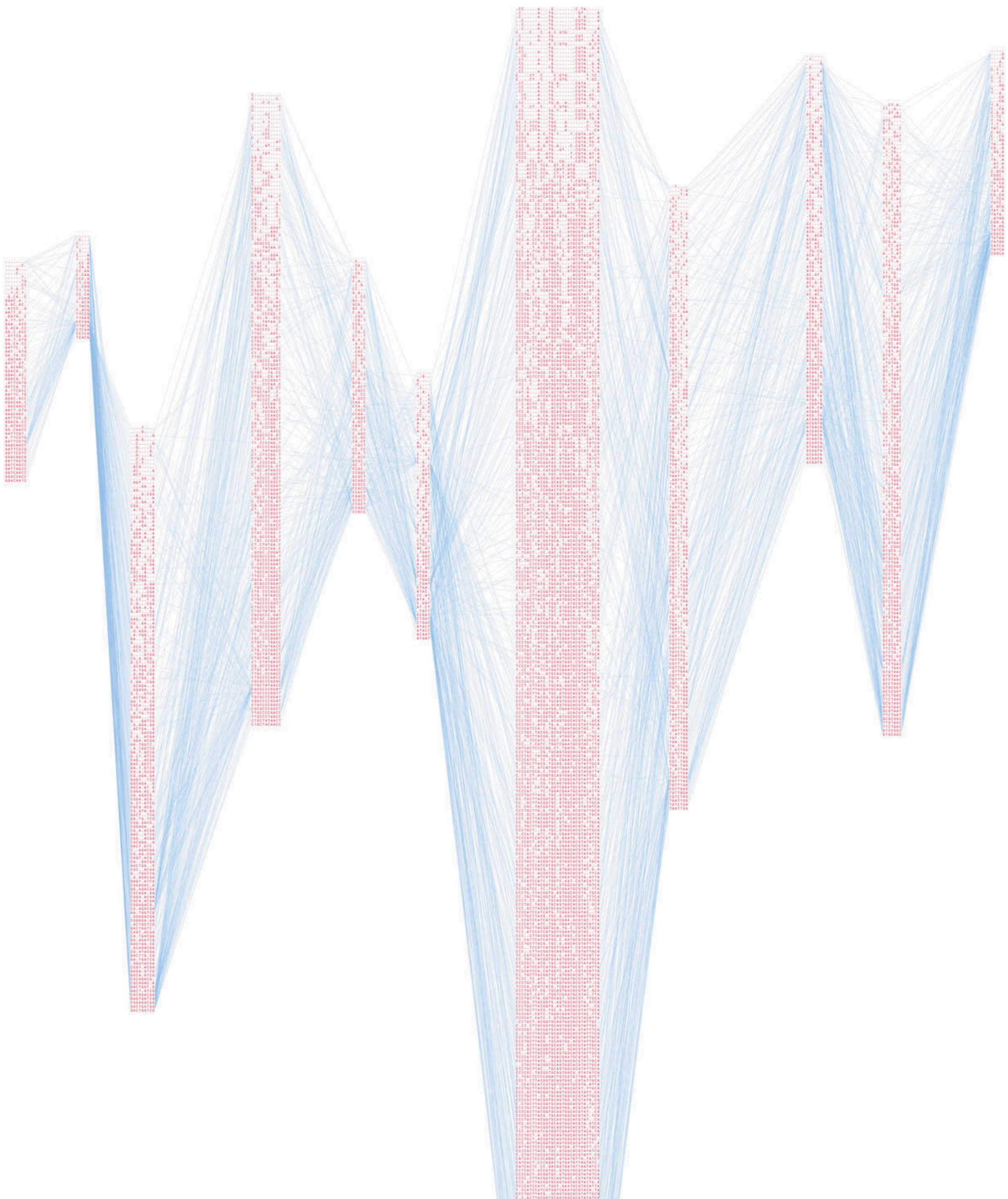

dotted line denotes gene that is predicted but not known for certain
GPR38
G PROTEIN-COUPLED RECEPTOR 38
KIAA0970
name of gene and its functional description (if one exists in the database)
region of no known function (volume is proportional to number of nucleotides)
volume of a *wireframe box* is proportional to number of nucleotides (introns vs exons) *filled box* is proportion that have a specific function (are found in exons)

The human genome is believed to be made up of approximately 35,000 genes, each of which have a specific task in cells of the human body. This image depicts two-thirds of those genes, the ones publicly catalogued by scientists by the middle of 2002. The image at the lower left is a depiction of this data, two stories tall (9 feet wide and 18 feet tall) at the International Center for Photography in 2003. The output from the original version of this software, which was for a single chromosome, is shown on the opposite page. Because the visualization is software-based, it is possible to change the underlying data to the entire genome without the need for a complete rewrite.

Genes known to exist are shown as blue wireframe boxes, with their name name and a description. Genes thought to hypothetically exist have a dotted blue wireframe. The size of the box is proportional to the amount of genetic code (A, C, G, and T letters) that make up the gene. The solid blue boxes inside the wireframes depict the proportion of code that is actually used by the gene. The yellow boxes show the amount of material between genes that has no known function.

The in-use material is most important, but there is far less in proportion to the unused data. To help even the balance, a semi-3D layout is used because the proportions are more easily compared as a three dimensional volume rather than as a horizontal sequence. Eight letters of in-use material can be shown with a 2 × 2 × 2 box. A set of 27 unused letters are shown 3 × 3 × 3. The volumes maintain the true proportion, but allow a simpler side-by-side comparison of 2:3 instead of 8:27.


13



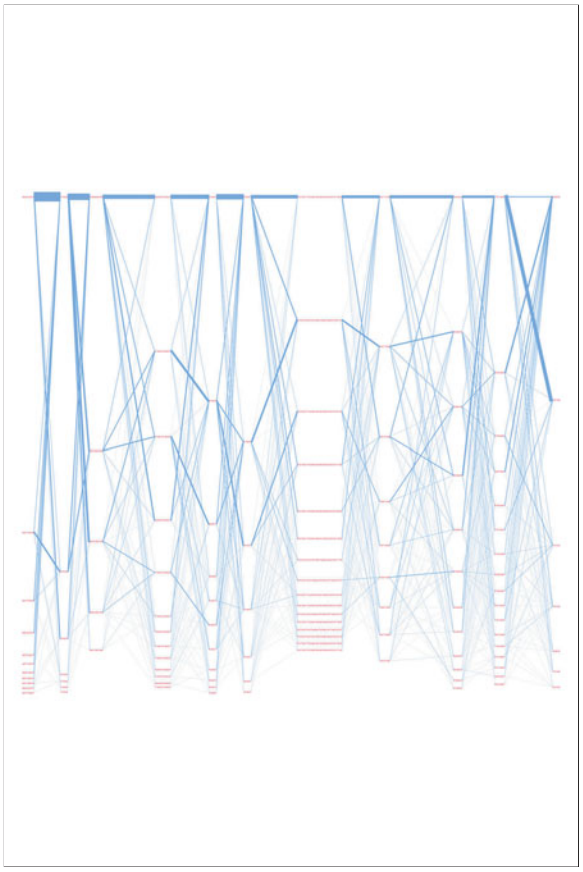
7.1.5 *Haplotype Lines*

This project preceded the work in understanding haplotype data found in chapter four. These were some of the studies that led up to the final solution.

The earlier representation connects each line of variation to the other lines. The result is the heavily intertwined form of hairlines, with the variations listed in order of relevance, with more relevant data found towards the bottom (the denser sets of letters). While an interesting first attempt, the image isn't particularly useful in its representation.

The program takes a set of data as input, and can either run as a java applet to show the representation in a web browser, or output it to postscript format for high-resolution rendering.

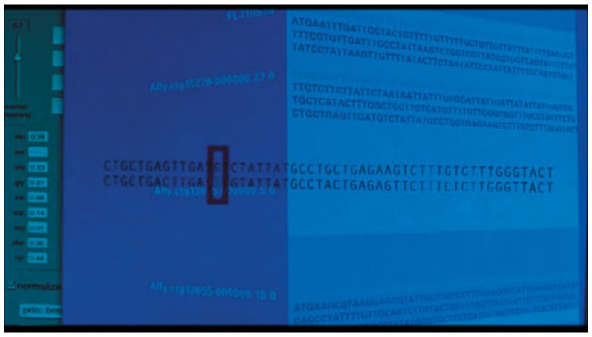
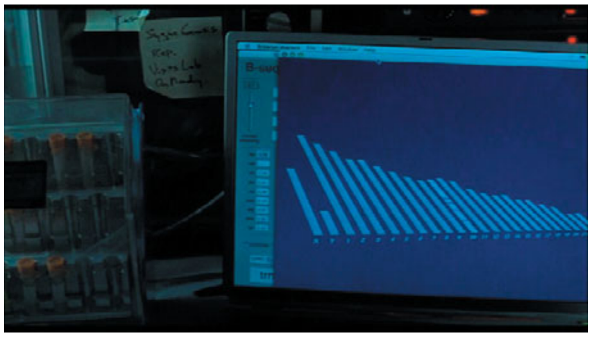
The representation inset on this page further condenses the sets of variations, and modifies line weight based on how often the sequences are found adjacent one another. With this representation, it is possible to see major trends. The most common variations are shown with thicker lines, and are sorted top to bottom based on their thickness.



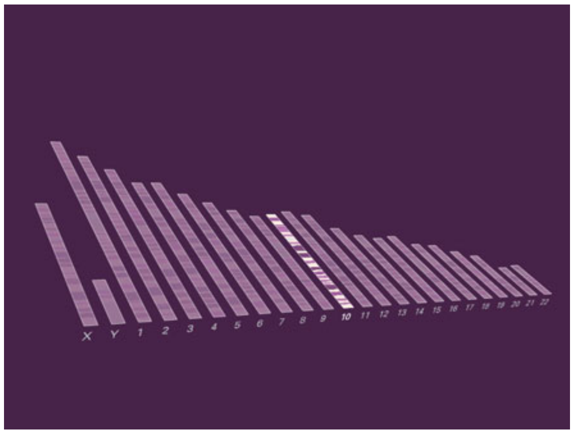
7.1.6 *Strippy*

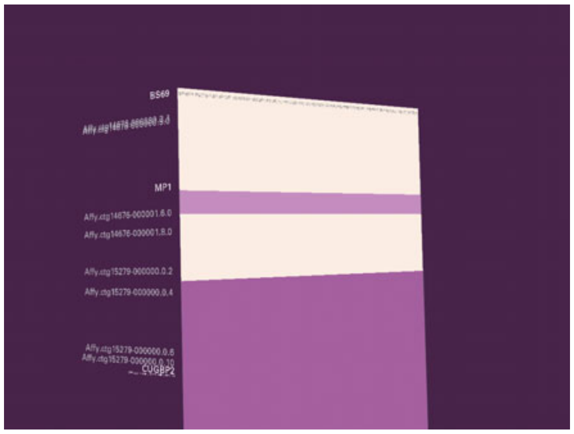
A three dimensional genome browser from spring 2002. It was an attempt at a type of "powers of ten" view of the genome, but I didn't like where it was going, and discontinued the project. It also never got out of the ugly stage before I dropped it. It was a full browser, however, and all the data shown is real (those are actual genes from chromosome 10 of the human genome).

As it happens, a powers of ten style visualization is problematic because of the scale of data being considered. Given an overview of the chromosomes in the human genome—on average representing 100 million letters apiece, the next most useful step will be a gene on the chromosome, each of which average at around a thousand letters. This difference of 100,000 : 1 means that several orders of magnitude must be animated through before reaching the next most useful point in the visualization. This can be done through a modification in the timing, yet even if the issues of lost context can be worked out, it's not clear whether this is even a useful mode of representation in such a case.

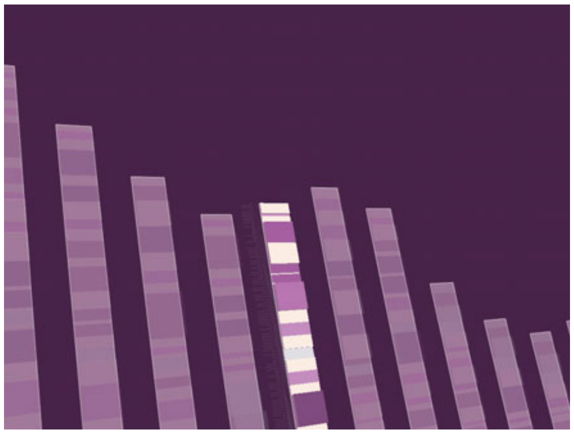



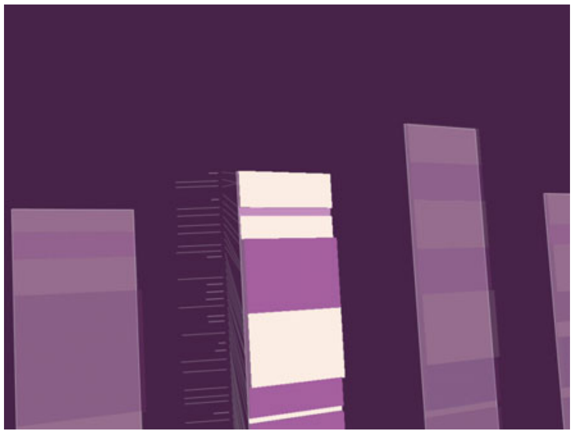
This project makes a brief appearance in *The Hulk* thanks to John Underkoffler, the film's Science and Technology Advisor. Nick Nolte's character uses it for analysis in the earlier part of the film. John is also responsible for the project's name.


X
Y
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22


BS69
MP1
Affy.ctg14676-000001.6.0
Affy.ctg14676-000001.8.0
Affy.ctg15279-000000.0.2
Affy.ctg15279-000000.0.4
Affy.ctg15279-000000.0.6
Affy.ctg15279-000000.0.10
CUGBP2

g15279-000000.11.0
FLJ10578
LOC57118
HsMCM10
g15279-000000.27.0
LOC51182
FLJ11106
ctg13655-000000





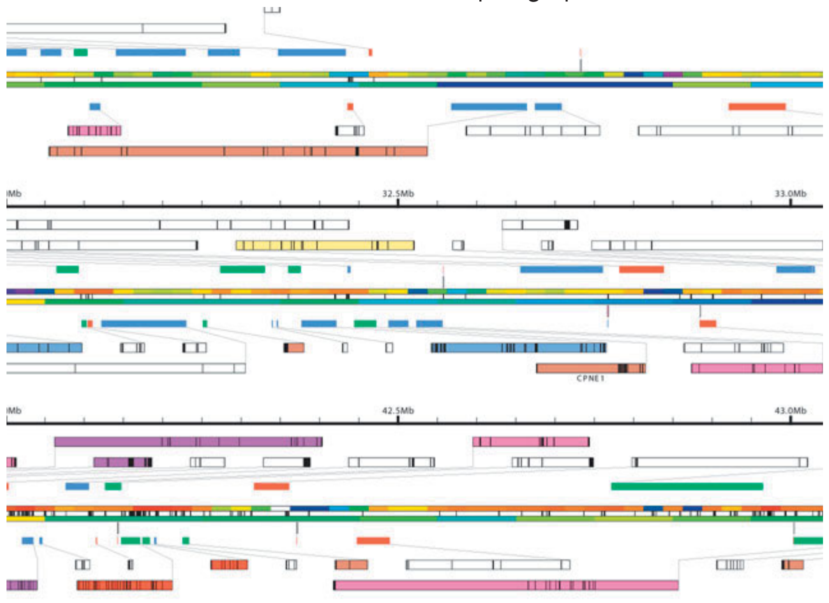
578
118
10
27.0
MNLIATVFVFAVVIYFQGFRVDLPIKSARYRGQYSSYPIKLFYTSNIPIILQ
MDEEEDNLSLLTALLEENESALDCNSEENNFLTRENGEPDAFDELFDAD

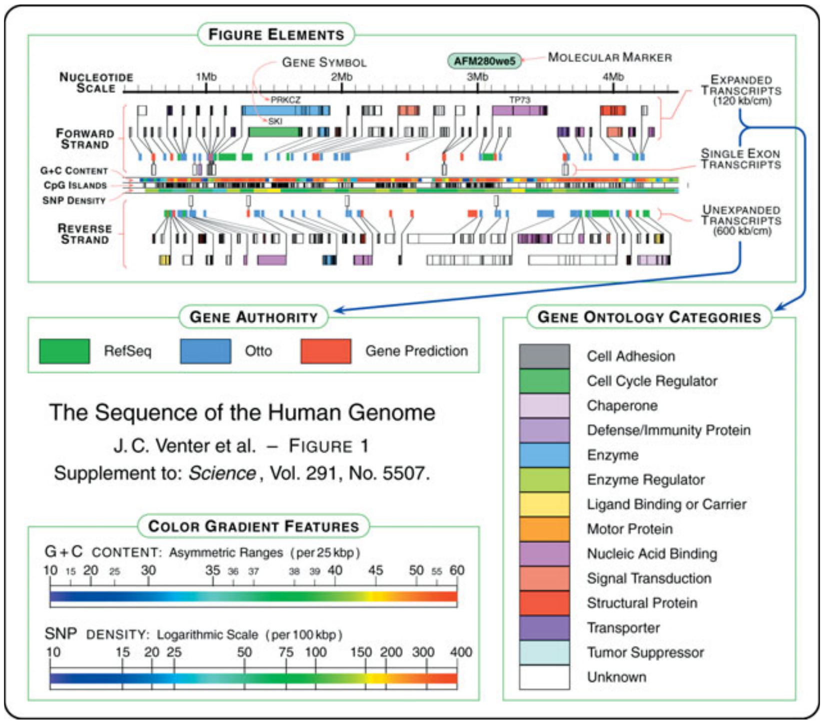
7.1.7 *Redesign of* GFF2PS

The visualization of genetic sequence data is primarily a representation problem. As one of the previously mentioned goals of in information design, diagrams should clearly depict the order of importance of the data they represent. This helps the viewer know what aspects of the diagram deserve the most attention, and by giving prominence to the most important items, those features can be read "at a glance."

The application GFF2PS [Abril and Guigó, 2000] has been used to produce images of the human [Venter *et al*, 2001], fruitfly [Adams *et al*, 2000] and mosquito [Holt *et al*, 2002] genomes for publication. The program reads a generic data file that lists a series of "features," essentially a short annotation (text or numeric) and the start and stop position of where that feature is found. A section of the resulting image from a recent publication is shown here:

Genomic sequence data is well suited to such a "track" oriented annotation format, but a tendency exists, when all the data is treated in such a generic manner, that the resulting diagram lacks a clear hierarchy of visual importance, and each track is generally given equal area on the page, because the image is driven more by how the data is stored than by considering the relative usefulness of each track to one another—it's a bulleted list of items rather than a clear paragraph of text narrative.



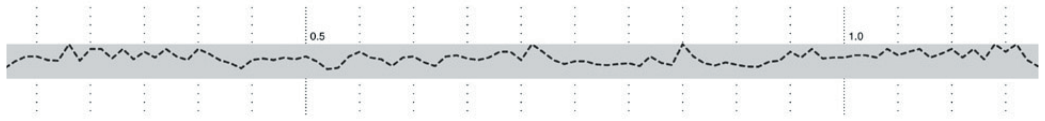


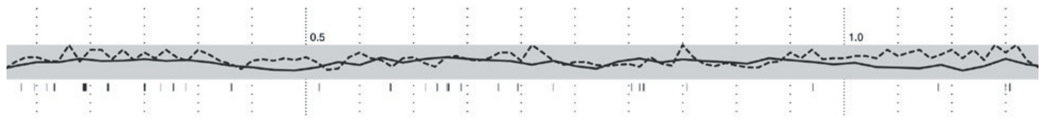
The solution is to first consider the order of importance for the data in question, and then construct a diagram that places greater emphasis on those elements, while diminishing the visual importance of "background" information like the diagram's scale, or features of lesser value like the database where each gene was found.

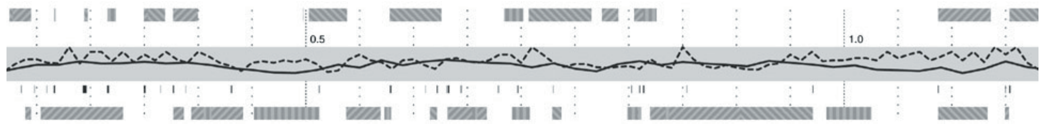
A new design, which includes all the features from the diagram above, is developed in several steps on the following page. The first step of the design begins with just a thick horizontal bar that provides a center point for the diagram, which gives the eye a place to "begin" each line of sequence when several lines are shown next to one another in a publication. Behind that, the nucleotide scale (the number of letters of genetic code) is shown using simple gridlines and small pieces of text. The scale has been moved to the middle of the diagram, rather than the top, also as a way to center the diagram, and because the features above and below are not shown on the same scale.

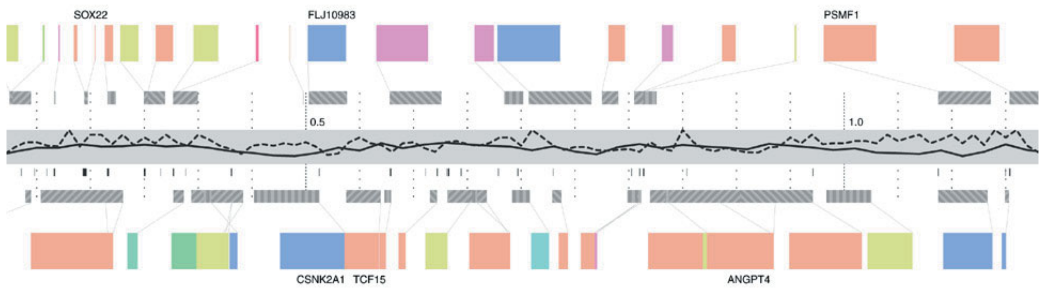
In the second step, a line graph representing SNP density is added. A line graph is appropriate because when reading this feature, the viewer's primary interest is in the relative (rather than an absolute, numeric value) or trends (upward or downward) in the data.

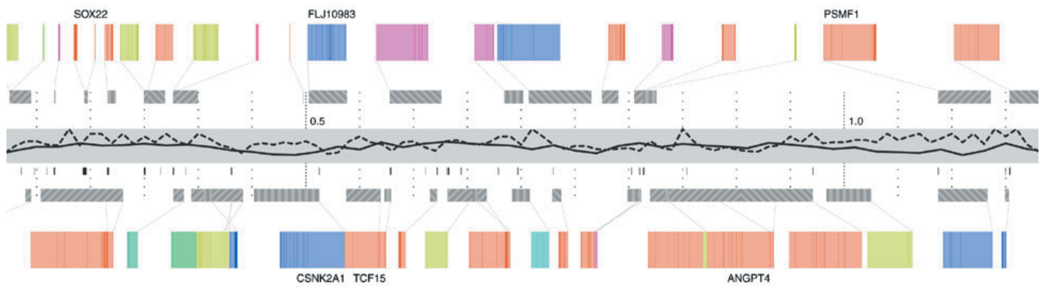
0.5
1.0


0.5
1.0


0.5
1.0


0.5
1.0


SOX22
FLJ10983
PSMF1
0.5
1.0
CSNK2A1
TCF15
ANGPT4


SOX22
FLJ10983
PSMF1
0.5
1.0
CSNK2A1
TCF15
ANGPT4

Step three adds G+C content as a second, solid-colored line graph, for the same reasons as SNP density. Instances of a related feature, CpG islands (often used to predict the location of genes), are shown just below, using the same color as the line itself. CpG islands are a small tickmark, since their exact location and length is less important than simpy seeing groups of them that are clustered together.

The next step shows the genes, above and below the center track, based on whether they're read by the cell's machinery in the forward or reverse direction (identical to the GFF2PS layout). One of three line patterns is drawn on top the genes, relating to the "gene authority," or the database where information about the gene originated. This item of information is of extremely low importance, which is why the pattern is so low contrast and given such little visual priority.

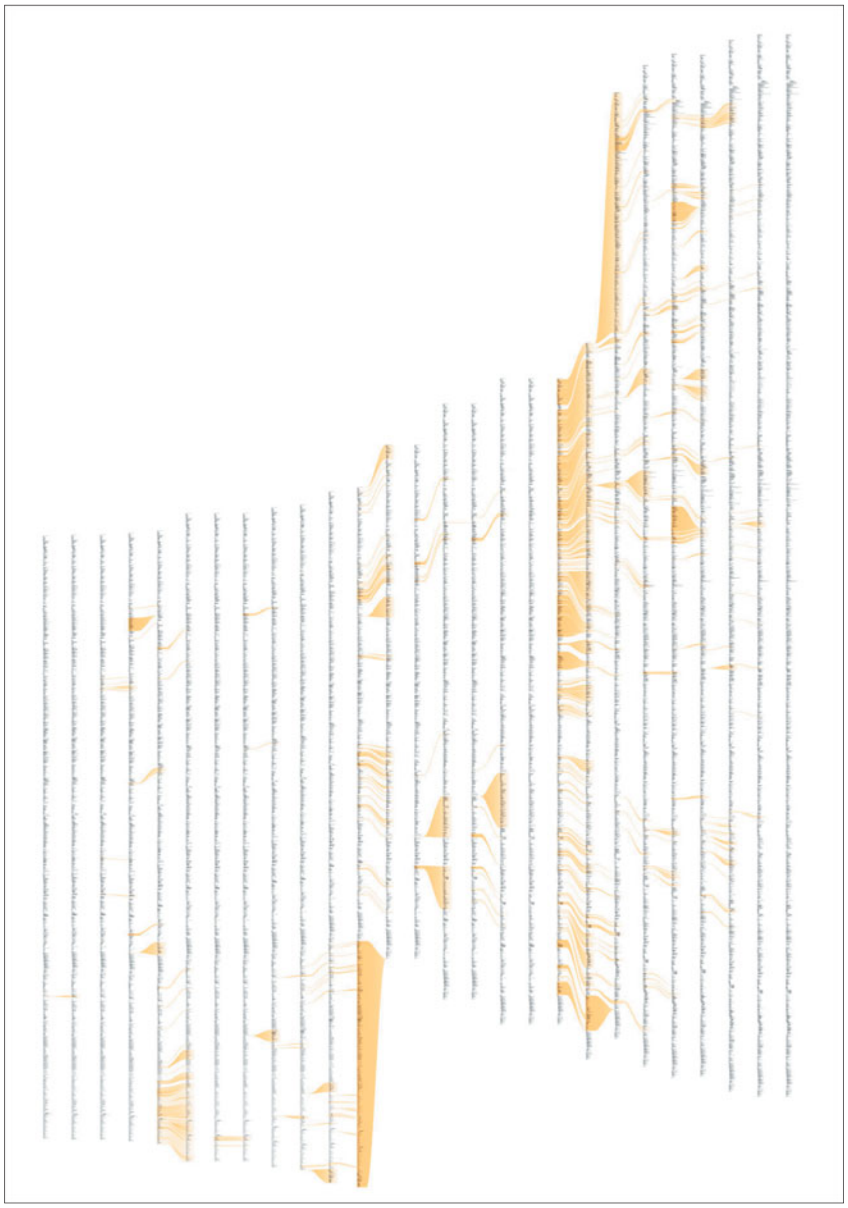
The final step adds an 'expanded' gene track (again, in the manner of the GFF2PS diagram) showing a series of filled rectangles, each one of five colors for the gene's ontology category. In keeping with a general rule of using no more than five colors, the ontology categories have been reduced to five, with the hope that even five more general categories will be more informative than twenty poorly distinguished ones.

It is not presented as the best way to show sequence data, but rather provides a useful example of how to address the issue of showing many varieties of data together. In addition, this diagram is considered as it has appeared on multiple occasions in high-profile publications. To properly re-address this diagram from the ground up, it would be necessary to consider more fundamental questions such as "who is the audience?" and "in what context will this diagram be used?" Answering such questions will help better define what data should be included on such a diagram, and how much importance should be given to its constituent tracks.

7.2 VISUALLY DECONSTRUCTING CODE

These projects were created as part of a series shown in the Ars Electronica 2003 CODE Exhibition. It consists of short (two days to one week) sketch projects that examine ways of looking at software code.

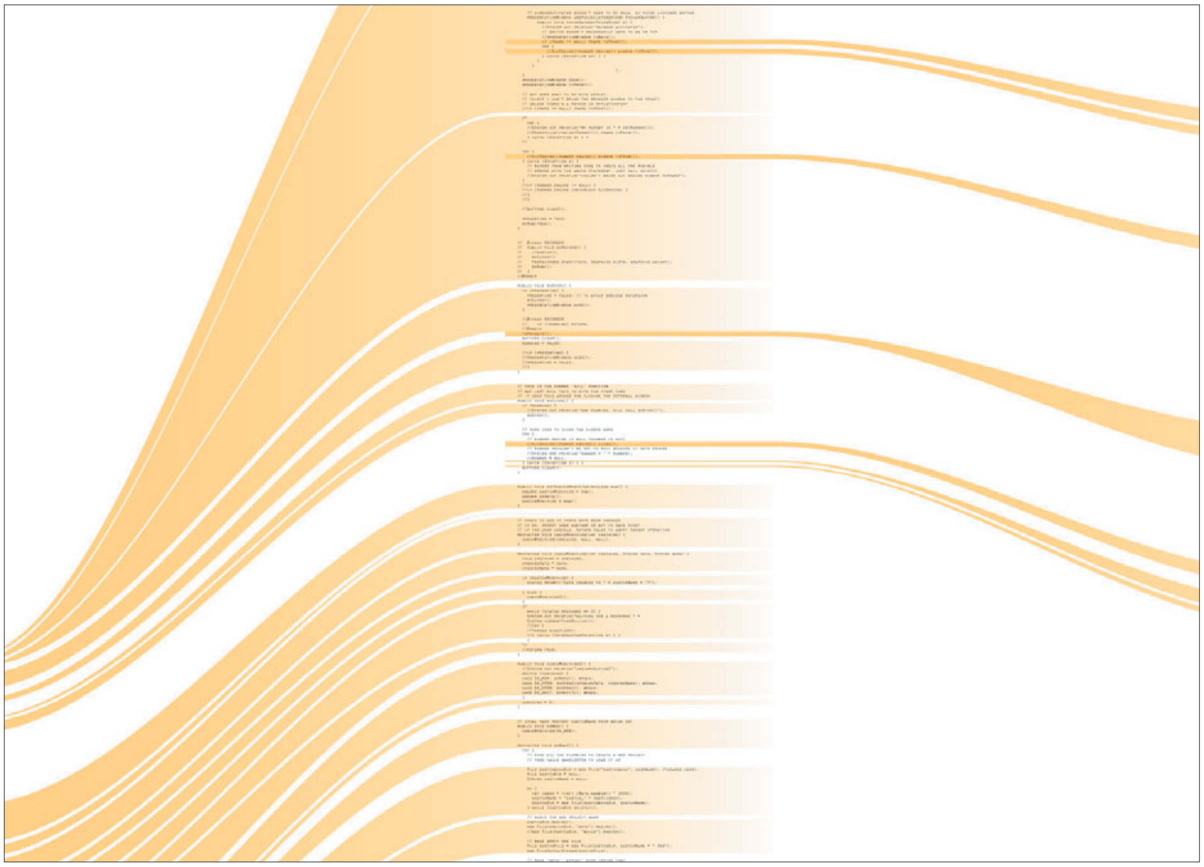
7.2.1 Revisionist



While it's unsurprising that the code in a software project changes over time, less obvious is the nature of how the changes have taken place in a broader sense. Projects are typically structured as a collection of files that are added, removed, and reorganized throughout the course of development. The contents of the individual files are modified, line by line or in large pieces for every fix and feature.

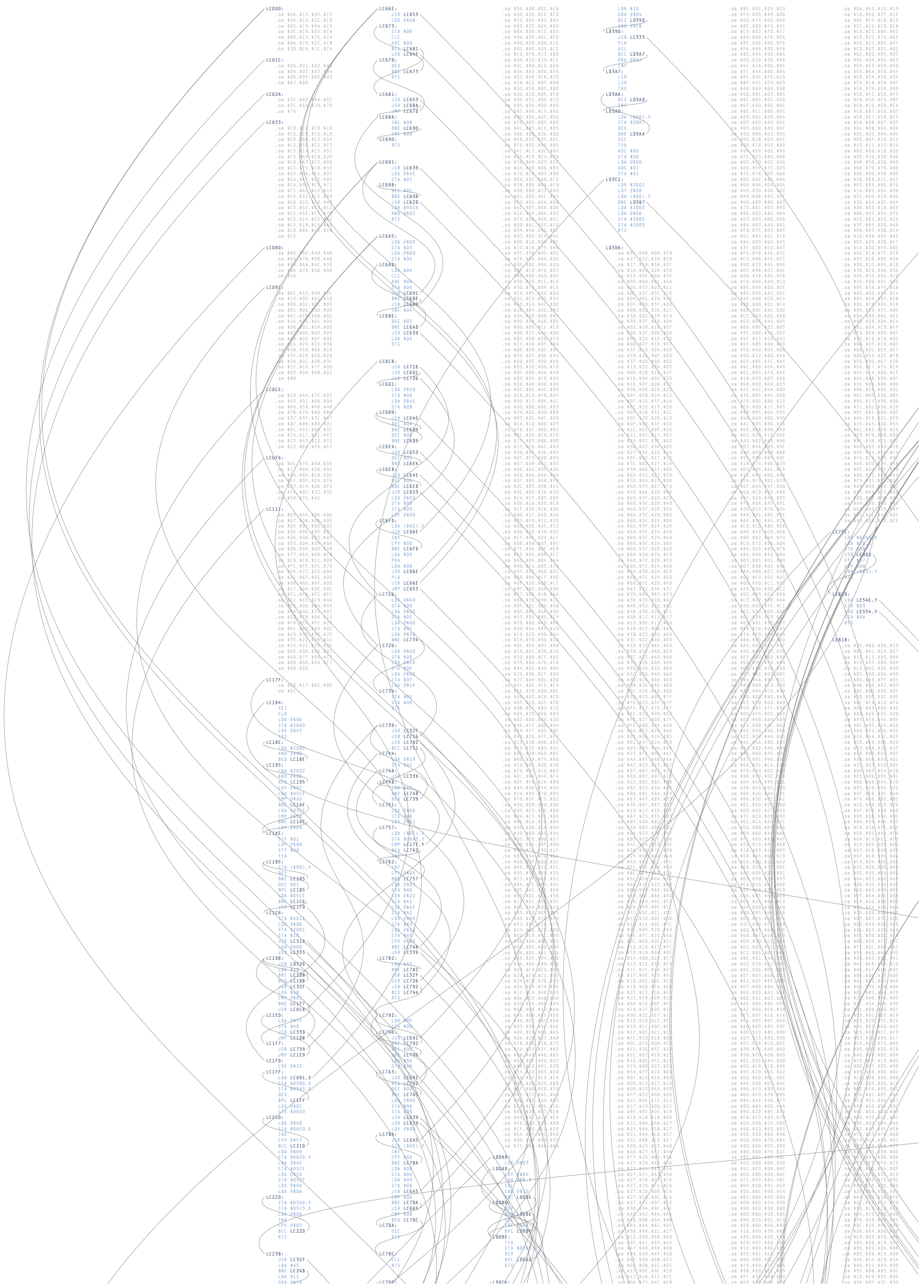
(Note that the actual prints of these projects are 32 inches wide and 50 inches tall, so this small page format is a less than ideal format, and contradicts the goal of seeing all the changes at once).

The revisionist software generates an image that shows the evolution of the structure and content of the *Processing* project over time, from its initial inception through forty



releases. The experiment consists of large-format printed pieces to depict broader changes over time. One image shows an overall picture of the changes, with each column representing one version of the software. Lines are drawn between the columns to connect lines that have changed. Notable changes can be seen for the removal of one large file (bottom, middle), and the later addition of a new file (top, right), or a spike in activity (about two-thirds to the right) coinciding with a push for the initial alpha release of the project. The second image shows a detail of the image, zoomed 13x to make the text of one portion of a column legible.

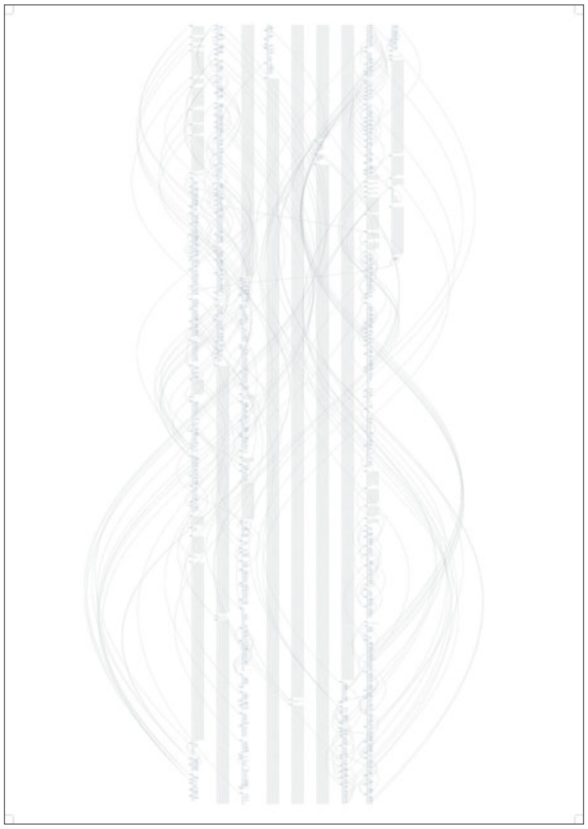
While the method of depicting changes between versions of a file is not new, the representation of many versions in a single instance is less conventional. The result is a depiction of the organic process in which even the smallest pieces of software code become mature through the course of its development, as they are passed between developers, revisited for later refinement, merged, removed, and simplified.

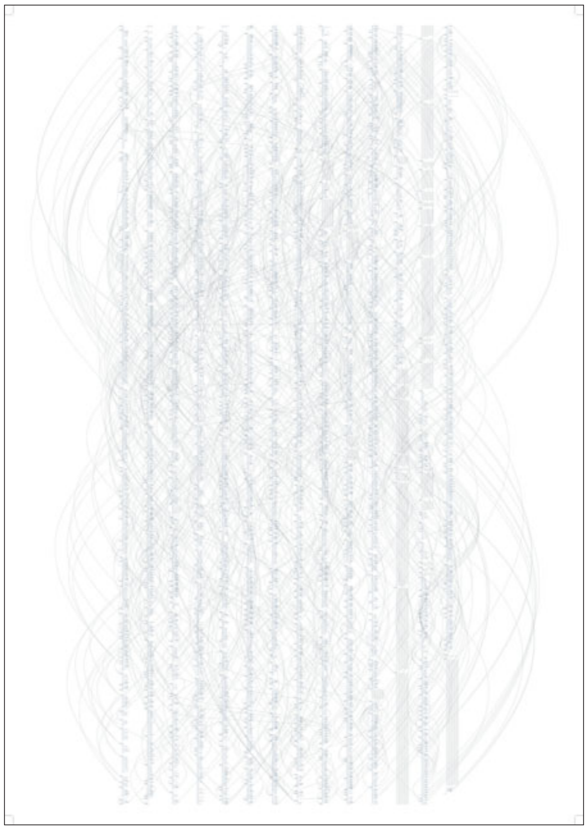


7.2.2 *Dismap*

Programs on modern computers are eventually compiled into machine language, a series of basic and direct instructions understood by the microprocessor. Programs are flattened from their hierarchical state into a long series of simple mathematical instructions (like multiply or add with carry) and interspersed with commands for jumping to another location in the program.

The simpler of these two diagrams shows the program for the "Excite Bike" game for the original Nintendo (this console was chosen for its simplicity). The blocks of gray text are "data" sections that are used to store images or game scenarios. The curved lines connect locations in the program where "jumps" occur, which can be a function or a conditional choice made by the software. The more complicated image depicts the original "Super Mario Brothers" game. The images were created in appreciation of the elegance in the structure of such software, not so much as a diagnostic tool for understanding their operation.

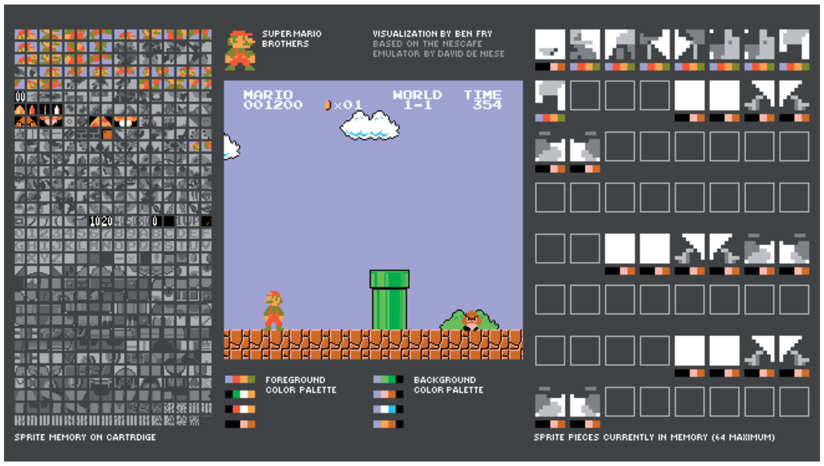




7.2.3 Deconstructulator

This is a deconstructed Nintendo emulator that shows how sprites and sprite memory are handled while a game is being played. The intent is to show insight for how software and hardware work, given the relatively simple example of a minimal architecture from an old game console system.

The emulator is a modified version of the NESCafe emulator written by David de Niese. David was kind enough to make the source code of his emulator available, which I hacked up a bit to dynamically show aspects of how the machine works.



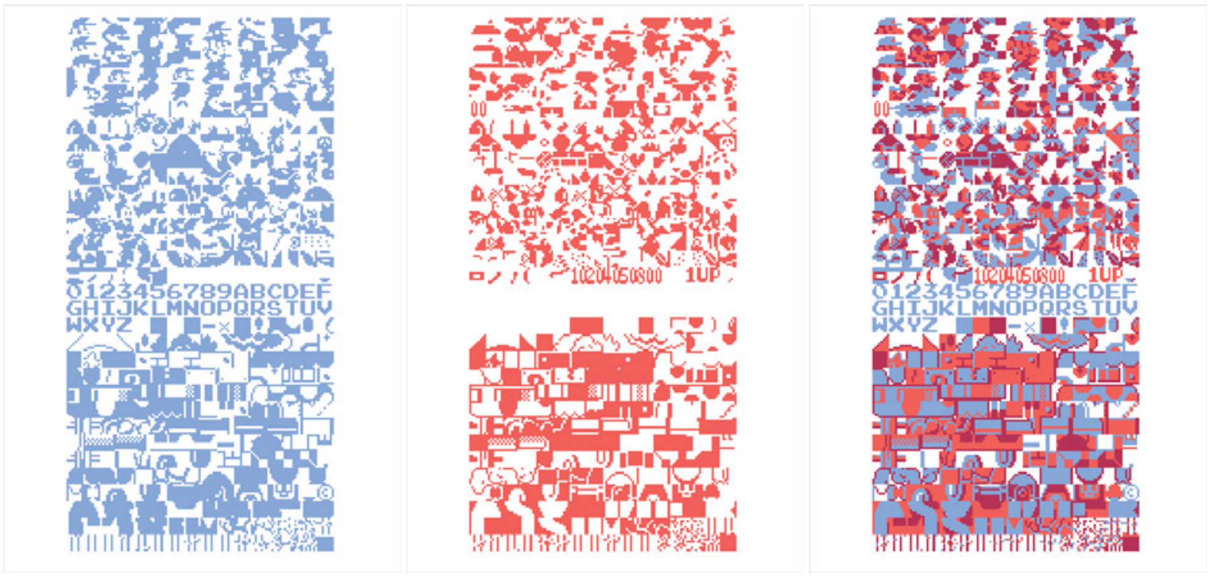
On the left is the sprite memory on the cartridge, a bank of 8x8 pixel tiles that are reassembled to create the images used in the game. Check out mariosoup for more information on how they work. The images are stored as four colors, but the colors are only specified as the program is run. While playing a game, the tiles are colored based on the last color set used to draw that image.

Colors are used in sets of four, of which there are four available for the background, and four more assigned to the foreground. The sets are shown just below the game image.

On the right are the 64 sprites (8x8 pixel tiles) currently in memory. Beneath each is the four-color set that has been applied to it as it was added to the screen. Only 64 can be active on the screen at any given time. In the case of Mario, he's assembled from several small tiles which can be seen towards the top.

7.2.4 Mariosoup

Any piece of executable code is also commingled with data, ranging from simple sentences of text for error messages to entire sets of graphics for the application. In older cartridge-based console games, the images for each of the small on-screen images (the “sprites”) were often stored as raw data embedded after the actual program’s instructions. This piece examines the unpacking of a Nintendo game cartridge, decoding the program as a four-color image, revealing a beautiful soup of the thousands of individual elements that make up the game screen.



The images are a long series of 8x8 pixel “tiles”. Looking at the cartridge memory directly (with a black pixel for an “on” bit, and a white pixel for an “off”) reveals the sequence of black and white (one bit) 8x8 images. Each pair of images is mixed together to produce a two bit (four-color) image. The blue represents the first sequence of image data, the red layer is the second set of data that is read, and seeing them together produces the proper mixed-color image depicting the actual image data.

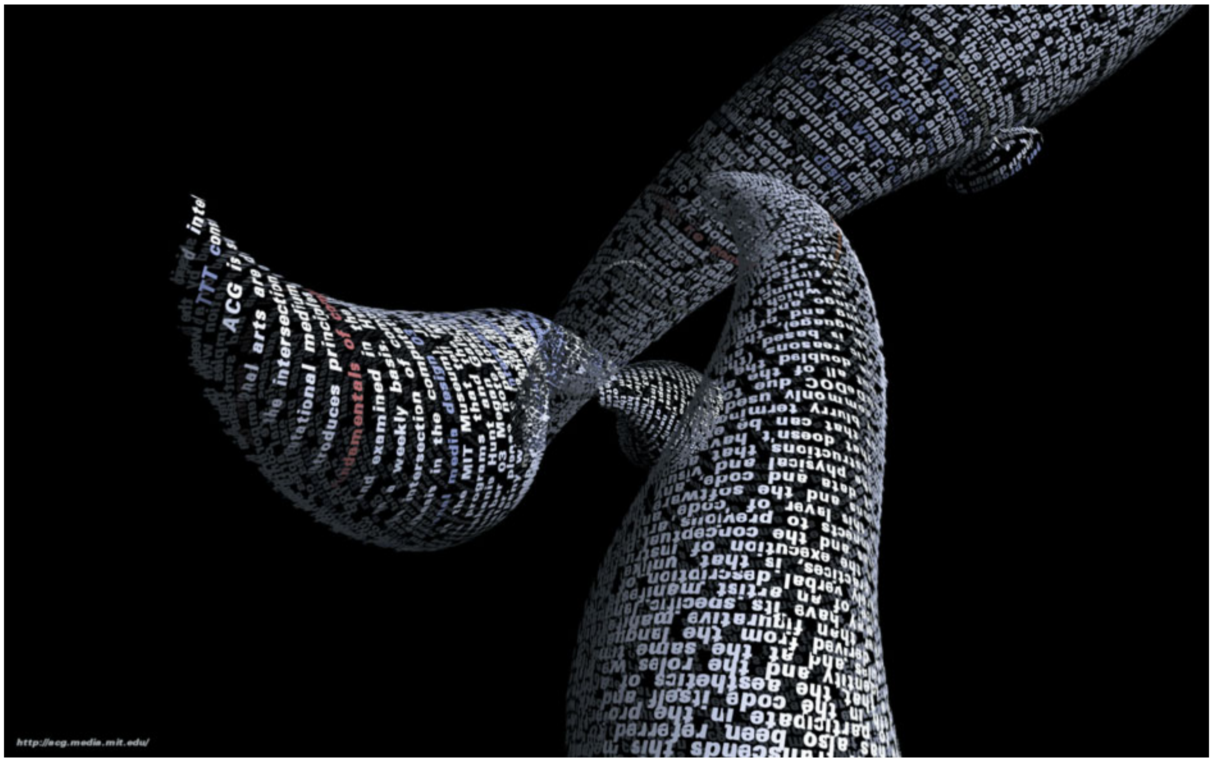
7.3 TENDRIL

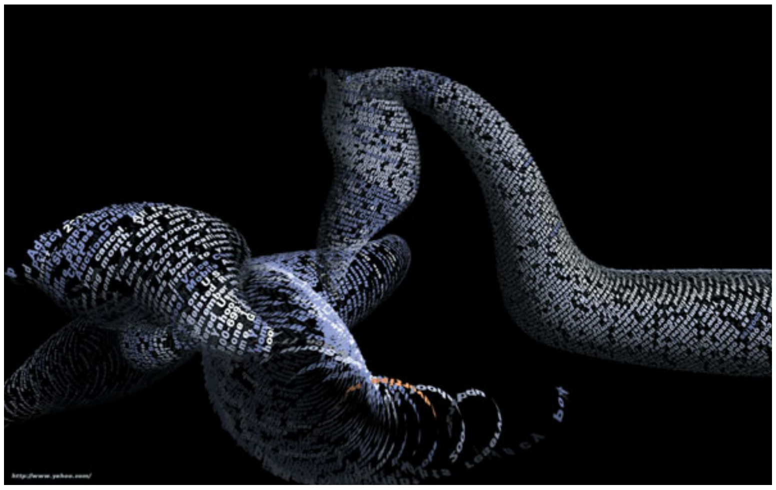
Tendril is a web browser that creates typographic sculptures from the content of web sites. Branches are formed from the text of a web page, each link on the page begins another branch for the linked page.

For instance, the first page of a site is rendered as a column of text. links in the text are colored, and when clicked, the text for the linked page grows from the location of the link.

The visualization is controlled by trackball input, allowing the user to rotate and zoom into and around the structure.

Over time, the result is an enormous branching structure, built purely out of the text that is contained in a set of connected web pages, and guided by the hand of the user as they choose the direction of links that are followed. The project was created as a means to bring form and structure to a large set of raw data, streamed live from a server, that is undergoing continuous change.




http://www.yahoo.com/

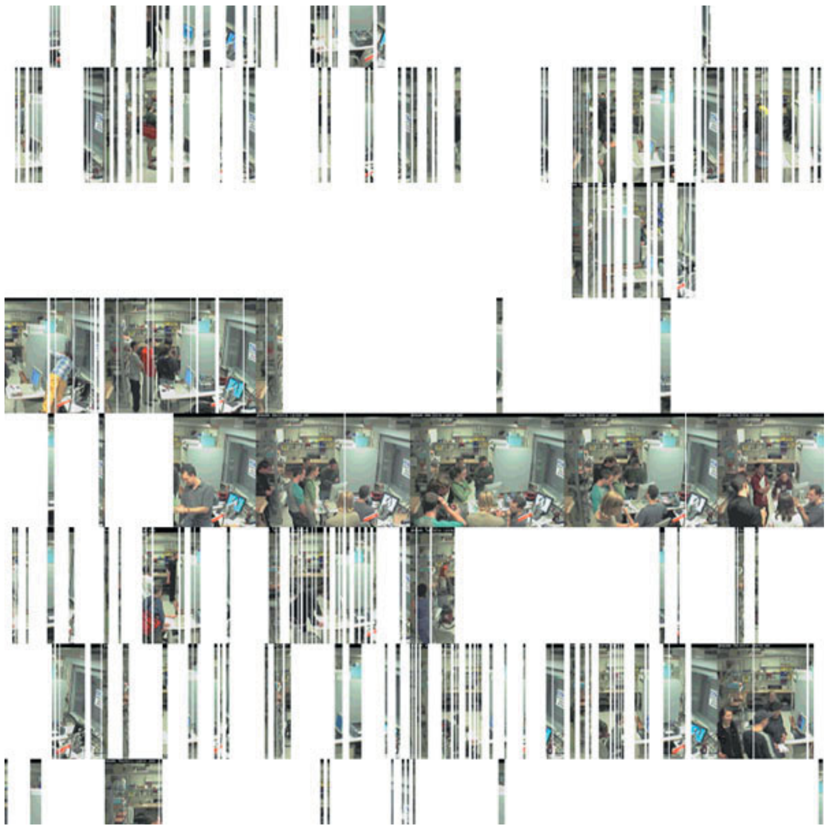
http://www.yahoo.com/

http://acg.media.mit.edu/

7.4 SHOPACTIVITY

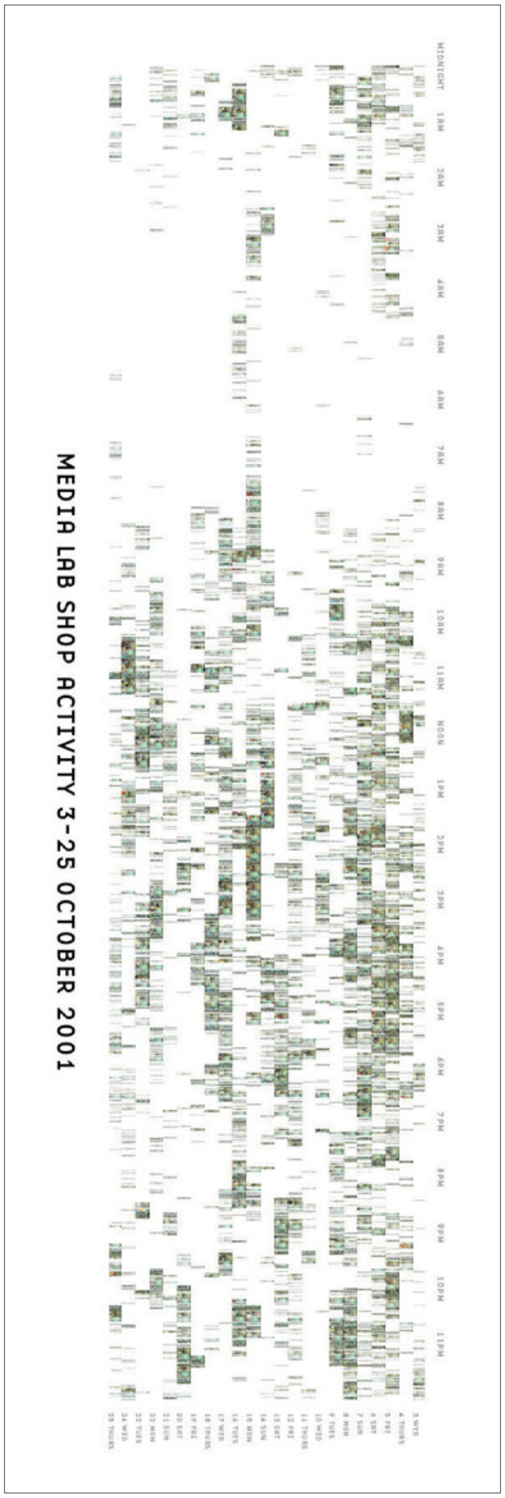
This image depicts activity in the MIT Media Lab's shop for two and a half weeks. Lines depict activity detected on the security camera. With enough activity, the lines coalesce to reveal the image of the people working at that time. It was created with two short programs, one in Java and another in Perl. While this version used just two weeks, it would be possible to run the programs on a larger set of data, or instead run on screen as a dynamic display that shows new images as they are added. A detail is shown below.

This project was an early example of sketching with software: how to explore a set of data by quickly developing a visual image. part of a set of experiments to create complex and sophisticated information graphics very rapidly (a matter of hours) through the use of short scripts or simple programs. In pursuing such work, a library of useful functionality for sketching data visualization has been created, and influenced the development of *Processing*.



The image here is rotated to the right because of the space and layout constraints for this document.

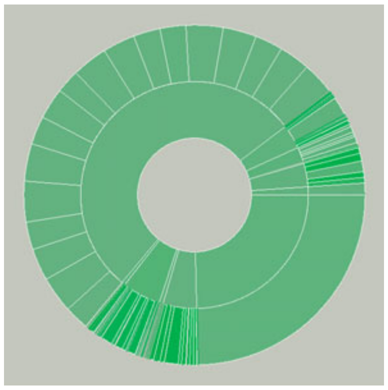
Actual size of the image is 72 × 24 inches.

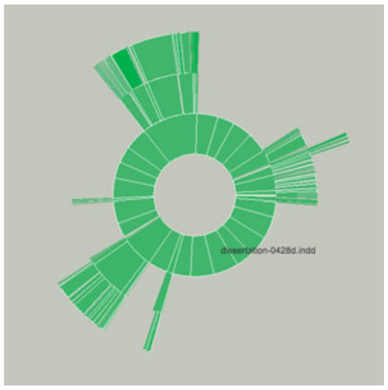


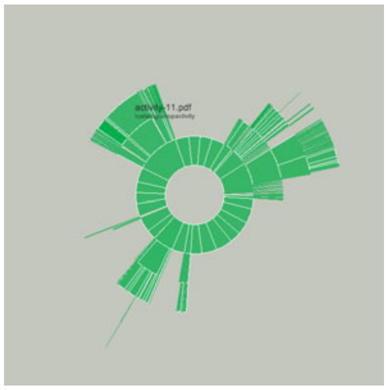
7.5 RADIALSIZE

This sketch was created as a means to understand the relative sizes of objects in a hierarchy in a dynamic manner. The example shown on this spread shows an alternative view of a user's files, a series of folders and documents within folders.

The image is built from the center, the first inner ring has all the files and folders within a particular folder specified by the user. Each arc



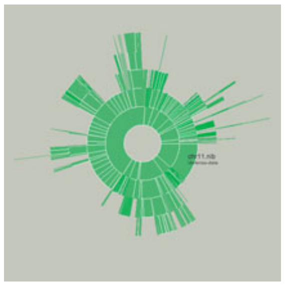


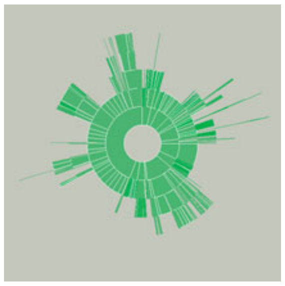


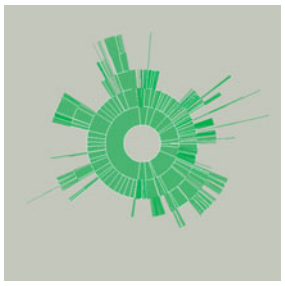
wedge is sized according to the size of the file, or if a folder, the total size of its contents. The three animation steps above show the initial traversal of the folder hierarchy, where rings are added as the program finds new folders and files. Placing the mouse over a wedge shows the user the name of the file or folder it represents.

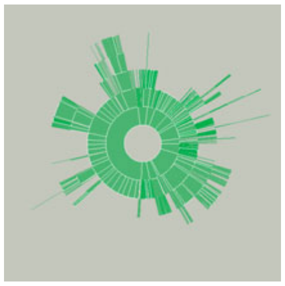
The coloring is based on the relative age of the files, normalized by their standard deviation (see the "Process" chapter). The medium color is for files that fall within the average ranges for the files, brighter colors are further outside the spectrum.

Changes to the folders being displayed will cause radialsize to dynamically update itself. For instance, in the example below, a large file was









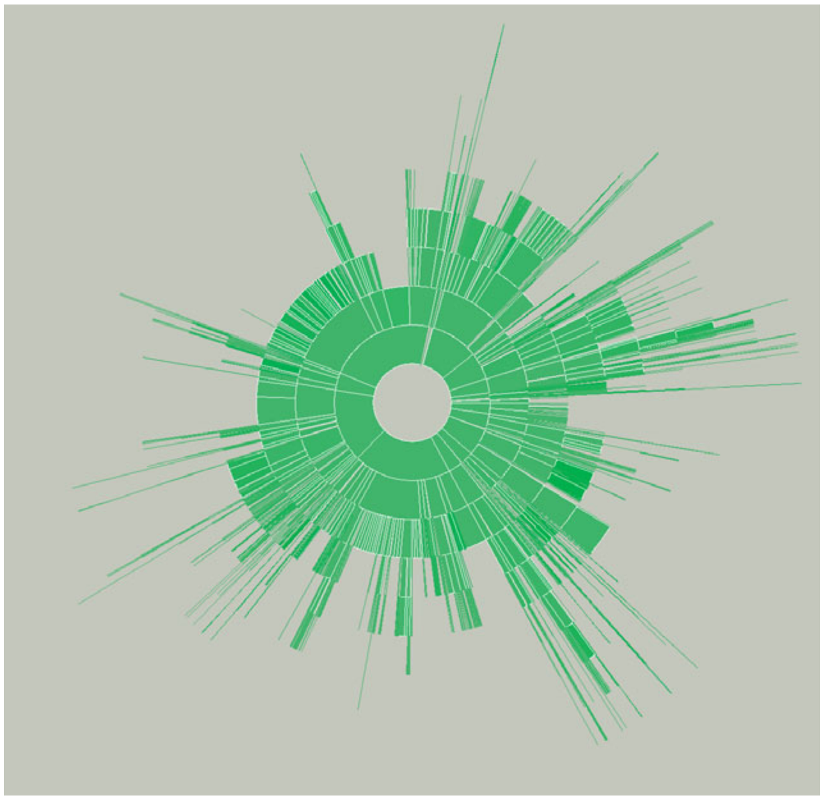
removed, so its wedge is slowly removed, an animated transition that helps the user to know what has changed.

The files in these examples are mostly of similar dates so there isn't much variety to the color.

The notion behind this example is providing a bird's eye view to understand what's taking up the most room on a disk. Additional features would allow the user to click and drag the pie wedge directly to the user's 'trash' (or 'recycle bin'), or compess files or entire folders that require too much room.

Key to this piece is the idea of a visualization that adapts as the data underneath it changes. Creating a diagram of a fixed data set is quite simple, but making them update, and demonstrate (through animation, in this case) to the user how they are being updated is often far more useful, but has received far less research attention.

Shown below is an example of a more complicated file hierarchy, covering six gigabytes of data in a total of 3,000 individual folders.



8 Closing

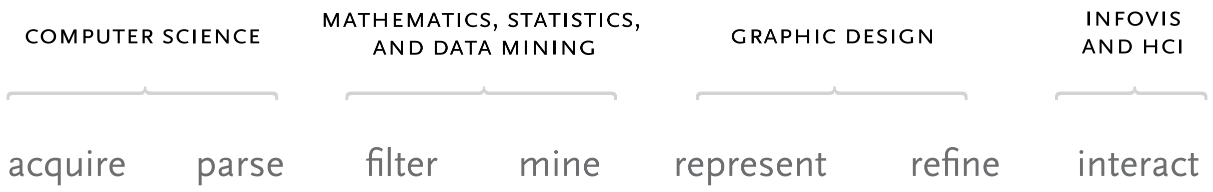
8.1 THE DESIGN OF INFORMATION THROUGH COMPUTATIONAL MEANS

This thesis presents how several typically disciplines are combined to focus on a process for understanding data. Through the combination of these fields, a single, unique discipline of Computational Information Design is formed. A key element of this dissertation is to make the process as accessible to as large an audience as possible, both through the cursory introductions to the multiple disciplines in the "Process" chapter, and online via the programming tool described in chapter six. The tool has been developed and tested by more than ten thousand users, a proven practical use that demonstrates the usability of the software and ability for others to create with it. The examples in the second and fourth chapter serve as a means to demonstrate that the process is grounded in actual implementation rather than simply a rhetorical exercise.

8.2 APPLICATIONS

A near limitless number of applications exist for data that needs to be understood. The most interesting problems are those that seem too "complicated" because the amount of data means that it's nearly impossible to get a broad perspective on how it works.

This might be applied to understanding financial markets or simply the tax code. Or companies trying to track purchases, distribution, and delivery via RFID tags – more data collection without a means to understand it. Network monitoring remains an open problem: is there an intruder? has a machine been hacked or a password stolen? In software, questions of how ever more complex software, whether an operating system or a large scale project, will be necessary. Looking at math and statistical modeling, what does a Hidden Markov Model look like? How do we know a neural network is doing what it's supposed to? Security and intelligence services have an obvious data problem, how to sift through the million bits of information received daily for legitimate threats, without doing so in a manner that infringes on the civil liberties of individuals.



8.3 EDUCATION

Dissemination is a pragmatic issue of simply making Computational Information Design known to a wider audience, through publication, producing more examples, and presentation. This is limited however, by this author's output.

An alternative is to consider how this line of study might be taken up in a school setting, where practitioners of the individual fields (computer science, statistics, graphic design, and information visualization) are brought together to teach parts of a curriculum that cover all aspects of data handling.

It is not uncommon for attempts to be made at such cross-disciplinary curriculum. Some architecture and design schools have core courses that include computer programming alongside more traditional basics like drawing. The problem with these programs is that the focus is on a well-rounded student, rather than the relevant content of what is enabled by the "well-roundedness" to which they aspire. The result is that the programming skills (or others deemed irrelevant) are forgotten not long after (even before?) completion of the course.

The issue is understanding the relevance, and practicing the field. Perhaps by focussing on the goal of understanding data, the individual fields might be given more relevance. The focus being on what each area offers to the goal, in addition to the mind-stretching that comes naturally from learning a topic previously out of one's reach. Some of this author's least favorite high school courses were in Geometry and Linear Algebra, up until graduate school where an intuitive sense of what concepts like sine and cosine meant and became everyday components of the work seen in this thesis. A previous belief that Computer Graphics was about making awful 3D landscapes and virtual reality creatures was traded for an understanding of what software-based graphics brought to the field of Information Design.

8.4 TOOLS

Another challenge of this work is to improve the tools. One side of this challenge is to make simpler tools, making them faster for prototyping than writing software code, as well as usable by more people. The other side is that they must be more powerful, enabling more sophisticated work. Software tools for design (like Illustrator, PageMaker, and Fontographer) led a revolution in layout, design, and typography by removing the burden of rigid printing and typesetting media.

Ideally, the process of Computational Information Design would be accompanied by a still more advanced toolset that supports working with issues of data, programming and aesthetics in the same environment. The logical ideal for such a tool set would be an authoring environment for data representation—combining the elements of a data-rich application like Microsoft Excel, with the visual control of design software like Adobe Illustrator, and an authoring environment like MatLab. The development of this software environment is outside the scope of this thesis because a disproportionate amount of time would be spent on end-user aspects of software engineering, rather than using it to develop relevant projects. Such a toolset would be less code oriented for the aspects of visual construction, employing direct manipulation tools as in Illustrator. Yet the on-screen instantiations of data would, rather than being fixed points in space, be assigned to values from a table or a database.

A step further would look at how to make the visualization of data more like a kind of "kinetic information sculpture," or a lathe for data. The text medium of programming provides great flexibility but is sensory deprivation compared to the traditional tools of artists that employ a greater range of senses. What if the design of information were more like glass blowing? As interfaces for physical computing become more prevalent, what will this mean to how data is handled? Having mapped out the field of Computational Information Design, all the interesting work lies in helping it mature.

9 Bibliography

Abril, J.F. and Guigó, R. *gff2ps : visualizing genomic annotations*. Bioinformatics, 16(8):743-744 (2000).

Adams *et al*. *The Genome Sequence of Drosophila melanogaster*. Science 287(5461):2185-2195(2000).

Anscombe, F. J. *Graphs in Statistical Analysis*. American Statistician, 27. February 1973, 17-21. *via Tufte, 1992*.

Bederson, B.B., Shneiderman, B., and Wattenberg, M. *Ordered and Quantum Treemaps: Making Effective Use of 2D Space to Display Hierarchies*. ACM Transactions on Graphics (TOG), 21, (4), October 2002, 833-854.

Ben-Dor, Amir. Shamir, Ron. Yakhini, Zohar. *Clustering Gene Expression Patterns*. Journal of Computational Biology (Mary Ann Liebert, Inc.) Volume 6, Numbers 3/4, 1999. pp. 281-297.

Bertin, Jacques. *Semiology of Graphics: Diagrams, Networks, Maps*. Madison, Wis.: University of Wisconsin Press, 1983. *English translation and later edition of a text first published in 1967*.

Card, Stuart K. Mackinlay, Jock D. Shneiderman, Ben. *Readings in Information Visualization: Using Vision to Think*. San Francisco: Morgan Kauffman, 1999.

Cho, Peter. *Pliant Type : Development and Temporal Manipulation of Expressive, Malleable Typography*. M.S. Thesis. Massachusetts Institute of Technology, Program in Media Arts & Sciences, 1999.

Chernoff, H. *The use of faces to represent points in k-dimensional space graphically*. Journal of the American Statistical Association. 68, 361-368, 1973.

Couzin, Jennifer. *New Mapping Project Splits the Community*. Science 2002 296: 1391-1393

Coyne, Richard *(Editor) et al*. *Knowledge-Based Design Systems*. Reading, MA. Addison-Wesley Publishing Company, 1990.

Daly, Mark J. *et al. High-resolution haplotype structure in the human genome*. Nature Genetics 29, 229-232 (October 2001) Letters.

The Office of Charles and Ray Eames. *Powers of ten; a film dealing with the relative size of things in the universe and the effect of adding another zero*. Santa Monica, Calif.: Pyramid Films, 1978.

Fayyad, Grinstein, and Wierse. *Information Visualization in Data Mining and Knowledge Discovery*. Morgan Kaufmann. 2001.

Fishwick, Paul *(Editor)*. *Aesthetic Computing*. Cambridge, Massachusetts: MIT Press, in production.

Frankel, Felice. *Sightings : Information Visualization*. American Scientist, March-April, 2004. p. 173-5

Fry, Benjamin. *Organic Information Design*. M.S. Thesis. Massachusetts Institute of Technology, Program in Media Arts & Sciences, 2000.

Gabriel, Stacey B. *et al*. *The Structure of Haplotype Blocks in the Human Genome*. Science 2002 296: 2225-2229.

Garland, Ken. *Mr. Beck's Underground Map*. Capitol Transport Publishing: Middlesex, 1994.

Gershenfeld, Neil A. *The Nature of Mathematical Modeling.* New York : Cambridge University Press, 1999.

Grinstein, Georges & Ward, Matthew O. *"Introduction to Data Visualization"* in Information Visualization in Data Mining & Knowledge Discovery. 2002.

Hand, David J. Manila, Heikki. Smyth, Padhraic. *Principles of Data Mining (Adaptive Computation and Machine Learning)*. Cambridge, MIT Press, 2001.

Hartl, D. L. and A. G. Clark. *Principles of population genetics*. Sunderland, Mass., Sinauer Associates, 1997.

Holt *et al*. *The Genome Sequence of the Malaria Mosquito Anopheles gambiae*. Science 298(5591):129-149 (2002).

The International Genome Sequencing Consortium. *Initial sequencing and analysis of the human genome.* Nature 409, 860-921, 2001.

Ishizaki, Suguru. *Typographic Performance*. Ph. D. Thesis. Massachusetts Institute of Technology, Program in Media Arts & Sciences, 1995.

Ishizaki, Suguru. *Improvisational Design : Continuous, Responsive Digital Communication*. Cambridge, Massachusetts: MIT Press, 2003.

Lamping, John. Rao, Ramana. *Laying out and visualizing large trees using a hyperbolic space*. Proceedings of UIST '94, pp 13–14, 1994.

Lander, Eric S. *The New Genomics: Global Views of Biology*. Science 1996 274: 536-539

Lewontin, R. C. *The interaction of selection and linkage. General considerations; heterotic models*. Genetics 49, 49–67 (1964).

Levin, Golan. *Painterly Interfaces for Audiovisual Performance*. M.S. Thesis. Massachusetts Institute of Technology, Program in Media Arts & Sciences, 2000.

Maeda, John. *The South Face of the Mountain*. Technology Review, Cambridge, MA: MIT Press, vol.101 no.4, July/Aug. 1998.

Maeda, John. *Design By Numbers*. Cambridge, Massachusetts: MIT Press, 1999.

Nickerson *et al*, Nature Genetics, 19:233-240, 1998

Norman, Donald. *Things That Make Us Smart*. Cambridge, Massachusetts: Perseus Books, 1993. p. 43.

Overbye, Dennis. *Pure Math, Pure Joy*. New York Times, June 29, 2003.

Playfair, William. *The Commercial and Political Atlas*. London: Corry. 1786. *via Tufte, 1983*

Rao, Ramana. Card, Stuart K. *The table lens: merging graphical and symbolic representations in an interactive focus+context visualization for tabular information*. ACM CHI '94 Conference Proceedings. (Boston, MA, April 24-28, 1994) 222.

Resnick, Elizabeth. *John Maeda*. Eye Magazine. 37, 2000, p. 15.

Rieder, Mark J. *et al*. *Sequence variation in the human angiotensin converting enzyme*. Nature Genetics 22, 59 - 62 (01 May 1999) Letters.

Rioux, J.D. *et al. Hierarchical linkage disequilibrium mapping of a susceptibility gene for Crohn's disease to the cytokine cluster on chromosome 5.* Nature Genet. 29, 223–228 (2001).

Sabeti, Pardis *et al. Detecting recent positive selection in the human genome from haplotype structure*, Nature 419: 832-837.

Schiffman, Jared. *Aesthetics of Computation – Unveiling the Visual Machine*. M.S. Thesis. Massachusetts Institute of Technology, Program in Media Arts & Sciences, 2001.

Shneiderman, Ben. *Tree visualization with tree-maps: 2-D space-filling approach*. ACM Transactions on Graphics, ACM Press: New York, NY. Volume 11, Issue 1 (January 1992)

Shneiderman, Ben. *Dynamic Queries for Visual Information Seeking*. IEEE Software, 11(6) 1994. pp. 70-77.

Strausfeld, Lisa. *Embodying virtual space to enhance the understanding of information*. M.S. Thesis. Massachusetts Institute of Technology, Program in Media Arts & Sciences, 1995.

Triesman, A. and Gormican, S. *Feature analysis in early vision: Evidence from search asymmetries*. Psychological Review 95(1): 15-48.

Tufte, Edward R. *Envisioning Information*. Cheshire, Conn.: Graphics Press, 1990.

Tufte, Edward R. *The Visual Display of Quantitative Information*. Cheshire, Conn.: Graphics Press, 1992.

Tukey, John Wilder. *Exploratory Data Analysis*. Reading, MA: Addison-Wesley, 1977.

Tukey, John W. *The Future of Data Analysis*, Ann. Math. Statist., 33, 1-67.

Venter *et al*. Science 291(5507):1304-1351 (2001).

Weitzman, Louis Murray. *The Architecture of Information: Interpretation and presentation of information in dynamic environments*. Ph. D. Thesis. Massachusetts Institute of Technology, Program in Media Arts & Sciences, 1995.

Zhang, W. *et al*. *Properties of linkage disequilibrium (LD) maps*. Proc. Natl. Acad. Sci. USA 99, 17004-17007, 2002.