Master's thesis
Master's Programme in Data Science

Grammar-Based Interactive Genome Visualization

Kari Lavikka

June 8, 2020

Supervisor(s):	Professor Giulio Jacucci
Examiner(s):	Professor Sampsa Hautaniemi
	Professor Giulio Jacucci

University of Helsinki
Faculty of Science

P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty Faculty of Science	Koulutusohjelma — Utbildningsprogram — Degree programme Master's Programme in Data Science

Tekijä — Författare — Author

Kari Lavikka

Työn nimi — Arbetets titel — Title

Grammar-Based Interactive Genome Visualization

Työn laji — Arbetets art — Level Master's thesis	Aika — Datum — Month and year June 8, 2020	Sivumäärä — Sidantal — Number of pages 76

Tiivistelmä — Referat — Abstract

Visualization is an indispensable method in the exploration of genomic data. However, the current state of the art in genome browsers – a class of interactive visualization tools – limit the exploration by coupling the visual representations with specific file formats. Because the tools do not support the exploration of the visualization design space, they are difficult to adapt to atypical data. Moreover, although the tools provide interactivity, the implementations are often rudimentary, encumbering the exploration of the data.

This thesis introduces GenomeSpy, an interactive genome visualization tool that improves upon the current state of the art by providing better support for exploration. The tool uses a visualization grammar that allows for implementing novel visualization designs, which can display the underlying data more effectively. Moreover, the tool implements GPU-accelerated interactions that better support navigation in the genomic space. For instance, smoothly animated transitions between loci or sample sets improve the perception of causality and help the users stay in the flow of exploration.

The expressivity of the visualization grammar and the benefit of fluid interactions are validated with two case studies. The case studies demonstrate visualization of high-grade serous ovarian cancer data at different analysis phases. First, GenomeSpy is being used to create a tool for scrutinizing raw copy-number variation data along with segmentation results. Second, the segmentations along with point mutations are used in a GenomeSpy-based multi-sample visualization that allows for exploring and comparing both multiple data dimensions and samples at the same time. Although the focus has been on cancer research, the tool could be applied to other domains as well.

ACM Computing Classification System (CCS):
Visualization → Visualization systems and tools → Visualization toolkits
Applied computing → Life and medical sciences → Bioinformatics

Avainsanat — Nyckelord — Keywords

genomic data visualization, grammar of graphics, fluid interactions

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Acknowledgements

Although the text in this thesis is a result of independent work, it would not have been possible without collaboration and support from numerous people.

I'm extremely grateful to Jaana Oikkonen, one of the domain experts. She has given valuable feedback during the research and development process. I would also like to extend my deepest gratitude to another domain expert, Rainer Lehtonen, who has taught me a lot about cancer genetics and inspired me during my studies and this work.

I would like to thank Oskari Lehtonen and Antti Häkkinen for great discussions and testing GenomeSpy with their own data. I am also grateful to Yilin Li and Amjad Alkodsi for data analyses. Moreover, thanks also goes to Ville Rantanen for help in infrastructure problems. I'd like to recognize the feedback from all the former and current lab members who participated in the user study. Finally, I'm grateful to Mikko Kivikoski for recruiting me to the research group, giving helpful feedback, and discussions about birds!

I very much appreciate Luana Micallef for teaching information visualization to me and for the encouragement in the very early phases of this work. Thanks also to Chen He for a great discussion about visualization in general and feedback on the design of GenomeSpy. I also thank Kari Pitkänen for feedback on my writing.

I thank my lovely wife Rita for guidance and helpful feedback, and my brother Antti for feedback.

I must, of course, thank professor Sampsa Hautaniemi for the opportunity to work in his lab, all the valuable feedback and guidance, and providing an access to all that interesting data. Finally, I thank professor Giulio Jacucci for constructive criticism and supervision of this thesis.

Contents

1 Introduction **1**

2 Background **5**

2.1 Biological Background 5

2.1.1 Human Genome 5

2.1.2 Genomic Alterations in Cancer 6

2.2 Genomic Features 7

2.3 Genome Browsers 8

2.3.1 UCSC Genome Browser 8

2.3.2 Integrative Genomics Viewer 10

2.3.3 HiGlass 12

2.4 Grammars of Graphics 14

2.4.1 ggplot2 14

2.4.2 Vega-Lite 17

2.5 GPU Rendering 22

2.5.1 State-of-the-art Examples 23

2.6 Fluid Interactions 24

3 Design and Architecture of GenomeSpy **25**

3.1 JavaScript Library 26

3.2 The Grammar 27

3.2.1 Data Input 27

3.2.2 Graphical Marks 27

3.2.3 Data Transforms 29

3.2.4 View Composition 30

3.3 Supporting Genomic Data 31

3.3.1 Concatenated Genomic axis 31

3.3.2 Built-in Annotation Tracks 31

3.4 Faceting Multiple Samples 33

3.4.1 Facet View 34

3.4.2 Sample Attributes as a Metadata Heatmap 34
3.4.3 Sorting and Filtering Samples 35
3.4.4 Focus+Context with "Fisheye" and "Peek" 37
3.5 Avoiding Overplotting while Zooming 38
3.5.1 Semi-geometric Zoom 39
3.5.2 Score-based Semantic Zoom 40
3.6 Architecture 42
3.6.1 Initialization 42
3.6.2 Rendering with WebGL 42

4 Validation **45**
4.1 Data 45
4.2 Case Study I: Copy-Number Segmentation Assessment 45
4.2.1 Copy-Number Segmentation 46
4.2.2 Zoomable Visualization 48
4.2.3 Task Abstration 48
4.2.4 Implementation 48
4.3 Case Study II: Multidimensional Sample Comparison 52
4.3.1 Task Abstraction 52
4.3.2 Visualization Design 53
4.3.3 Implementation 55
4.3.4 User study 57

5 Discussion and Conclusions **59**
5.1 Increasing Expressivity 59
5.2 Shortcomings of the Architecture 60
5.3 Insights from the User Studies 60
5.4 Towards Fluid Analysis 61
5.5 Concluding Remarks 61

Bibliography **63**

Appendix A Code examples **69**
A.1 Layering a Lollipop Plot 69
A.2 Cancer Gene Census Track 71

Appendix B Verbal Workshop Feedback **73**

1. Introduction

The Human Genome Project and later efforts employing next-generation sequencing technologies have generated vast amounts of DNA-sequence data, which have been refined into detailed knowledge of our genome [28]. Researchers not only exploit this existing knowledge to decipher the mechanisms of diseases but also use the same techniques to generate even more data.

Although computational and statistical methods form the foundation of the genomic data analysis, visualization has an indispensable role in the sense-making process. This is especially true in the initial exploratory phase when an investigator is building a mental model of the data and forming new hypotheses [4]. Consequently, numerous tools have been developed for researchers to visualize sequence data at different points of the analysis processes. However, the tools have often been developed in an ad-hoc manner to fulfill very definite needs [27]. Yet, comprehensive exploration of high-dimensional data calls for generality.

Because the human genome is vast, static visualizations have limited use in exploration. Genome browsers are a class of interactive genome visualization tools that allow for exploring coordinate-based genomic data by zooming and panning. However, they couple the visual presentation with specific file formats, limiting how the data dimensions can be visualized and explored. On the other hand, although the tools provide interactivity, the implementations are often rudimentary. For instance, the lack of animation in transitions between zoom levels necessitates users to reorient themselves after each zoom interaction [7], encumbering the exploration process.

This work aims to improve the exploration of coordinate-based genomic data. I approach the challenge on two fronts: First, I break away from the specificity and rigidness by introducing abstractions that decouple the visual representation from the underlying file formats. In other words, the first goal is to allow the users to explore the visualization design space and implement visualizations that properly display the underlying data. Second, I develop and implement interaction designs that allow users to explore the vast genomic space more efficiently. In other words, the second goal is to provide a user experience that excites the user to explore the data, and ultimately, experience insights that lead to new hypotheses.

To design the abstractions and allow for more flexible visualization authoring, I apply the theory of declarative, grammar-based [46] visualization authoring. A grammar consists of building blocks and rules for combining them. Although visualization grammars are widely utilized in statistical graphics [44] and static genome visualizations [47], no genome browser has embraced the grammar-based approach so far.

To design the interactions, I apply the guidelines of fluid interactions [10], which include themes such as smoothly animated transitions between views and avoidance of indirection by direct manipulation [37]. Prior research suggests that animation has potential to improve the perception of causality, keep the viewers oriented, and provide an emotionally engaging experience [16]. Thus, proper use of animation could increase productivity. However, animating data items of large datasets such as genomic data is challenging. Yet, the challenge has been overcome in other disciplines, such as geospatial visualization, by using graphics processing unit (GPU) powered rendering [1].

The research materializes into GenomeSpy, a flexible genome-browser-like tool that allows researchers to build interactive visualizations with greater expressivity than what is generally possible with the current state-of-the-art. Also, the visualizations provide fluid interactions that facilitate exploration. Finally, I apply GenomeSpy to different phases of cancer genome analysis, demonstrating its practical utility.

The key contributions of this thesis are as follows:

1. A visualization grammar optimized for genomic data. The grammar is heavily inspired by related work but adds support for genomic coordinates, sample metadata, and means for a side-by-side comparison of multiple samples and data dimensions, both at the same time.

2. An interaction design for a genome browser. The design improves upon the state of the art by introducing zoom behaviors that help managing overplotting, and a user interface and animated transitions for sorting and filtering multiple (up to thousands of) samples.

3. A software architecture that utilizes a GPU in rendering the visualization. Based on a grammar-based specification, data translates into geometric primitives that are handled by the GPU in a highly parallel manner. GPU is utilized for implementing smooth animations that support the fluid interactions.

4. A novel visualization design and GenomeSpy-based implementation for multidimensional cancer genomics data. The design allows biologists to observe copy-number variation, loss of heterozygosity, and point mutations simultaneously and compare them between samples or to various annotations.

The rest of the thesis is organized as follows:

Chapter 2 introduces the human genome and common genomic alterations in cancer and features the state-of-the-art in genome browsers. The chapter continues by surveying prior work that this thesis builds upon, including popular visualization grammars, visualization toolkits that exploit the graphics processing unit for high rendering performance, and the principles of fluid interactions.

Chapter 3 outlines the design and architecture of GenomeSpy. The first half present the visualization grammar and the latter half focuses more on the interaction design.

Chapter 4 presents two practical case studies that demonstrate GenomeSpy's applicability to real biological research problems. The latter case study introduces a novel visualization design, implements it using GenomeSpy, and summarizes the results of a user study conducted during a workshop.

Chapter 5 discusses the limitations of the current design and sketches future research and development plans, including a concept of "fluid analysis".

2. Background

This chapter briefly explains the application domain of this work, discusses the current state of the art, and covers the research and technologies this work builds upon.

2.1 Biological Background

2.1.1 Human Genome

All cells on earth use deoxyribonucleic acid (DNA) as the hereditary material, a medium that stores and passes the blueprints of the cells to next cell generations [2]. DNA is a polymer, a long chain of *nucleotides* that consist of a sugar-phosphate backbone and four different bases, which are commonly symbolized as letters: A, C, G, and T. These successive letters similarly encode information as the English language with its 26 distinct letters. The nuclear DNA is stored in cells as discrete *chromosomes*, which form the *genome* of the cell. Except for the sex chromosome, humans normally have two copies of each chromosome.

Parts of the DNA contain *genes*, nucleotide sequences that encode proteins. Three consecutive nucleotides form a *codon*, which is translated into amino acid, a building block of proteins. Several different codons translate into the same amino acid. Thus, the code has redundancy. Because the chromosomes appear as pairs, humans have two copies of each gene. If the two copies are not identical, they are called *heterozygous*.

Genome Assemblies

Just two decades ago, researchers could only give genes an approximate location based on the chromosomal banding pattern, which is visible in stained chromosomes, and nearby genes. For example, the *locus* (location) of the tumor suppressor gene *TP53* could be written as *17p13.1*, which means chromosome 17, *p* arm, and band *13.1*.

Today, the *loci* can be expressed precisely. The Human Genome Project delivered a reference genome assembly, which is not only a substrate for sequence alignment but also provided a coordinate system for expressing the exact locations of genomic features such as genes [35]. For instance, using the *GRCh38* reference assembly, the locus of the *TP53* gene can now be written as chr17:7661779-7687550. The numerical values are ordinal numbers of the nucleotides.

2.1.2 Genomic Alterations in Cancer

Cancer is a result of deregulated cell proliferation and metastasis [15]. More precisely, cell division and growth in normal tissues are carefully controlled, but mutations affecting critical genes may disturb this orderliness and lead to uncontrolled proliferation. Sometimes the neoplastic (abnormal) cells gain means to escape their original tissue and continue proliferation at new sites, forming metastases. The next subsections introduce the mutation types that are relevant for this thesis.

Point Mutations

As cells proliferate, they need to replicate their DNA, i.e., make a copy of it. Although the replication machinery has sophisticated error detection mechanisms, an incorrect nucleotide or two may sometimes end up in the nascent DNA strand. On the other hand, environmental radiation and chemicals may damage the DNA and evade the cell's repair mechanisms.

The substitution of a single nucleotide is called a *point mutation*. Because the genetic code has redundancy, the substitution may be synonymous – it does not change the amino acid. If the mutation occurs in a critical location, such as the active site of a protein, the protein function is altered.

Small Insertions and Deletions

Sometimes a mutation leads to a missing or extra nucleotides in the DNA. These kinds of mutations are called *indels* (insertion-deletion). Because the DNA sequence is interpreted in frames of three nucleotides, an indel may garble all the subsequent codons, leading to a malfunctioning protein.

Structural Alterations

If the DNA double-strand breaks for some reason, the cell tries to fix it. Occasionally the fix fails, and the free ends find incorrect mates. A direct consequence may be a *fusion gene* that contains parts from two genes.

More commonly, incorrectly joined breaks cause problems in further cell divisions. Each daughter cell should inherit two sets of homologous chromosomes. However, the deformed chromosomes segregate incorrectly and may lead to even more breaks. Thus, one of the daughter cells gets excess DNA, and the other one too little. Such *copy number alterations* usually affect the expression of multiple genes, i.e., too much or little protein is produced.

Development of Cancer

The progression of cancer is a multi-step process. Traits emerging from mutations and epigenetic changes give cells a selective advantage to outgrow from the local tissue environment [15]. However, getting the ability to metastasize cancer needs multiple genetic alterations. Still, most mutations are only passengers and have no role in tumor development. Finding the drivers needs scrutiny of the genome, and data visualization is a crucial method in this challenge. In the next section, we continue with the taxonomy of genomic features in order to better understand how the genome could be visualized.

2.2 Genomic Features

The reference genome consists of roughly 3.2 billion nucleotides. Such a string of letters per se is of little interest. More often, the focus of interest is in *genomic features*, which are data items derived from or associated with the reference genome and having specific coordinates.

Nusrat et al. [27] describe a taxonomy of genomic features, allowing for abstract understanding of the genomic data. Features are data items that can be mapped to a single nucleotide (*point feature*) or a range of nucleotides (*segment feature*). The features may be associated with any number of attributes, which may be nominal (categorical), ordinal, or quantitative. A feature without attributes contains only its genomic coordinates. However, it might as well have multiple attributes, such as a name and the role in cancer, if it represents a gene, for example.

A *feature set* contains features that belong to the same entity. For example, all point mutations of a single sample, or all genes in the human genome are feature sets. A set may be *sparse*, such as point mutations here and there, or it may be *contiguous*, such as the GC-content in windows of one thousand bases over the whole genome.

The feature sets may have associated *metadata*. For example, if a feature set contains point mutations, its metadata might include attributes such as the patient id, anatomical site, or type of disease.

2.3 Genome Browsers

Genome browsers are a group of interactive visualization tools that display genomic features using a linear layout [27]. They map the genomic coordinates onto the horizontal axis and allow the user to *navigate* around the genome by zooming and panning. Commonly the visualization consists of multiple stacked *tracks* that may display various types of feature sets. Usually, one of the tracks displays gene annotations, which give positional context for the other features.

The linear layout of genome browsers is not ideal for all genomic data. Notably, if the genomic coordinates are not relevant for the research question, a more abstract visual representation may be better. For instance, when comparing the mutational status of a set of genes between several samples, a matrix view like Oncoplot [24] displays the data more concisely.

The next subsections survey some state-of-the-art genome browsers.

2.3.1 UCSC Genome Browser

One of the earliest genome browsers is the University of California Santa Cruz (UCSC) Genome Browser [17] (Figure 2.1), which is still developed and widely used [21]. It is a web-based service that allows the user to build a visualization from a large number of available tracks, which display data hosted at the UCSC or external *track hubs* located outside the UCSC. Users may also visualize their own data by first uploading it to the service or making them available on a publicly accessible web server.

Some tracks, such as the *Genome Base Position* and *Vertebrate Multiz Alignment & Conservation*, support a single, specific dataset. The former displays genomic axis ticks, the reference sequence, and the amino acids in three reading frames. The latter displays the sequence conservation between different species.

Some other track types are more generic; for example, the *GC Percent in 5-Base Windows* track uses the *bigWig* track type, which displays contiguous, quantitative features. The data must be provided in any of the three file formats that support a single quantitative variable, namely *wiggle* (WIG), *bigWig*, or *bedGraph*. Wiggle is a text-based format, which supports both variable and fixed-width elements. BigWig is an indexed binary format, which has superior performance on large datasets but supports only fixed-width elements. BedGraph is similar to wiggle, but supports sparse data, i.e., it may have gaps.

Figure 2.2 displays the configuration page for the GC Percent track. The user can configure settings such as the height of the track, data domain, and smoothing. Default track settings can be provided in the header of the data file. Thus, the track

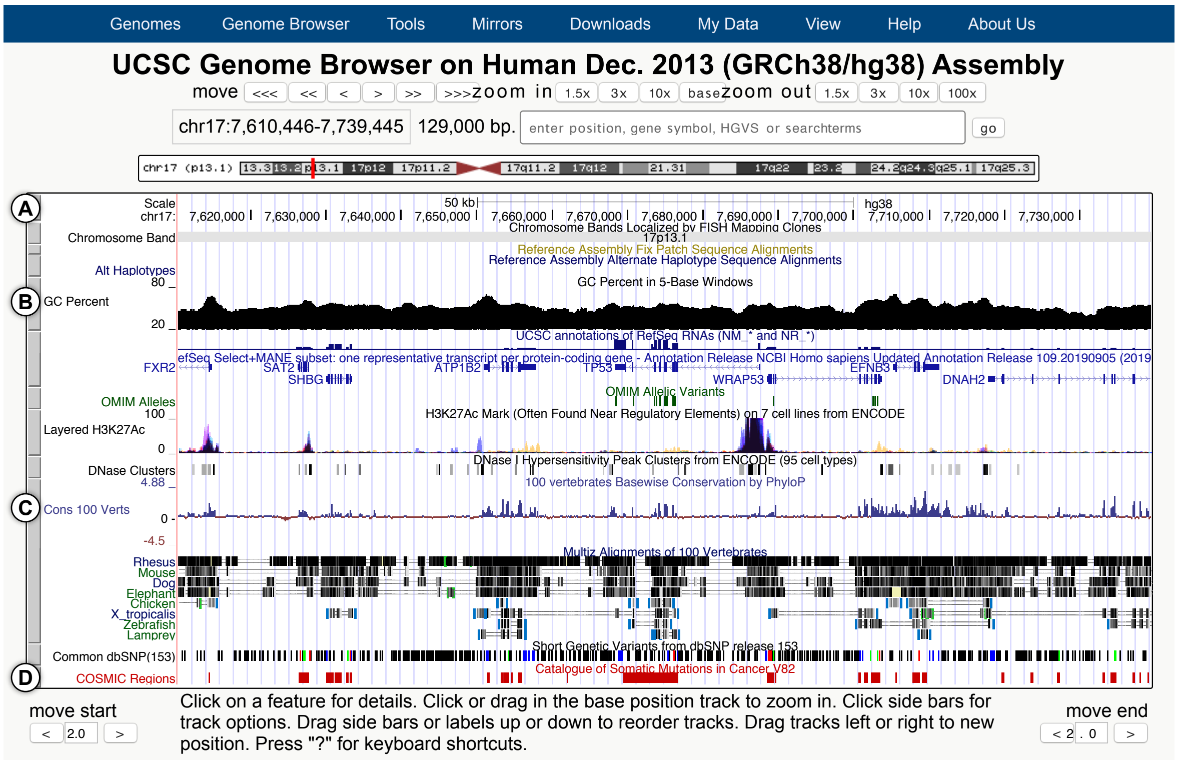

Genomes
Genome Browser
Tools
Mirrors
Downloads
My Data
View
Help
About Us
UCSC Genome Browser on Human Dec. 2013 (GRCh38/hg38) Assembly
move <<< << < > >> >>>
zoom in 1.5x 3x 10x base
zoom out 1.5x 3x 10x 100x
chr17:7,610,446-7,739,445
129,000 bp.
enter position, gene symbol, HGVS or searchterms
go
A
B
C
D
Scale
chr17:
Chromosome Band
Alt Haplotypes
GC Percent
FXR2
OMIM Alleles
Layered H3K27Ac
DNase Clusters
Cons 100 Verts
Rhesus
Mouse
Dog
Elephant
Chicken
X_tropicalis
Zebrafish
Lamprey
Common dbSNP(153)
COSMIC Regions
GC Percent in 5-Base Windows
UCSC annotations of RefSeq RNAs (NM_* and NR_*)
OMIM Allelic Variants
H3K27Ac Mark (Often Found Near Regulatory Elements) on 7 cell lines from ENCODE
DNase I Hypersensitivity Peak Clusters from ENCODE (95 cell types)
100 vertebrates Basewise Conservation by PhyloP
Multiz Alignments of 100 Vertebrates
Short Genetic Variants from dbSNP release 153
Catalogue of Somatic Mutations in Cancer V82
move start
move end
Click on a feature for details. Click or drag in the base position track to zoom in. Click side bars for track options. Drag side bars or labels up or down to reorder tracks. Drag tracks left or right to new position. Press "?" for keyboard shortcuts.

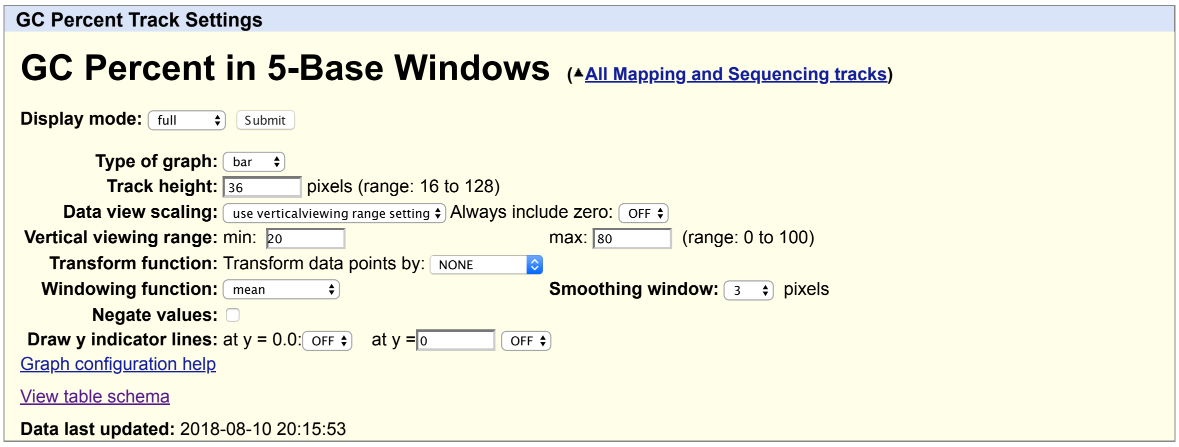
Figure 2.1: A screenshot of the UCSC Genome Browser with a custom set of tracks. The following tracks are discussed in the main text: (A) Genome Base Position, (B) GC Percent in 5-Base Windows, (C) Vertebrate Multiz Alignment & Conservation, (D) Catalogue of Somatic Mutations in Cancer

is defined by its track type, the default settings, and the data shown on the track.

The *Catalogue of Somatic Mutations in Cancer* track uses *BED*, another generic track type. It displays sparse segments that may have a limited set of attributes, namely a name, score (0-1000), and a red-green-blue (RGB) color. The score can be visualized as different shades of gray. The track type also supports a few other attributes that are useful if the data represents genes with exons. The data can be provided in the text-based BED file format or as the binary *bigBed* format, the latter being optimized for large datasets.

The technology stack of the browser is old-fashioned and has changed little since the initial release. The graphics are drawn at the server and are delivered to the user's web browser as images. When the user clicks the navigation buttons or pans the viewport by dragging, the server draws a new graphics and delivers them to the web browser. Although such an approach is lightweight for the web browser, the user experience is sub-par by modern standards. There is a considerable delay after each navigation action, and the viewport changes abruptly, without animation, when zoomed. Thus, the user has to scan the viewport carefully to find the new locations of the features.

The UCSC Genome Browser is written in the C programming language. Although


GC Percent Track Settings

GC Percent in 5-Base Windows (▲All Mapping and Sequencing tracks)

Display mode: full Submit

Type of graph: bar
Track height: 36 pixels (range: 16 to 128)
Data view scaling: use verticalviewing range setting Always include zero: OFF
Vertical viewing range: min: 20 max: 80 (range: 0 to 100)
Transform function: Transform data points by: NONE
Windowing function: mean Smoothing window: 3 pixels
Negate values:
Draw y indicator lines: at y = 0.0: OFF at y = 0 OFF
Graph configuration help

View table schema

Data last updated: 2018-08-10 20:15:53

Figure 2.2: Screenshot of the GC Content settings.

the application is provided primarily as a web-based service, the user can also install it on a local Linux server. Despite being insurmountable for an average user, that may be necessary if sensitive data is visualized.

2.3.2 Integrative Genomics Viewer

The Integrative Genomics Viewer (IGV) (Figure 2.3) is a genome browser that is designed specifically to visualize users' own data [42]. It allows for integrating clinical and phenotypic metadata, which can be used to dynamically group, filter (Figure 2.4), and sort (Figure 2.5) the datasets. The user can study correlations between distant loci by opening multiple viewports that display different genomic regions. With some exceptions, the user can overlay tracks; for instance, point mutations can be shown on top of copy-number variation.

File format support of IGV and the UCSC Genome Browser is comparable. However, because IGV is a desktop application running on the user's computer, opening and viewing local data files is much faster. As in the UCSC Genome Browser, some tracks are very specific; for example, *BAM* files, aligned next-gen sequencing read data, are displayed in a sophisticated "pileup" view along with a coverage plot. The generic *data tracks* support a single contiguous attribute, which IGV can render as a bar plot, line plot, points, or heatmap. The user can adjust the data domain and colors. Negative and positive values can be assigned different colors, except in the heatmap, which supports a three-level threshold scale with interpolated colors.

IGV is written in the Java programming language. The IGV development team has also written a web-based JavaScript library, igv.js, that supports many of the same file formats and track types as the IGV. However, it has no support for metadata, and it is overall less mature.

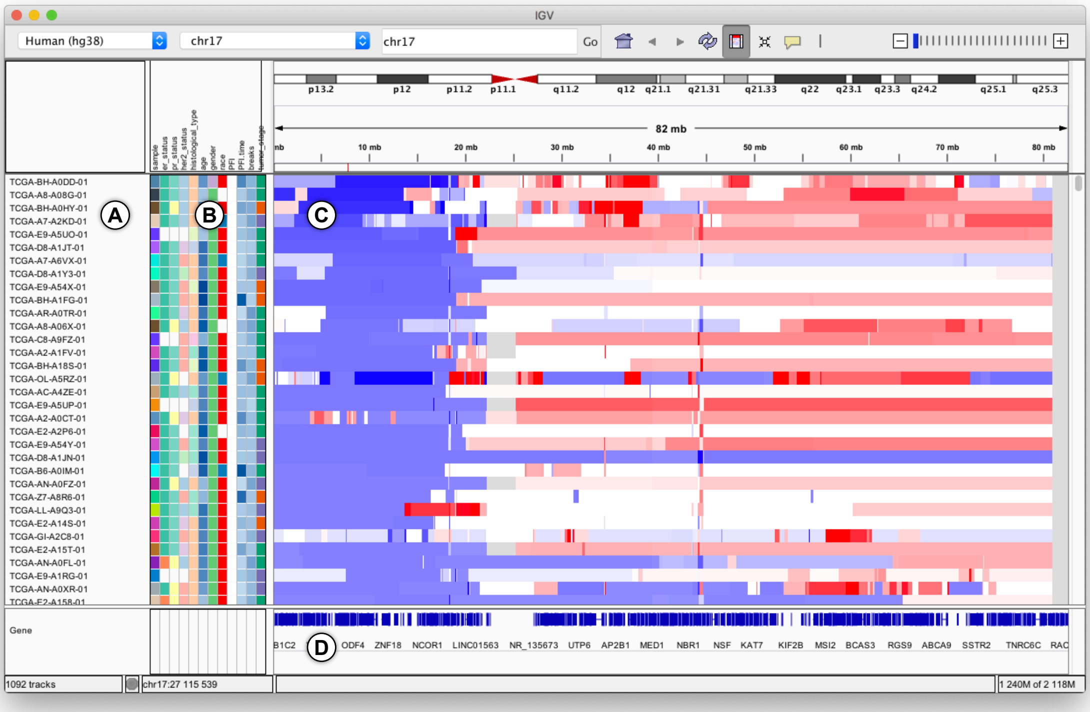
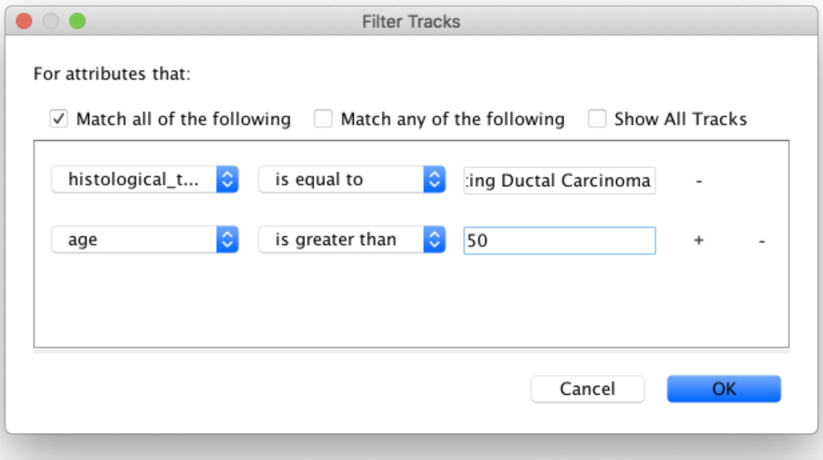


Figure 2.3: Screenshot of the Integrative Genomics Viewer with copy-number variation data. Components of the user interface: (A) track names, (B) sample attributes (metadata), (C) segmented copy-ratios, blue: deletion, white: neutral, red: amplification, (D) gene annotations

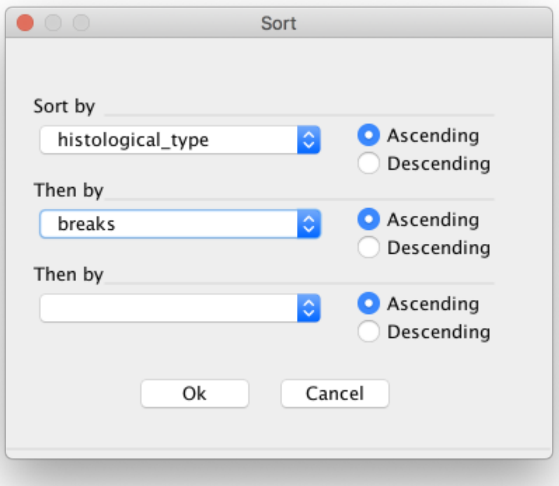

Filter Tracks

For attributes that:

[✓ Match all of the following] Match any of the following Show All Tracks

histological_t...	is equal to	:ing Ductal Carcinoma
age	is greater than	50

Figure 2.4: Screenshot of the IGV filter dialog where the user can filter the tracks by multiple sample attributes. However, the functionality is not integrated into the visualization and poses a cognitive load on the user. For instance, the dialog does not provide a dropdown list for the values of the nominal attribute *histological_type* – instead, the user has to remember and type the value into the text field.



Sort by
histological_type [✓ Ascending] Descending

Then by
breaks [✓ Ascending] Descending

Then by
[✓ Ascending] Descending

Figure 2.5: Screenshot of the IGV sort dialog where the user can choose up to three sample attributes for sorting the tracks. When sorting just by a single attribute, the user can click the attribute headers in the main window.

2.3.3 HiGlass

HiGlass is a web-based visualization tool (Figure 2.6) for exploring chromosomal contact matrices along with one-dimensional genomic tracks [18]. Contact matrices allow investigators to study the spatial organization of the genome in the nucleus and, for example, its effect on gene regulation. Although HiGlass is designed primarily for two-dimensional data, we focus here on the one-dimensional genomic tracks.

HiGlass supports vast datasets by using a multiscale architecture. All data are binned using their genomic coordinates and tiled at multiple resolutions. Thus, as the user zooms into the data, only the required tiles with an appropriate resolution are loaded. BigWig files support such a data access and can be directly viewed in HiGlass. Just as in the IGV, the user can display the contiguous, quantitative data as a bar plot, line plot, points, or heatmap.

Sparse features are supported through the BED file format. However, sparse data with possibly overlapping segments cannot be aggregated by computing, for example, averages or sums for each bin. Instead, each feature must be assigned an "importance value", which is used for choosing the representative feature for each bin. HiGlass uses this scheme on the gene annotation track – each bin contains only the gene that has the most citations in the literature. Thus, as the user views the whole genome with all its chromosomes, they likely see familiar genes. When the user zooms closer, less known genes become visible.

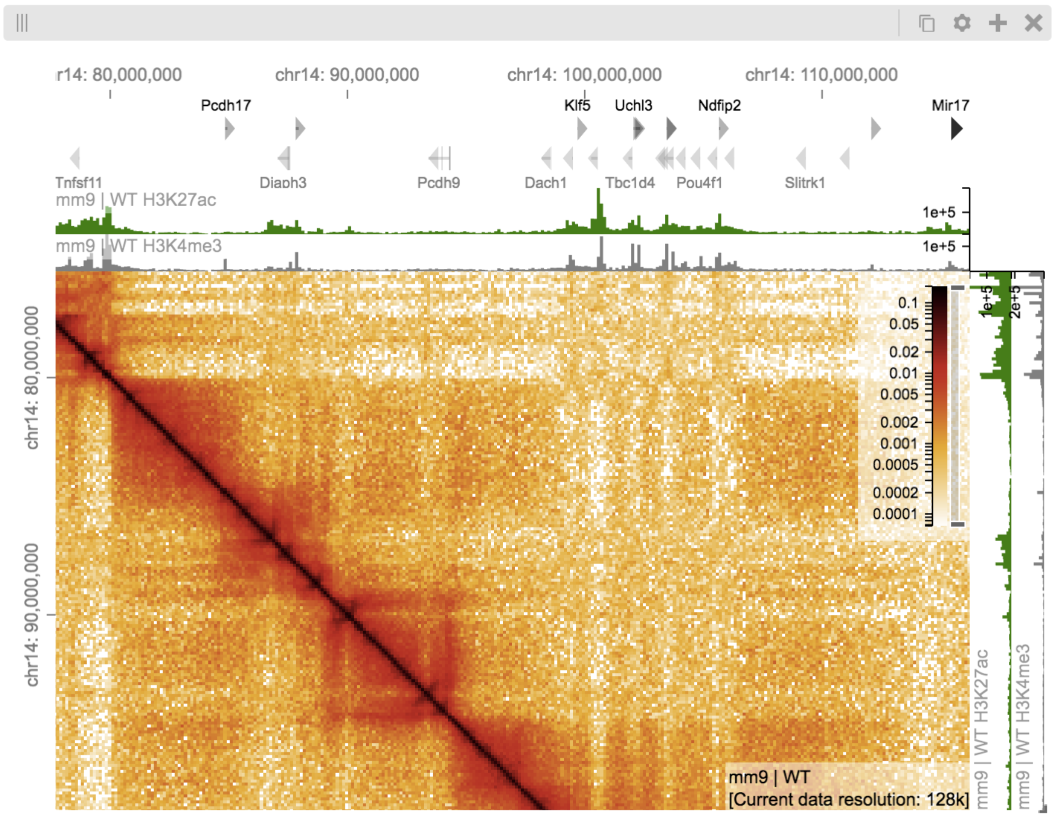


Figure 2.6: Screenshot of HiGlass.

Whereas the IGV supports multiple viewports, HiGlass provides a more flexible concept of *composable linked views.* The user can create a complex layout of views that may have synchronized axes or have an overview-detail relationship. For example, the visualization might contain several contact matrices that represent different cases; as the user zooms or pans one of them, all the others respond similarly. This concept can be applied to one-dimensional tracks as well, resulting in the same behavior as in the UCSC Genome Browser and IGV with their stacked tracks. However, HiGlass does not support sample attributes and, thus, does not allow the user to filter and sort the tracks by them.

HiGlass consists of a server written in the Python programming language and the client written in JavaScript. The user cannot directly open their own data files but, instead, has to "ingest" them into a locally installed HiGlass server. Setting up a local installation is straightforward with the provided Docker container. Moreover, the HiGlass website provides a client and a number of hosted datasets for exploration.

To provide a smooth user experience with support for continuous zooming and panning, HiGlass uses WebGL (section 2.5) to render the graphics.

2.4 Grammars of Graphics

Visualization tools often provide the user with a predefined set of different chart types, such as bar charts, pie charts, and multi-series line charts. However, for the user of a visualization software, no matter how extensive, the variety of predefined chart types may be insufficient [46]. On the other hand, when a developer of a visualization software focuses on chart types:

> ..., our package will have no deep structure. Our computer program will be unnecessarily complex, because we will fail to reuse objects or routines that function similarly in different charts. And we will have no way to add new charts to our system without generating complex new code. ([46])

In his book *The Grammar of Graphics* [46], Leland Wilkinson outlines a vocabulary and grammatical rules for creating *graphics*, perceivable graphs. While a language consists of words, grammar is a formal system that defines the lawful sentences that can be formed from these words. Words are analogous to components of a graphic: scales, guides, and graphical marks with various visual properties. A grammar defines how these components can be used in a structured way to specify graphics. The variety of possible graphics is no longer limited to predefined sets of chart types.

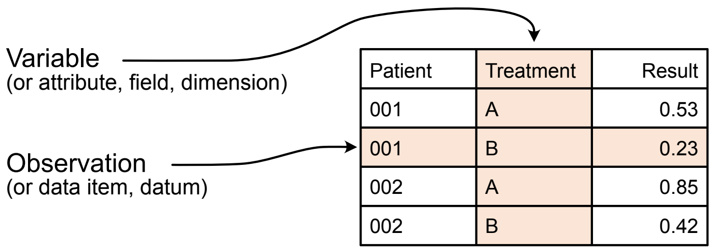
Although Wilkinson originally introduced the concept, no freely available implementation exists. Thus, I focus on two more recent and widely used grammars, both of which are heavily influenced by The Grammar of Graphics. Because *the Grammar of Graphics* refers to Wilkinson's grammar, I use the term *visualization grammar* for such grammars in general.

2.4.1 ggplot2

Ggplot2 [44] is an implementation of the Grammar of Graphics for the R statistical programming language [29]. However, ggplot2 develops Wilkinson's ideas further, introducing a layered grammar with a hierarchy of defaults. Moreover, as the grammar is embedded in the R programming language, some of its components are already provided by the language. This subsection presents a very brief introduction to ggplot2, along with the central concepts of visualization grammars.

Tidy Data

The creation of a visualization usually begins with data. A grammar allows us to specify how the data should be transformed into a graphic. However, there are numerous ways to organize data in a table. While many of the ways may be practical



Patient	Treatment	Result
001	A	0.53
001	B	0.23
002	A	0.85
002	B	0.42

Figure 2.7: An example of tidy data. This data frame is called `treatments` in the examples.

for presentation purposes or for entering the data in a spreadsheet, few are practical for visualization. *Tidy data* is organized in a specific, standardized way that makes analysis and visualization uncomplicated [45].

Figure 2.7 displays an example of tidy data. Each row represents an observation and each column represents a variable. Data organized in such a way are easy to manipulate by filtering, aggregating, and by deriving new variables from the existing ones. Variables of tidy data are straightforward to map to visual properties such as size, color, or symbol.

Creating a plot with ggplot2 starts by specifying the data:

```
ggplot(data = treatments)
```

However, this code produces just an empty plot because we have yet to specify how the provided data should be visualized.

Geometries and Aesthetics

Marks are geometric objects used as building blocks in a graphic [26]. They have properties that influence their appearance. For example, a rectangle might have a width, height, color, opacity, and so on. On the other hand, the properties could be parameterized differently – instead of having a width, a rectangle could have its left and right coordinates as properties. These properties are generally called *visual channels*. The Grammar of Graphics and ggplot2 call them *aesthetics*, however.

In ggplot2, a *geometry* is a parameterization of a graphical mark and an associated *statistic*. For instance, `geom_rect` parameterizes a rectangle using its four corners. `geom_col`, on the other hand, uses the `x` and `y` channels to parameterize a rectangle that has a position on the x axis and a height. Such a parameterization allows for creating a graphic that is known as a *bar chart*. Both of these geometries use `identity` statistic. `geom_bar`, however, uses the `count` statistic, which aggregates the observations and calculates grouped counts. Thus, the height of the bar is based on the number of observations.

```
ggplot(data = treatments) +
    geom_col(aes(x = patient,
                 y = result,
                 fill = treatment),
             position = "dodge")
```

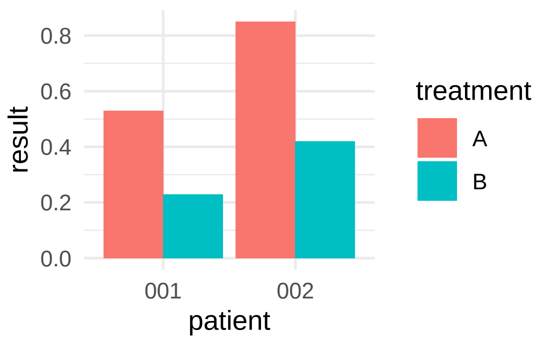


Figure 2.8: Using `geom_col` to create a bar chart. The `aes` function “quotes” its arguments so that they can be evaluated in the context of the data frame. The `fill` aesthetic maps a variable to geometry’s fill color. The positions are “dodged” to prevent them from overlapping.

```
ggplot(data = treatments,
       mapping = aes(x = patient,
                     y = treatment)) +
  geom_tile(aes(fill = result)) +
  geom_text(aes(label = result))
```

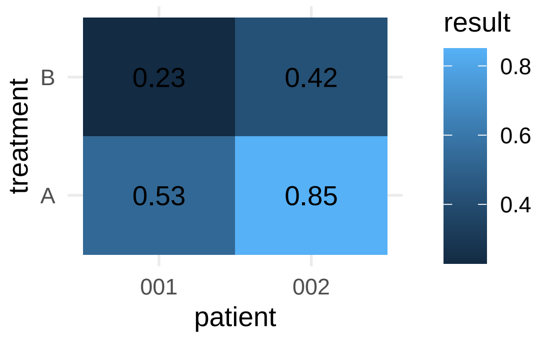


Figure 2.9: The same data as in Figure 2.8 but displayed as rectangles that are parameterized in a different way. The `geom_tile` creates a heatmap. The exact values are layered on top of the heatmap and displayed using `geom_text`. The aesthetic mappings (`x` and `y`) that are common to both geometries are passed to the `ggplot` function, keeping the code succinct.

Sometimes the marks can obscure each other. For such cases, ggplot2 allows for adjusting their positions using various strategies. For instance, `jitter` dislocates point geometries randomly a little bit, whereas `stack` and `dodge` strategies adjust bar and col geometries by stacking them or placing them tightly next to each other.

In Figure 2.8, `geom_col` is used to display the observations in our treatments data as a bar chart. Figure 2.9 displays the same data as a heatmap using `geom_tile`, which parameterizes the rectangles using their center points.

Scales

In Figures 2.8 and 2.9, the variables of the data have been mapped to various visual channels of the rectangles. For example, in Figure 2.8 the fill color of a bar depends on the `treatment` variable. However, a treatment does not naturally map to a color – we need a function that has its domain based on the variable, and its range based on the visual channel (or aesthetic). Such functions are called *scales*. Treatments could be converted to colors with a scale function such as:

$$f(x) = \begin{cases} \text{“red”}, & \text{if } x = \text{“A”} \\ \text{“cyan”}, & \text{if } x = \text{“B”} \end{cases} \tag{2.1}$$

The type of a treatment is a *nominal* (categorical) variable, and can be converted to colors by using a simple lookup table. Such functions are often called *categorical color schemes.*

The treatment result, on the other hand, is a *quantitative* variable, which could be converted to heights with a function such as:

$$f(x) = \frac{x - \min(\mathbf{x})}{\max(\mathbf{x}) - \min(\mathbf{x})} \times \mathit{canvas\_height} \tag{2.2}$$

This function represents a *linear* scale. The scales could implement different types of transformations, and allow the user of the grammar to create visualizations with, for example, logarithmic axes.

The *axes* and *legends* in a graphic are *guides* that allow the user to map the visual channels back to the data domain visually. For example, the legend allows the viewer to see the meaning of each color.

Ggplot2 tries to infer the types of the variables and choose sensible default scales automatically. The user can change the scales, but the defaults allow for succinct specification for common cases.

Facet

Sometimes it may be practical to partition the data by some categorical variable, and display the partitions juxtaposed side-by-side. Such a design is known as *small multiples* and the partitioning as a method is called *faceting* [26]. For example, in a scatter plot, observations belonging to multiple classes could be separated by a color. However, each class of observations could alternatively be displayed as a separate scatter plot side-by-side with the other classes. Figure 2.10 displays how our simplistic example dataset could be faceted with ggplot2.

2.4.2 Vega-Lite

Vega-Lite [33] is a high-level grammar of interactive graphics. The grammar allows for succinct visualization specifications, facilitating quick exploration of the design space. The Vega-Lite library compiles the specification to Reactive Vega [34], a lower level but more expressive grammar. Interactive Vega-Lite visualizations can be embedded

```r
ggplot(data = treatments) +
  geom_col(aes(x = "",
               y = result)) +
  facet_grid(cols = vars(patient),
             rows = vars(treatment))
```

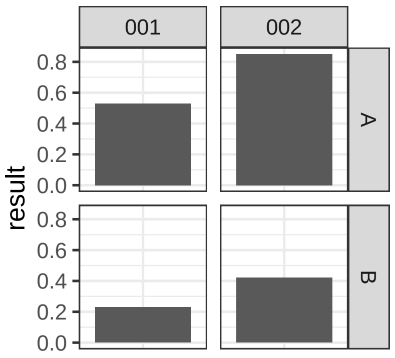


Figure 2.10: Faceting by two variables. The dataset has been partitioned into four parts: each patient-treatment pair forms a subplot that contains a bar chart, albeit with just a single bar.

on web pages and applications. This section covers the central concepts of Vega-Lite, but omits irrelevant parts, such as interactivity.

Syntax

The listing in Figure 2.11 displays an example of a Vega-Lite view specification. The specification is a JavaScript Object Notation (JSON) document. Curly brackets denote an object (dictionary), square brackets denote arrays (list). The JSON specification is more verbose than the equivalent ggplot2 code (Figure 2.9), but the syntax allows for easier programmatic parsing and production.

Data input

Vega-Lite loads the data from a URL (line 3 in Figure 2.11) as delimited (CSV, TSV) or JSON format. Alternatively, the data can be embedded in the specification as a JSON array. Vega-Lite also includes sequence generators, which create numerical sequence data of arbitrary length.

Vega-Lite infers the file format from the file extension. However, in bioinformatics, tabular data is commonly stored as tab-separated values, but the CSV file extension is (incorrectly) used. In such cases, the user can specify the file format explicitly (line 4 in Figure 2.11).

Marks and Channels

Marks in Vega-Lite analogous to geometries in ggplot2. However, their parameterization has some differences. For instance, although the `rect` mark allows for creating arbitrarily sized rectangles, Vega-Lite uses categorical positional scales to compute their widths and lengths automatically.

```
 1 {
 2   "data": {
 3     "url": "treatments.csv",
 4     "format": { "type": "tsv" }
 5   },
 6   "encoding": {
 7     "x": { "field": "patient", "type": "ordinal" },
 8     "y": { "field": "treatment", "type": "ordinal" }
 9   },
10   "layer": [
11     {
12       "mark": "rect",
13       "encoding": {
14         "color": { "field": "result", "type": "quantitative" }
15       }
16     },
17     {
18       "mark": "text",
19       "encoding": {
20         "text": { "field": "result", "type": "quantitative" }
21       }
22     }
23   ]
24 }
```

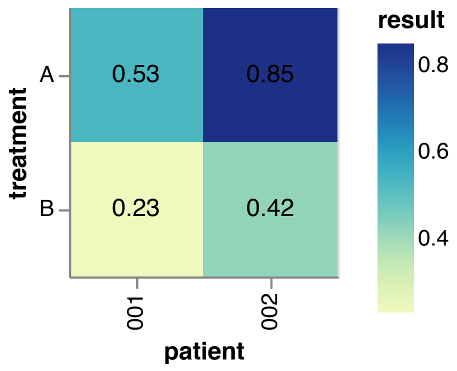


Figure 2.11: Creating a heatmap with Vega-Lite, replicating the ggplot2 example in Figure 2.9. The hierarchical specification consists of a composite *layer view* (line 1) and two *unit views* (lines 11 and 17) that have associated *marks*. The unit views inherit the encodings and data from their parent (the layer view). The Vega-Lite compiler uses rule-based defaults for scales. For example, a `rect` mark with a quantitative color field results in a green-blue color scheme.

Vega-Lite provides a number of marks, some of which are: `rect`, `text`, `point`, `line`, and `arc`. All the marks support a basic set of visual *channels* (equivalent to aesthetic in ggplot2), which represent the mapping between the visual properties of the marks and the data fields. Some marks have specific channels: the `text` mark has a `text` channel. A channel can also be assigned a constant visual value such as a specific named color.

Vega-Lite expects the data type to be specified explicitly (line 7 in Figure 2.11 for each data field to avoid ambiguity. For instance, a string of digits may represent a patient code or a quantity. However, the former is categorical data, and must be presented accordingly.

Scales

By default, Vega-Lite uses rule-based defaults for scales. The field type, visual channel, and mark are taken into account. The data domain is automatically extracted from the data. However, the user can change the scale function and adjust the domains and ranges. The scales are categorized as *continuous* (e.g., linear, pow, log), *discrete* (e.g., ordinal), and *discretizing* (e.g., quantile, threshold). Listing 1 displays an example of a custom scale.

```
{
  ...,
  "color": {
    "field": "segMean",
    "type": "quantitative",
    "scale": {
      "type": "threshold",
      "domain": [0],
      "range": ["blue", "red"]
    }
  }
}
```

Listing 1: A custom scale in Vega-Lite. The `color` channel uses a `threshold` scale. Values below zero are shown as blue, values equal to or greater than zero are shown as red.

Data Transforms and Expressions

With transforms, the users can filter the dataset or derive new data. Ggplot2 does not provide any transforms by itself because it is integrated into a programming language. Vega-Lite, however, provides a number of transform operations that allow for manipulating the data prior to visual encoding. Listing 2 displays a chain of transforms with Vega-Lite.

```
{
    "data": { ... },
    "transform": [
        { "calculate": "2*datum.b", "as": "b2" },
        { "filter": "datum.b2 > 60" }
    ],
    ...
}
```

Listing 2: A chain of transforms in Vega-Lite. First, a `calculate` transform uses an *expression* to compute a new value, which is introduced as new field `b2`. Next, only the data items with $b2 > 60$ are retained. Vega-Lite infers the intended transform operations from the object property names.

Hierarchical Composition

Ggplot2 improved the Grammar of Graphics by introducing layers. Vega-Lite takes a step further with the concept of *hierarchical view composition*. Instead of only allowing for a flat list of superimposed layers, Vega-Lite provides several composition operators, including *layering*, *concatenation*, and *repeating*. The operators can be combined hierarchically. For example, multiple layered views can be concatenated to create a dashboard.

Data propagates down the hierarchy and can be transformed at any node. Thus, the architecture allows for reusing, deriving, or replacing the data at any node. Depending on explicit configuration or rule-based defaults, data domains and scales are pulled towards the root and merged, ensuring concordant visual encoding.

The specification in Figure 2.11 has a simple view hierarchy, which contains a single composition operator. Figure 4.7 displays a higher hierarchy.

Facet

Vega-Lite provides a specific `facet` composition operator for faceting. Alternatively, `facet`, `row`, or `column` channels can be used.

Architecture

Vega-Lite is a JavaScript library, which uses Reactive Vega, a lower-level library, to process the data and render the graphics. The architecture can be outlined as follows:

1. **Vega-Lite** parses the visualization specification into a data structure
2. The compiler applies rule-based defaults to ambiguities
3. The compiler performs optimizations and creates a Reactive Vega specification
4. **Reactive Vega** parses the specification and creates a data flow graph
5. The data flow graph is executed, and data is read, and it is transformed into a scene graph
6. The scene graph is rendered using the HTML Canvas or transformed into SVG graphics

Interactivity is handled by the Reactive Vega runtime. When the visualization is altered by interaction, the runtime updates parts of the scene graph and subsequently re-renders the graphics.

2.5 GPU Rendering

Modern graphics processing units (GPUs) have a programmable pipeline, which can parallelize the computation to hundreds of processor cores [3]. However, the computation power comes with a cost – its programming model is more difficult than traditional, imperative "pen-and-plotter" models (Listing 3).

```javascript
var ctx = canvas.getContext('2d');

// Draw a filled rectangle
ctx.fillRect(25, 25, 100, 100);

// Draw a path
ctx.beginPath();
ctx.moveTo(75, 50);
ctx.lineTo(100, 75);
ctx.lineTo(100, 25);
ctx.stroke();
```

Listing 3: "Pen-and-plotter" drawing with the HTML canvas API. The programmer issues drawing calls that are performed sequentially. The model is straightforward for the programmer, but difficult to parallelize.

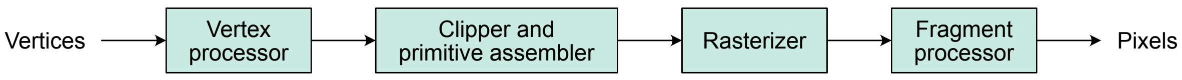


Figure 2.12: Geometric pipeline. Adapted from [3].

While pen-and-plotter application programming interfaces (APIs) provide high-level drawing commands, GPU APIs are low level – they allow the programmer to upload floating-point arrays to the GPU memory, compile *shader* programs, and issue drawing commands that render graphical primitives (triangles, lines, and points) defined by the vertices.

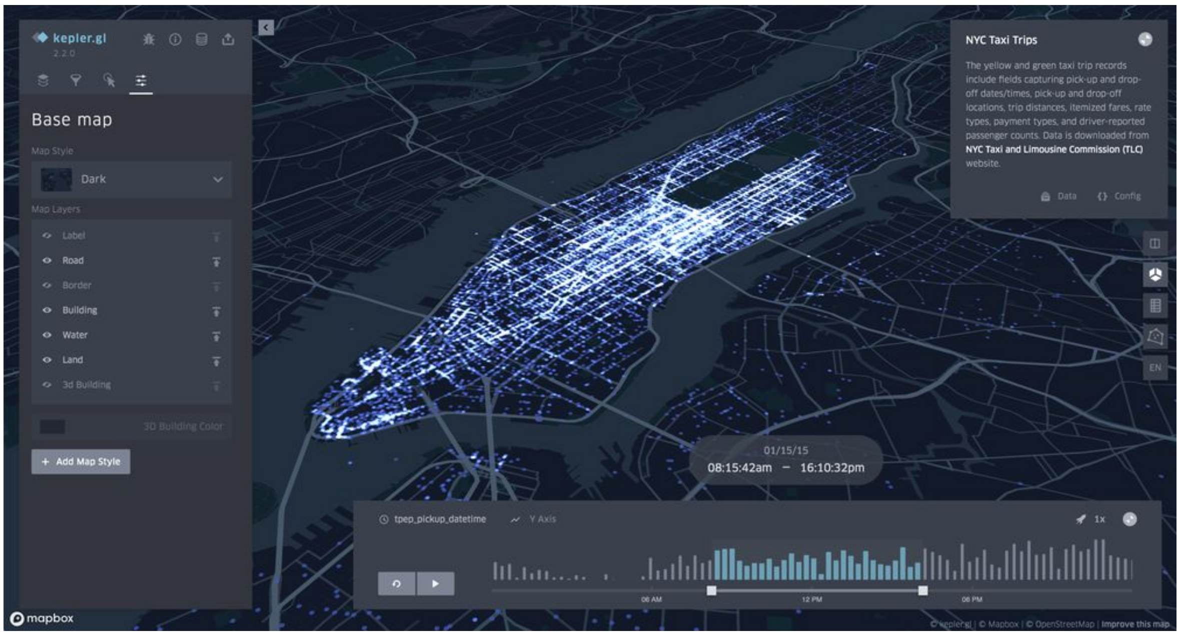
Figure 2.12 displays a schematic diagram of a GPU pipeline. The pipeline inputs *vertices* as vectors. *Vertex processor* does geometric transformations independently for each vertex. *Clipping and Primitive Assembly* constructs primitives that fall into the clipping volume (viewport). *Rasterizer* turns the primitives into fragments ("potential" pixels), which are rendered as colorized pixels by the *Fragment Processor*. Both the vertex and fragment processor are fully programmable with *shader* programs. Thus, they are not limited to specific geometric transformations or predefined texture mapping methods. Instead, they can be used flexibly and even employed for general-purpose computing on GPU (GPGPU). Because each vertex and fragment are computed independently, and the computations mainly consist of vector and matrix operations, they parallelize effectively, allowing for high performance graphics rendering.

WebGL is a GPU API available in web browsers. It allows for transferring data between the application and the GPU and orchestrating the rendering pipeline. Because of its complexity and low-level nature, it is not covered in more detail here.

2.5.1 State-of-the-art Examples

Kepler.gl [1] (Figure 2.13) is a WebGL-powered spatial data analysis tool. It has some similarities to Vega-Lite – for instance, the user can choose how data are mapped to different visual channels of the provided layers (analogous to the graphical marks) and which color schemes to use. The data can be filtered and aggregated interactively. Kepler.gl provides basic interactions such as panning, zooming, and details on demand by hovering with the mouse cursor.

Stardust [30] is a WebGL visualization library, which mimics the API of the widely used d3 [5] library. The API allows JavaScript programmers to create interactive visualization from components such as marks and scales. The marks and scales are compiled to shader programs for efficient rendering. P4 [22] develops the GPU usage further, extending it to data processing (transforms), providing efficient aggregation functionality. It also introduces a visualization grammar with JSON syntax. DXR



NYC Taxi Trips

The yellow and green taxi trip records include fields capturing pick-up and drop-off dates/times, pick-up and drop-off locations, trip distances, itemized fares, rate types, payment types, and driver-reported passenger counts. Data is downloaded from **NYC Taxi and Limousine Commission (TLC)** website.

Data {} Config

Figure 2.13: kepler.gl, a geospatial data visualization tool

[38] implements part of Vega-Lite grammar and renders the visualization specification using the GPU-accelerated Unity (https://unity.com) game engine. All of the three are mainly research projects and not used widely.

2.6 Fluid Interactions

Advancements in GPU performance and multi-touch displays have contributed to the development of user interfaces that are responsive, aesthetic, and enjoyable to use. That is especially apparent in consumer-oriented mobile phone applications. Elmqvist et al. introduced the concept of *fluid interactions*, which helps to characterize such user interfaces in the context of information visualization:

“ Fluidity in information visualization is an elusive and intangible concept characterized by smooth, seamless, and powerful interaction; responsive, interactive and rapidly updated graphics; and careful, conscientious, and comprehensive user experiences. ” ([10])

They hypothesize that comprehensive expression of fluidity in an information visualization tool makes the sense-making process enjoyable and insightful, helping the user stay in the flow of exploration. To promote further research and help designers to build more efficient information visualization tools, they compiled a set of design guidelines for fluidity. I apply the guidelines in this work.

3. Design and Architecture of GenomeSpy

This chapter presents the rationale, design, architecture, and some implementation details of GenomeSpy, a grammar-based genome visualization tool that I have developed.

The most important interaction method in genome browsers is navigation by zooming and panning. Navigation allows the user to change the viewpoint in a large and complex dataset [26], transitioning from an overview to details and back. On the other hand, investigators work with large cohorts of patients and samples, trying to find patterns by filtering, grouping, and sorting. GenomeSpy provides interactions that allow the visualization end-users to perform these tasks with pleasure, immersing themselves into the flow of exploration. The interactions comprise of navigation by zooming and panning, and manipulating the sample set by sorting and filtering. The navigation interactions are facilitated by two zoom behaviors that help in managing overplotting.

What is being visualized in GenomeSpy and how it is encoded into a graphical representation is up to the *visualization designer*. The designer may be a data scientist creating an interactive website for the scientific community, or it may be the *end-user*, such as a bioinformatician validating their analysis results. GenomeSpy provides a grammar and a set of combinatorial blocks for exploring the design space and implementing new visualizations. A *visualization specification* defines how the data is presented to the end-user. The visualizations are not limited to specific file formats that are displayed in a specific, sometimes suboptimal way. Instead, designers can focus on the data, the essential attributes, and conceive a design that is able to present the data in an adequate way. However, it makes no sense always to create new designs. For common cases, the designs can be reused and even packaged into components that can be imported to other GenomeSpy visualizations.

GenomeSpy provides the interactions, and the grammar allows for specifying what is being visualized. However, instead of inventing yet another visualization grammar, GenomeSpy is heavily inspired by Vega-Lite (subsection 2.4.2). Providing users with a familiar grammar lowers the threshold to start using a new tool. On the other hand, designing a concise yet extendable grammar is difficult. Although Vega-Lite is less popular than ggplot2, it has characteristics that make it particularly suitable for genomic visualization – perhaps not out of the box, but through extensions. Its main strengths are independence of a programming language and the powerful view composition algebra. However, the architecture of Vega-Lite, as a software package, makes it difficult to apply to interactive genome-scale visualizations. The most severe obstacle is the performance with large datasets. Vega-Lite supports up to some tens of thousands of data items, but millions or more are required in genomic visualizations. On the other hand, while Vega-Lite allows for specifying visualizations with sophisticated interactions, it does not adapt to the interactions that I have designed for GenomeSpy. Thus, extending the Vega-Lite software was not a feasible choice. Instead, GenomeSpy is an independent implementation of a subset of Vega-Lite's visualization grammar. GenomeSpy also extends the grammar to better support common genome visualization needs.

GenomeSpy exploits GPU in rendering to implement the interaction design. GPU-programming is difficult if compared to immediate mode graphics APIs such as HTML Canvas. The grammar provides an abstraction that allows visualization designers and application developers to exploit the GPU declaratively in their visualizations, without having to get acquainted with low-level graphics APIs.

Sections 3.2 and 3.3 focus mainly on the visualization grammar. Focus of sections 3.4 and 3.5 is more on the interaction design, which consists of handling multiple samples and navigation around the genome.

3.1 JavaScript Library

GenomeSpy is not an application that can be opened by choosing an item from the start menu or by browsing to a WWW-address. Instead, it is a JavaScript software library that can be used as a building block for other applications. Section 4.2 presents a GenomeSpy-based application that allows users to visualize their own, albeit very specific type of data. Section 4.3 describes a use case where a GenomeSpy visualization has been embedded on a web page for others to consume. The application programming interface (API) of GenomeSpy is not discussed in this thesis, as it is available in the documentation.

GenomeSpy itself uses a few other libraries. Parts of D3 [5] are used for data processing. The expression language, scales, and data loading employ Vega [34] libraries. A Tiny WebGL helper Library (TWGL, `https://twgljs.org/`) is used for making the WebGL API less verbose.

3.2 The Grammar

As the visualization grammar of GenomeSpy is heavily inspired by Vega-Lite many of the concepts have already been introduced in Subsection 2.4.2. This section briefly describes the building blocks (data input, marks, view composition) the grammar provides. However, some parts, such as scales and expressions, are omitted because they are virtually identical to their Vega-Lite counterparts.

3.2.1 Data Input

Genome browsers generally input specific file formats such as BED or bigWig. However, GenomeSpy only supports generic delimited formats such as comma-separated values (CSV), tab-separated values (TSV), or specially crafted JavaScript object notation (JSON) files. The reasons are twofold. The focus of GenomeSpy has been on visualizing custom data for which no standard formats exist. On the other hand, the specific file formats, such as BED, can be easily converted to the generic delimited formats.

Currently, the data input is identical to Vega-Lite. However, it can be later extended with custom loaders that automatically convert the more specific formats into the internal tabular layout, streamlining the usage.

3.2.2 Graphical Marks

GenomeSpy supports three graphical marks, namely the *point*, *rect*, and *rule*. All the marks support the basic channel set: Positional (`x`, `y`), `color`, and `opacity`. The marks are presented below.

Point

The point mark displays each data item as a symbol with optional x and y coordinates. Points are often used in scatter plots to display the relationship of two variables using two-dimensional Cartesian coordinates.

The size of a point does not generally depend on the mapping between the data domain and the range of the x or y channels (for an exception, see Section 3.5). Instead, the size may be constant, or it can be mapped to a data field. The size in pixels is defined as the area of the symbol's bounding rectangle.

Figure 3.1 displays an array of points with varying visual channels. However, some of the channels are most useful for stylistic purposes and should not be used for visual encoding. For instance, `gradientStrength` controls the amount of the radial color gradient inside the symbols. Although it could be mapped to a data field, `color`

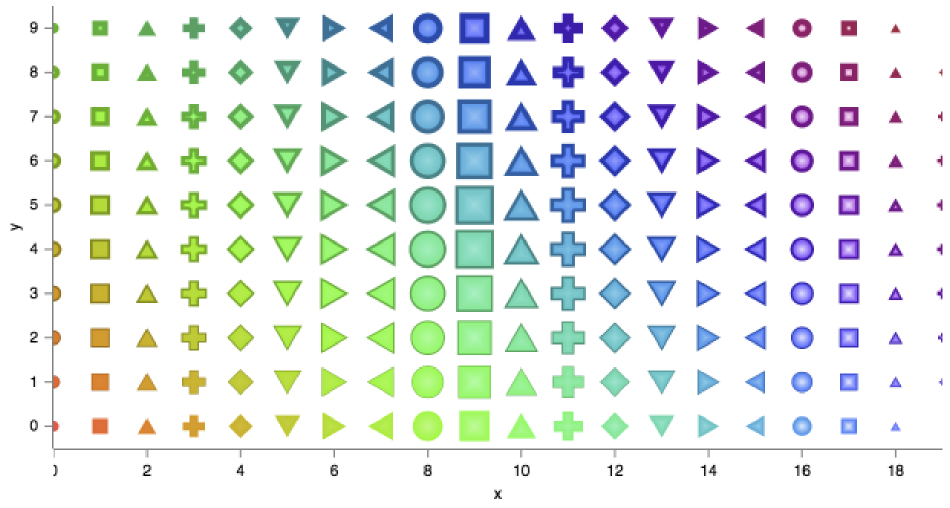


Figure 3.1: An array of points demonstrating multiple visual channels. In addition to the positional `x` and `y` channels, `size`, `shape`, and `gradientStrength` vary horizontally, `strokeWidth` varies vertically, and `color` varies diagonally.

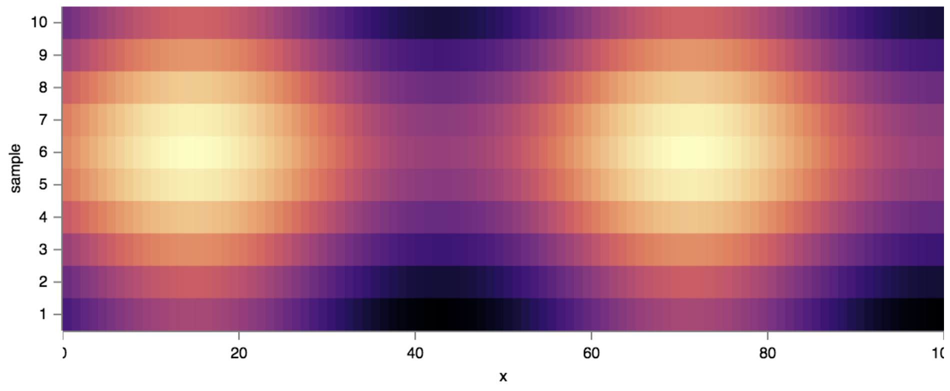


Figure 3.2: A heatmap built with *rect* marks.

would be better choice, as it is perceptually more effective.

Rect

The rect mark displays each data item as a rectangle. Because rectangles have a width and a height in addition to their position, they can be used flexibly to build various visualization such as heatmaps (Figure 3.2) and bar charts (Figure 3.3). To allow for defining the width and height, rect mark supports two additional positional channels: `x2` and `y2`.

Rule

Rule mark displays a data item as a horizontal or vertical line with optional endpoints. Although one could create such lines with the *rect* mark, its width or height would

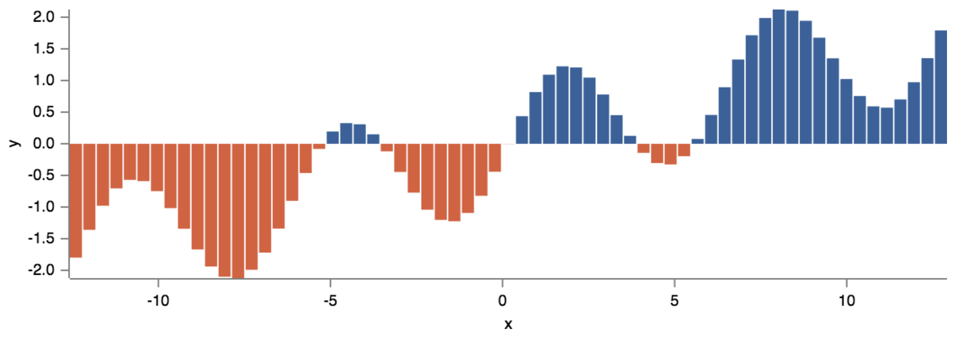


Figure 3.3: A bar chart built with *rect* marks.

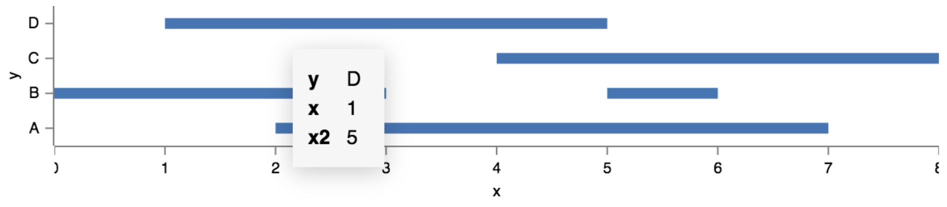


Figure 3.4: An example of *rule* marks. Their vertical position is mapped to a nominal *y* field and the endpoints to a quantitative *y* field. The stroke width has been defined in pixels. When a data item in GenomeSpy is hovered with the mouse cursor, a tooltip reveals all its attributes, including those that are not mapped to any visual channel.

be specified using the data domain. Rule, however, allows for specifying the width as pixels. This distinction is specifically important when the visualization is zoomed. With the rule mark, the line width stays constant, whereas with the rect mark, it would change. Figure 3.4 displays an example of rule marks.

3.2.3 Data Transforms

The transform grammar of GenomeSpy is based on Vega instead of Vega-Lite. While Vega-Lite is more succinct (Listing 2), Vega is easier to extend because the transform type is defined explicitly.

GenomeSpy currently supports a basic set a transforms: `filter` filters rows using an expression, `formula` derives new field using an expression, `stack` computes a stacked layout, `gather` pivots data to tidy format, `regexExtract` derives new fields using a regular expression, and `flattenDelimited` flattens field that contain delimited values, i.e. creates new rows for them. The first three are identical to their Vega counterparts, the rest are specific to GenomeSpy. Transforms in GenomeSpy are discussed in more depth in the sample-comparison case study in Section 4.3.

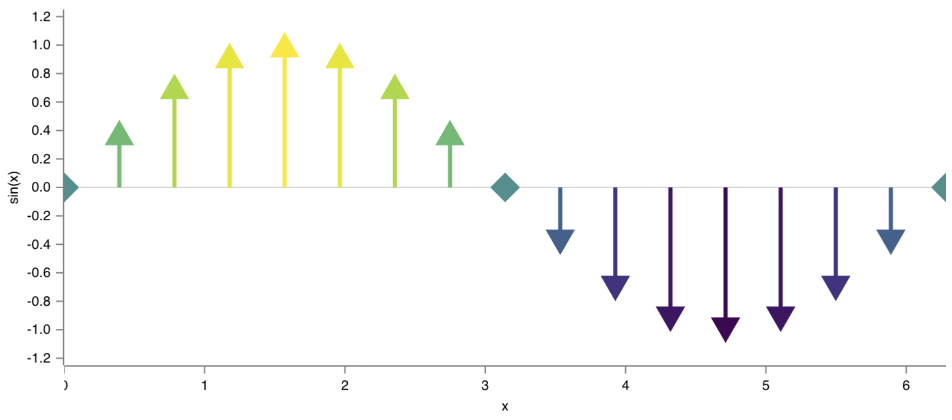


Figure 3.5: View composition by layering. The arrows consist of layered rules and points with various symbols. The x axis ticks are implicitly added using concatenation. The view specification for this example can be found in Section A.1.

3.2.4 View Composition

GenomeSpy partially replicates the *hierarchical view composition* of Vega-Lite. However, only layering and vertical concatenation (analogous to tracks) are supported. As the data and encodings propagate down the tree, the specification stays succinct. The hierarchy also allows for specifying whether scales should be shared among the views or not.

In addition to just displaying multiple datasets, visualization designers can use layering to create new composite marks. For example, Figure 3.5 displays a "lollipop plot", which consists of layered vertical rules and points. Lollipops are often used in genomic visualizations to display variants [21]. The potential of hierarchical view composition is explored more thoroughly in the case study in section 4.2.

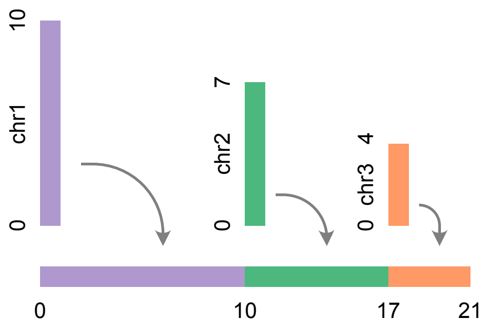


Figure 3.6: Concatenating the discrete chromosomes to create a contiguous, linear coordinate system.

3.3 Supporting Genomic Data

This section describes how GenomeSpy extends the visualization grammar to better support genomic data.

3.3.1 Concatenated Genomic axis

Most genome browsers handle chromosomes as discrete units, i.e., the user has first to choose which chromosome they would like to explore. For instance, IGV has a separate view that displays the whole genome by concatenating (Figure 3.6) all the chromosomes onto the horizontal axis. However, the view is not freely zoomable, and during exploration, the user has to jump between the whole-genome view and individual chromosomes.

GenomeSpy allows for continuous zooming from the whole-genome view to the nucleotide level. Technically, it uses an approach similar to HiGlass': the chromosome-position pairs are mapped to coordinates on a single concatenated x-axis. Although the mapping would be straightforward to implement as a separate `transform` step, such an approach would complicate trivial cases excessively. Instead, GenomeSpy extends the grammar by providing a shorthand in the `encoding` block (Listing 4).

3.3.2 Built-in Annotation Tracks

GenomeSpy provides several built-in annotation views (tracks) that the user can import into their visualization. They are currently implemented in an ad-hoc manner, disregarding the visualization grammar as it needs to be first developed further. For example, the *text* mark has not been implemented yet. The built-in views can be imported using the `import` directive as shown in Listing 5. The import mechanism also allows for including external specification in the visualization, promoting reuse.

```
{
    "genome": { "name": "hg38" },
    ...,
    "encoding": {
        "x": {
            "chrom": "Chr",
            "pos": "Pos",
            "offset": -1.0,
            "type": "quantitative"
        },
        ...
    }
}
```

Listing 4: Specifying the coordinate mapping in the `encoding` block. The `genome` property specifies the genome assembly to use for coordinate transformation. The `offset` property allows for adjusting for different interval indexing schemes.

```
{
  "genome": { "name": "hg38" },
  "concat": [
    { "import": { "name": "cytobands" } },
    { "import": { "name": "geneAnnotation" } },
    { "import": { "name": "genomeAxis" } }
  ]
}
```

Listing 5: An example of a view specification with all three built-in annotation tracks. The resulting visualization can be seen in Figure 3.7 and Figure 3.8. The `import` directive can also be used for importing track specifications from local files or remote URLs.

Gene Annotations

The `geneAnnotation` track displays RefSeq [28] gene annotations. The user first sees only the gene symbols*, but transcription directions, gene bodies, and exons become visible as the user zooms closer (Figure 3.7 and Figure 3.8).

GenomeSpy uses the citation-count-based prioritization introduced in HiGlass to display only the most important genes at each zoom level (Subsection 2.3.3). Thus, in the the whole-genome view, the best-known genes act as landmarks aiding the user in navigation. As the user zooms closer, less-known genes become gradually visible.

*Gene symbols are short identifiers for genes, usually abbreviations

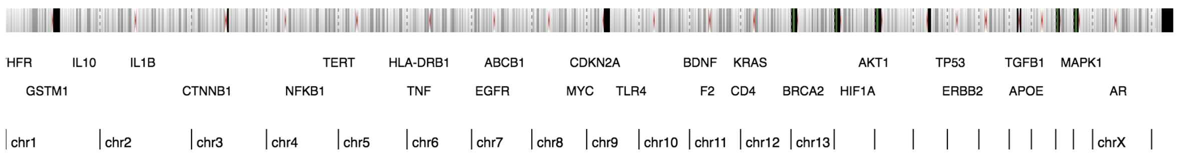


Figure 3.7: An example of the built-in annotation tracks. The top track displays the chromosome bands (cytobands). The middle track displays RefSeq gene annotations. The bottom track displays chromosome-aware coordinate ticks.

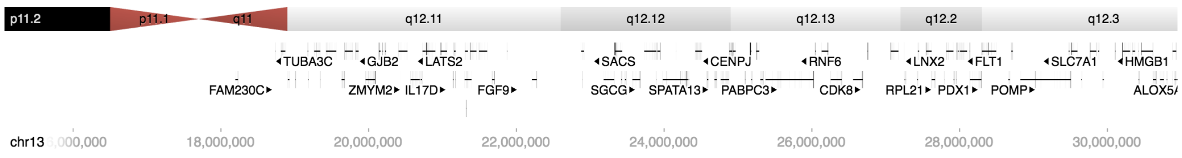


Figure 3.8: The same tracks as in Figure 3.7 but zoomed into the chromosome 13. The gene annotation track now displays the transcription directions, gene bodies, and exons. The axis track displays intra-chromosomal coordinates.

Chromosome-aware Axis Ticks

The `genomeAxis` track displays the chromosome names and boundaries. When the user zooms in, the intra-chromosomal coordinate ticks become visible.

Cytobands

The `cytobands` track displays the chromosomal bands (subsection 2.1.1) that aid in navigation around the genome.

3.4 Faceting Multiple Samples

One of the motivations in the development of GenomeSpy was to allow for exploring and comparing multiple samples, e.g., multiple patients or treatment outcomes. Although genome browsers generally support multiple tracks, they rarely provide powerful tools for manipulating a large set of samples by filtering and sorting. The IGV (Subsection 2.3.2) and the University of California Santa Cruz XENA [12] are the two most competent tools for handling multiple samples. However, their visualization capabilities are limited.

Support for multiple samples in GenomeSpy builds upon the *facet* operation of existing visualization grammars, combines it with familiar concepts from IGV, and equips it with *fluid interactions* [10] for smooth and engaging user experience. The next subsections discuss how faceting is specified in GenomeSpy and how the user can interactively manipulate the sample set.

3.4.1 Facet View

Although the grammar of GenomeSpy tries to follow Vega-Lite faithfully, faceting is an exception because:

" When faceting a composite view, only the dataset targeted by the operator is partitioned; any other datasets specified in sub-views are replicated. " (Satyanarayan et al. in [33])

In other words, Vega-Lite does not allow for creating layered views that display multiple types of data. An example of such a view is copy-number variation overlaid with point mutations. See the case study in Section 4.3 for further details.

A faceted layout in GenomeSpy is activated by defining the `sample` channel in the `encoding` block (Listing 7). The `sample` channel of GenomeSpy is analogous to the `row` channel of Vega-Lite — each subset is displayed as a row (Figure 3.9). However, in GenomeSpy, a special type of track gathers the sample identifiers from the view hierarchy and creates an own virtual subtrack for each sample. The view hierarchy may contain multiple datasets representing different types of data. This design does not support nested faceting, but on the other hand, allows for faceted layering of multiple datasets.

```
{
    ...,
    "encoding": {
        ...,
        "sample": {
            "field": "sampleId",
            "type": "nominal"
        }
    }
}
```

Listing 6: Specifying faceting using the `sample` channel.

3.4.2 Sample Attributes as a Metadata Heatmap

GenomeSpy supports sample attributes (metadata) with a visual representation similar to the IGV (Figure 3.9). The data source and optional scales for the attributes can be specified as shown in Listing 7.

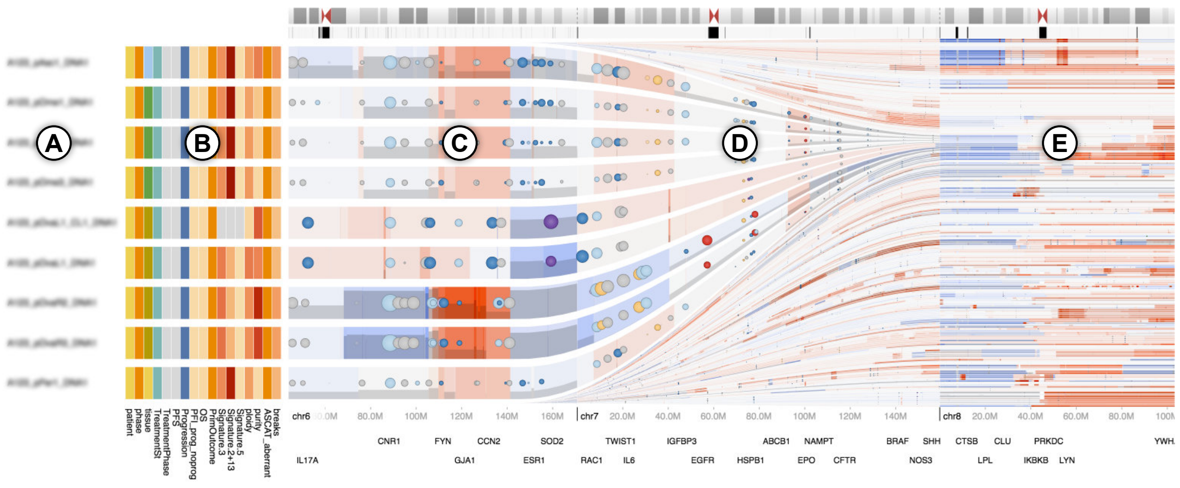
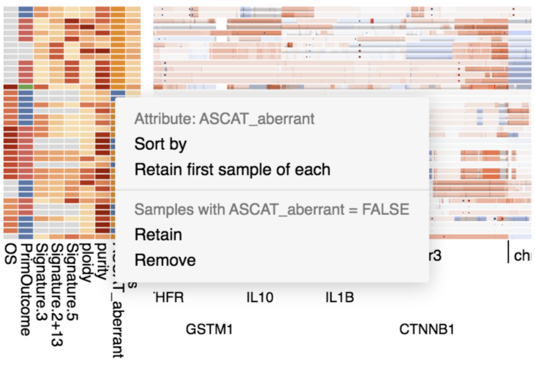
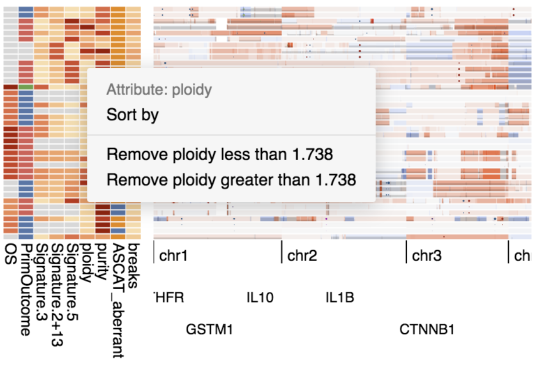


Figure 3.9: A faceted visualization as the user focuses on a subset of samples. (A) The dataset has been partitioned into facets using the sample identifiers. (B) A heatmap displays the metadata. (C) A layered visualization design displays the genomic data. (D) A smoothly animated transition wave reveals the new set of samples as it travels through the view. (E) The original sample set that the user is filtering disappears under the wave. This visualization is discussed in more detail in Section 4.3.



(a) Nominal attribute



(b) Quantitative attribute

Figure 3.10: Context menus for different types of sample attributes.

3.4.3 Sorting and Filtering Samples

Vega-Lite is not only a grammar of graphics but a grammar of *interactive* graphics [33]. However, while the grammar of GenomeSpy tries to follow Vega-Lite, it has a much narrower scope and more specific interaction requirements. Thus, implementing generic and abstract support for interactivity in GenomeSpy is not practical. Instead, it provides a predefined set of interactions, such as sorting and filtering, and does not support the declarative interactivity features of Vega-Lite.

GenomeSpy allows the user to sort and filter the samples (facets) by their metadata or actual data, such as copy-ratios. The user can, for example, focus on a specific patient, a mutational signature, or both of them.

```
{
    "samples": {
        "data": { "url": "samples.tsv" },
        "attributes": {
            "OS": {
                "type": "ordinal",
                "scale": {
                    "domain": ["<=6", "6 - 12", "12 - 24", ">100"],
                    "scheme": "orangered"
                }
            }, ...
```

Listing 7: Including metadata in a faceted view with the `samples` property. The data file must contain a `sample` field, which identifies the sample, and any number of attributes. By default, GenomeSpy auto-infers the data type and uses a discrete or continuous color scheme accordingly. The user can override the inferred data types and domains and specify a custom scheme. In this example, the *OS* (overall survival) is declared as *ordinal* and the values are enumerated in the correct order.

The user interface and interaction design follow the guidelines of *fluid interactions* [10] to provide a user experience that makes the sense-making process illuminating and joyful, letting the user immerse into the flow of exploration. For instance, all transitions between the sets of samples are animated smoothly (Figure 3.9), providing immediate feedback that helps the user understand the relationship between the previous and current views [16]. The user can clearly see how the set of visible samples changes when it is filtered or sorted: the transition is animated as a wave that travels from left to right, gradually revealing the new configuration. The interface supports *direct manipulation* [37] to minimize indirection: the user can open a context-menu (Figure 3.10) by right-clicking an intersection of a sample and attribute in the heatmap. The context menu allows the user to choose actions such as removing samples with a specific property. All the actions are reversible. Thus, the user can perform multiple actions, backtrack a step or two, and explore another path.

The interaction design differs significantly from the IGV, which requires, for example, the user to open a dialog for entering the filtering rules (Figure 2.4). Moreover, the IGV does not have the backtrack functionality, nor are the transitions animated.

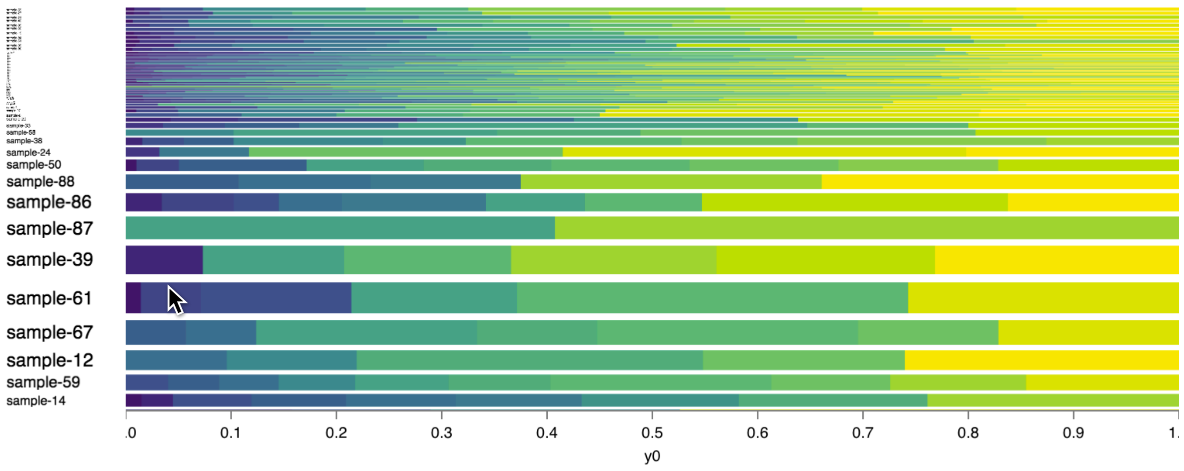


Figure 3.11: The fisheye tool. The focal point of the distortion follows the mouse cursor, allowing the user to focus on individual samples quickly. The samples outside the distorted area provide context and may display patterns spanning the whole sample set.

3.4.4 Focus+Context with "Fisheye" and "Peek"

When an investigator compares multiple samples, they may wish to observe patterns that span the whole set. On the other hand, it may be that the few outliers are the focus of interest. Genome browsers typically have a scrollable viewport and an option to adjust the height of individual tracks (samples). Thus, the user can minimize the track heights to view all the samples, and conversely, increase the heights to see less cluttered visualization but with fewer samples. However, such an approach for toggling between the bird-eye and close-up views is burdensome in exploration.

GenomeSpy uses different approach; the faceted views (analogous to tracks) always fill all the available vertical space. Thus the user can always see the whole sample set and the spanning patterns. When the user filters the sample set, the smaller set again uses all the available space, revealing more details because the individual views are now higher. However, the user may not always want to filter the samples. Instead, they may just want to scan them and briefly focus on some interesting outliers visible in the mass of samples. To support such a task, GenomeSpy provides two interactions: the "fisheye" and "peek".

Fisheye is a method where the visualization is distorted geometrically to reveal more details about the focused area while the surrounding is preserved as the context [6, 46]. Often the fisheye is implemented as a two-dimensional distortion. However, in GenomeSpy, the fisheye is one-dimensional, as it enlarges the samples under and near the mouse cursor (Figure 3.11). The user can quickly check the interesting looking samples in more detail by just moving the mouse cursor on them. The fisheye can be activated from a toolbar. However, using a keyboard shortcut is more efficient as

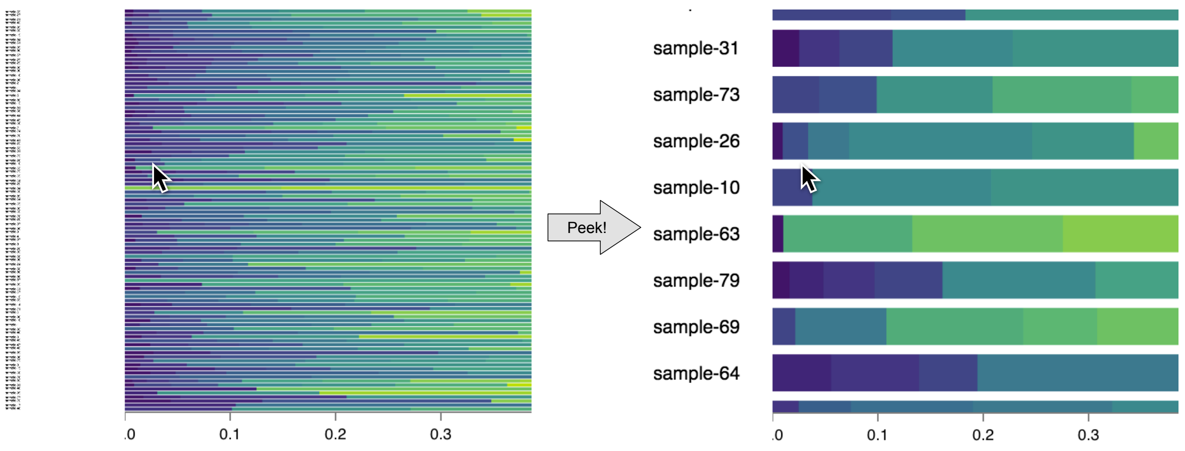


Figure 3.12: The peek tool. When the user activates peek, the viewport is zoomed vertically with a quick animated transition. The sample under the mouse cursor is preserved, helping the user stay focused. During an active peek, the user can scroll the viewport vertically with the mouse wheel.

the user does not need to move the cursor to the toolbar and back. Fisheye, however, has problems with target acquisition [6]. Interacting with individual graphical marks becomes gradually more difficult as the number of samples approaches several hundreds

"Peek", a transient vertical zoom, is an alternative to the fisheye. When the user activates peek with a keyboard shortcut, the view is zoomed vertically with a quick animated transition. The viewport is translated so that the sample under the cursor is preserved, helping the user maintain their focus (Figure 3.12). While normally the mouse wheel changes the zoom level, in an active peek it scrolls vertically. While such mode changes should be avoided [10], it allows the user to quickly scroll through thousands of samples and interact with graphical marks with high precision to get, for example, details on demand.

3.5 Avoiding Overplotting while Zooming

In genome browsers, the features are either point features representing single locus, or they are segment features representing an interval between two loci (Section 2.2). Point features are often visualized as graphical marks that have a constant size, i.e., they do not depend on the feature size or zoom level. Thus, they behave similarly to scatter plots. However, overplotting is a common issue in scatter plots: when the opacities of overlapping points add up and exceed the available dynamic range, information is lost [25].

Figure 3.13a displays a scatterplot of a bimodally distributed bivariate dataset. However, severe overplotting completely obfuscates the bimodal nature of the data. The problem can be mitigated by adjusting the opacity (Figure 3.13b) or point size

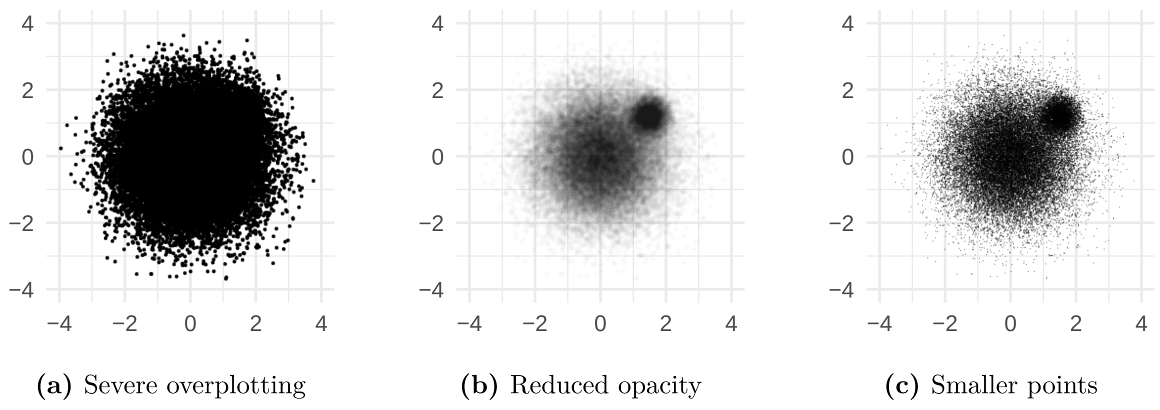


Figure 3.13: (a) Overplotting obfuscates the bimodal nature of the data. (b) Reduced opacity reveals the narrow peak, but the possible outliers are virtually invisible. (c) Reduced point size in behaves similarly to reduced opacity. However, interacting with pixel-sized points is difficult.

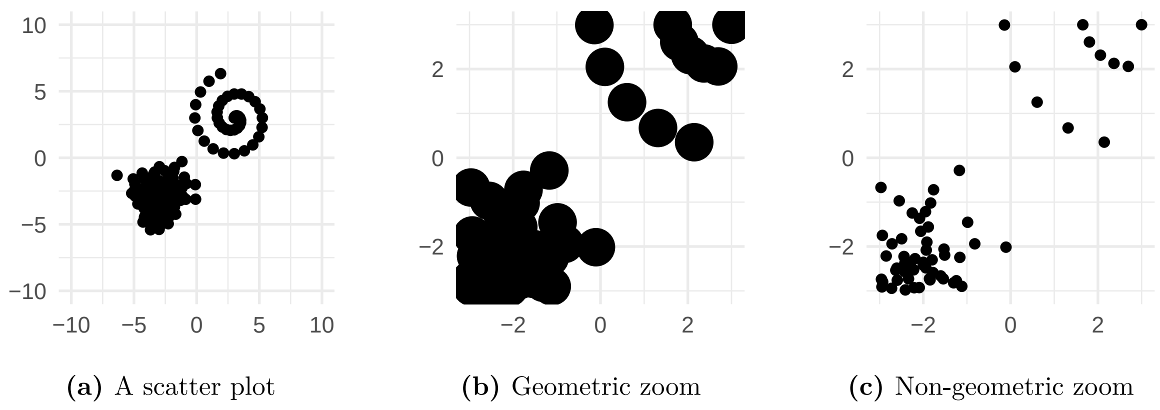


Figure 3.14: Geometric and non-geometric zoom. (a) A scatter plot with some generated data. (b) Geometric zoom changes the point size proportionally to the zoom level, behaving similarly to a camera that moves closer or further [26]. (c) In non-geometric zoom the point size stays constant. Thus, the closer the plot is zoomed, the sparser it becomes.

(Figure 3.13c) or by sampling the data. Finding an acceptable compromise requires trial and error, although cost-based optimization methods have been proposed [25]. Moreover, an interactive visualization with a zoomable viewport bears new challenges that are tightly intertwined with the overplotting problem. The next subsections present two methods that GenomeSpy provides for combating the overplotting challenges in a zoomable visualization.

3.5.1 Semi-geometric Zoom

Geometric zoom (Figure 3.14) resembles a camera that moves closer or further from the visualization [26]. Thus, the distances between objects and the diameters of the objects change in the same proportion. Also, the ratio between the background and object coverage stays constant. Conversely, in the non-geometric zoom, the diameters of objects stay constant, and the background-coverage ratio changes.

Point features are usually zoomed non-geometrically, but displaying thousands or even millions of points leads to severe overplotting. Commonly, genome browsers display data in the whole-genome view as binned or aggregated or prompt the user to zoom closer.

For use cases where the point features in a feature set are of equal importance, and their value is encoded as a position on the y axis, GenomeSpy provides semi-geometric zooming: a zoom-level dependent scaling factor is applied to the point size.

The zoom *level* is defined as $\frac{\text{domain width}}{\text{visible domain width}}$, which can be conveniently expressed as a power of two. For example, at 2^0 the whole genome is visible, at 2^{12}, $\frac{1}{1024}$ is visible.

The scaling factor (3.1) of the point diameter depends on the current zoom level ($level_{current}$) and the level where the points should reach ($level_{bound}$) their specified size. Because only one axis is zoomed, the square root ensures that the background-coverage ratio stays roughly constant – the **area** of the points is scaled proportionally to the zoom level of a single axis.

$$C = \sqrt{\min\left\{1, \frac{level_{current}}{level_{bound}}\right\}} \tag{3.1}$$

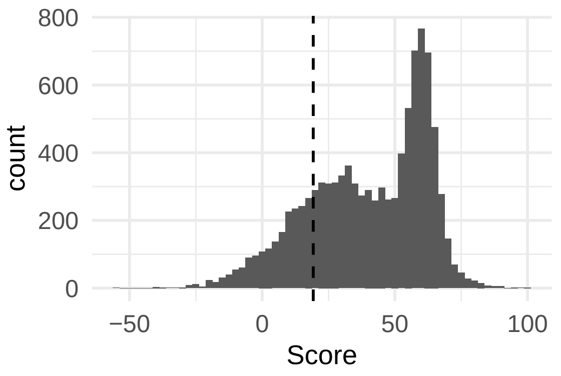
With semi-geometric zooming the visualization may display millions of points in the whole-genome view. As the user zooms in, the point size grows gradually until a specific point size is reached. Zooming even closer behaves like non-geometric zoom. Thus, the user can observe large-scale phenomena in the whole-genome view, quickly zoom into details, and easily interact with individual points. Section 4.2 presents such a scenario in practice.

3.5.2 Score-based Semantic Zoom

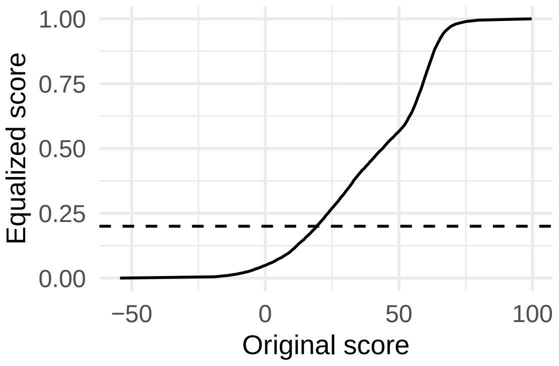
Strip plot is a specialization of scatter plot: it has just a single positional dimension. For instance, point mutations displayed as points on the x axis would qualify as a strip plot if their vertical position is constant.

Using semi-geometric zoom with a strip plot is unpractical because the points would shrink into an invisibly thin line when zoomed out. If the data items are not equally important, a better approach to reduce overplotting is displaying fewer points, only the most important ones. GenomeSpy provides *score-based semantic zooming* for such cases.

Let us assume that data items are uniformly distributed on the x axis. If some fraction of the data is randomly sampled, they are still uniformly distributed. When zooming closer and displaying only, for example, one fifth ($level_{zoom} = 5$) of the data



(a) Histogram of the original score



(b) Equalization function

Figure 3.15: Choosing the threshold score for semantic zoom by using quantiles. The dashed lines represent the equalized score 0.2 mentioned in the main text. (a) Displays a histogram of an arbitrary but non-uniform score. (b) The score can be equalized and normalized by computing p-quantiles. As the function is strictly monotonic, it is invertible, and the lookup can be performed in either direction.

but multiply the sampling fraction by the same number, the data are still uniformly distributed and, more importantly, have the same density. Thus, by making the sampling fraction inversely proportional to the zoom level, the visualization has always an equal number of uniformly distributed data. That rule, of course, breaks when the view is zoomed so close that there is not enough to sample.

If data have a score that is nearly uniformly distributed, it can be subset by the score instead of sampling randomly. For example, if the range of the score is $[0, 1]$, one fifth of the data can be "sampled" by choosing data that has $score < \frac{1}{5}$. However, scores are seldom uniformly distributed. In such cases, quantiles can be used instead: use the p-quantile 0.2 as the threshold (Figure 3.15). For simplicity, lower scores are assumed to be more important here. Thus, with an *equalized score* that has been quantile-transformed, a specific fraction of the most important data can be chosen.

By making the equalized score inversely proportional to the zoom level, GenomeSpy implements *semantic zoom*. Only the most important data are shown in the whole-genome view. As the user zooms in, less important data become gradually visible. Google Maps works similarly: in the whole-globe view, only the country names are visible. As the user zooms in, cities, towns, and villages appear. Zooming even closer reveals street names. Section 4.3 demonstrates how the score-based semantic zoom can be used with point mutations to reduce overplotting.

The semi-geometric and semantic zooms support efficient and enjoyable exploration. As the point sizes and filtering are connected to zooming, users can focus on the data without becoming interrupted to adjusts visualization and filtering parameters that reduce overplotting or make points large enough to be accessible.

3.6 Architecture

GenomeSpy is an independent implementation of Vega-Lite's grammar. However, the architecture differs considerably from Vega-Lite and Reactive Vega.

3.6.1 Initialization

The initialization process is divided into the following steps:

1. Read the JSON-formatted view specification
2. Process imports
3. Create an internal data structure for the view hierarchy, based on the specification
4. Resolve scales (shared or independent) and apply defaults
5. Load and parse data, execute the transform chains, extract data domains (if not specified explicitly)
6. Flatten the view hierarchy to tracks that may have layers
7. Apply scales and convert data items to geometric objects (points, rectangles), upload them to the GPU
8. Use WebGL to orchestrate the rendering process

Except for the horizontal genomic axis, GenomeSpy performs scale transformation before uploading the data to the GPU memory. Concatenated genomic coordinates are uploaded as-is and transformed in a vertex shader. Thus, zooming and panning along the genomic axis is efficient.

3.6.2 Rendering with WebGL

The scale-transformed data is translated into geometric objects. Each point mark instance translated into a single vertex. Rectangles translate into *triangle strips* that consist of adjacent triangles. Each rectangle requires at least six vertices. Long rectangles are tessellated into smaller triangles, allowing for smooth geometric distortions during animated transitions.

Point marks are rendered using signed-distance-field (SDF) primitives in a pixel shader [14]. The technique allows for crisp edges regardless of the symbol size. It also allows efficient computation of effects such as strokes and shadows.

WebGL is used for orchestrating the rendering process. The layers are rendered from bottom to top. Because the WebGL API imposes significant performance penalty to GPU draw calls, their number has been optimized to be as low as possible.

Emulating 64 Bit Floating-Point Numbers

Handling the over three billion bases long genome with WebGL requires special arrangements. Precise addressing of the genome requires 64-bit floating-point numbers, but shaders in WebGL only support 32-bit numbers. However, a 64-bit number can be emulated with two 32-bit numbers, albeit with substantial performance penalty [41]. On the other hand, because the computation can be limited to a linear transformation of a single dimension, the penalty is acceptable. GenomeSpy uses the emulation approach when the genomic axis is zoomed so close that the imprecision would cause visual artifacts.

4. Validation

This chapter demonstrates the utility of GenomeSpy with two case studies. In Case Study I, I use GenomeSpy as a software library to build an easy-to-use visualization tool for the assessment of copy-number segmentation results. The case highlights the expressivity of the visualization grammar and demonstrates the semi-geometric zooming. In Case Study II, I create an interactive visualization that allows for comparing hundreds of samples having multiple data dimensions. The case highlights the interactions in sample sorting and filtering and demonstrates the semantic zooming.

4.1 Data

Ovarian cancer is the seventh-most common and the eighth-most lethal cancer in women [11]. High-Grade Serous Ovarian Cancer (HGSOC) is the most common subtype, representing three-quarters of the cases, and its ten-year survival is less than 30% [20]. HGSOC is molecularly diverse and has few commonly recurring mutations other than the *TP53* tumor-suppressor gene [23]. The genomes are unstable, having large-scale amplifications and losses of genes [23].

The data consists of 375 HGSOC samples collected from 74 patients at diagnostic, mid-treatment, and relapse phases of the disease. The data were analyzed by other members of the research group except the copy-number segmentation in the Case Study I, which was analyzed by me. The analyses are explained to the extent that allows the reader to understand what is being visualized.

4.2 Case Study I: Copy-Number Segmentation Assessment

This case study validates the expressivity of the visualization grammar by replicating and improving an existing visualization design. The utility and performance of interactions, including the semi-geometric zoom, is demonstrated with a use case where the user explores a large dataset to find anomalies.

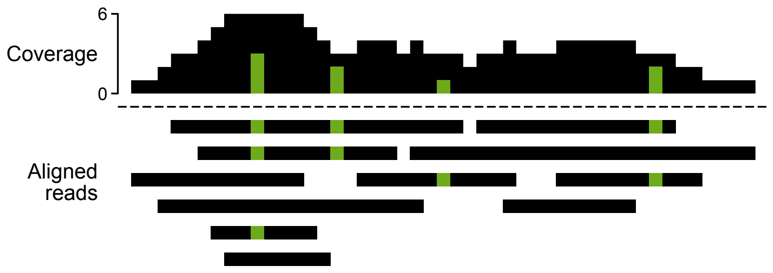


Figure 4.1: Next-generation sequencing platforms produce *reads*, short DNA sequences, which are aligned against a reference genome [43]. *Coverage* (or read count) means the number of reads at any given locus. The green boxes inside the reads portray *variants*, nucleotides that differ from the reference. The number of variants divided by the number of reads at a specific locus is called *variant allele frequency* (VAF) [39]. As the reads are essentially a random sample from the genome, the coverage varies along its length. The layout in the bottom part of the figure is called *pileup*, and is used in many genome browsers to visualize alignment results.

Genome Analysis Toolkit version 4 (GATK) [8] allows for detecting copy-number variants from next-geneneration sequencing data. Besides providing the results as tabular data, GATK also includes a rudimentary visualization tool that produces static whole-genome plots. The aim is to improve upon the tool by exploiting GenomeSpy.

4.2.1 Copy-Number Segmentation

Investigators usually want to examine copy-number alterations (subsection 2.1.2) as segments, i.e., stretches of DNA having the same copy number. Boundaries of the segments imply breaks that have occurred in cells' ancestors, and may thus reveal broken genes and give hints about their phylogeny in the cancer. Moreover, segmented data are easier to manage, analyze, and visualize.

Analyses performed with GATK consist of multiple, consecutive pipelines. In the first stage, the raw DNA sequence data is aligned (Figure 4.1). In later stages, two essential pieces of information can be extracted from the aligned data: *read counts* and *allele frequencies*. The read count indicates how much DNA was found for any specific locus, and thus, allowing for inferring the copy number. However, the data is noisy and unusable without further processing. Allele frequencies allow for inferring whether a locus is heterozygous or homozygous, i.e., are the copies different or identical.

To find the breakpoints and determine the segments, GATK inputs the noisy read-count and allele-frequency data, performs denoising and normalization, and employs a multi-phase segmentation algorithm. The toolkit has numerous tuning parameters that may require adjustment. The final results can be visualized with the `PlotModeledSegments` tool, which produces a plot similar to the one shown in Figure 4.2.

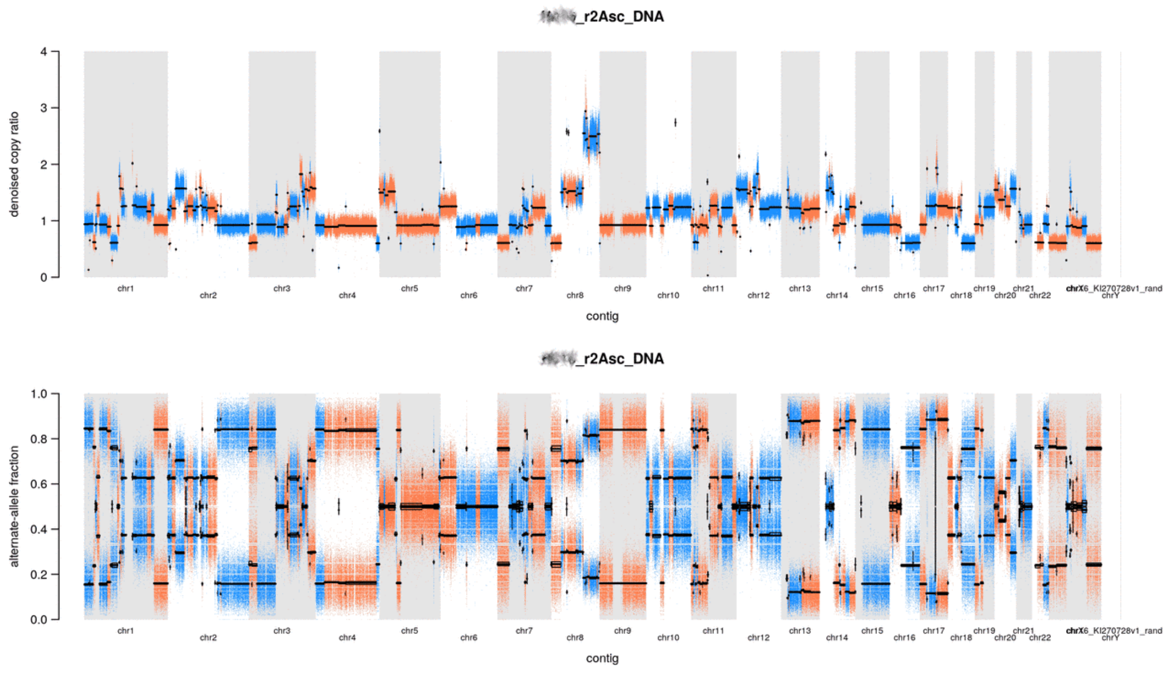


Figure 4.2: Output of the GATK `PlotModeledSegments` tool. The visualization consists of two subplots, both of which display the same segments on the concatenated genomic axis. The plots have three layers: a scatter plot displaying the raw data, rectangles encoding the 95% credible intervals of the segments, and horizontal rules (line segments) encoding the posterior means of the segments. The alternating colors in the scatter plots emphasize the segmentation – a continuous color means continuous segment. The **upper plot** displays normalized read counts (copy ratios). The genome has been divided into bins of one kilobase and the points represent mean copy ratios of the bins. The **lower plot** displays variant allele frequencies of select heterozygous loci. Values near 0 and 1 mean homozygous, whereas 0.5 means heterozygous. The visualization gives a rough overview of both the state of the genome and the segmentation quality. However, being non-zoomable, it does not allow for scrutinizing important details such as the exact breakpoint locations.

4.2.2 Zoomable Visualization

Adjusting the segmentation parameters requires close inspection of the segmentation results. However, the `PlotModeledSegments` tool cannot reveal the critical details at kilobase-scale resolution. Moreover, the read counts may correlate with the GC-content, expressing *wave artifacts* that may disturb the segmentation process [9]. GATK can perform GC-correction that reduces the correlation to some extent, but its performance is difficult to assess without a juxtaposed GC-content plot. The `PlotModeledSegments` tool does not provide such.

The GATK Best Practices documentation recommends the IGV and R for more demanding visualization tasks. However, the IGV poorly supports scatter plots, does not allow for rendering horizontal line segments, and has no means for displaying the credible intervals. Many genome-aware plotting packages exist for R, but scrutinizing numerous samples without interactivity is laborious. Because these shortcomings are trivial to address with GenomeSpy, I used it to create an easy-to-use visualization application that allows for examining the segmentation results in more detail. The design principles and architecture of *SegmentModel Spy* are described below.

4.2.3 Task Abstration

Although I had a pre-existing visualization design as a template, I identified the following tasks to better understand its limitations and to ensure that the new, improved design and implementation are adequate.

T1 Explore to get an overview of the copy-number status of the genome.

T2 Compare the raw and segmented data to assess the segmentation quality.

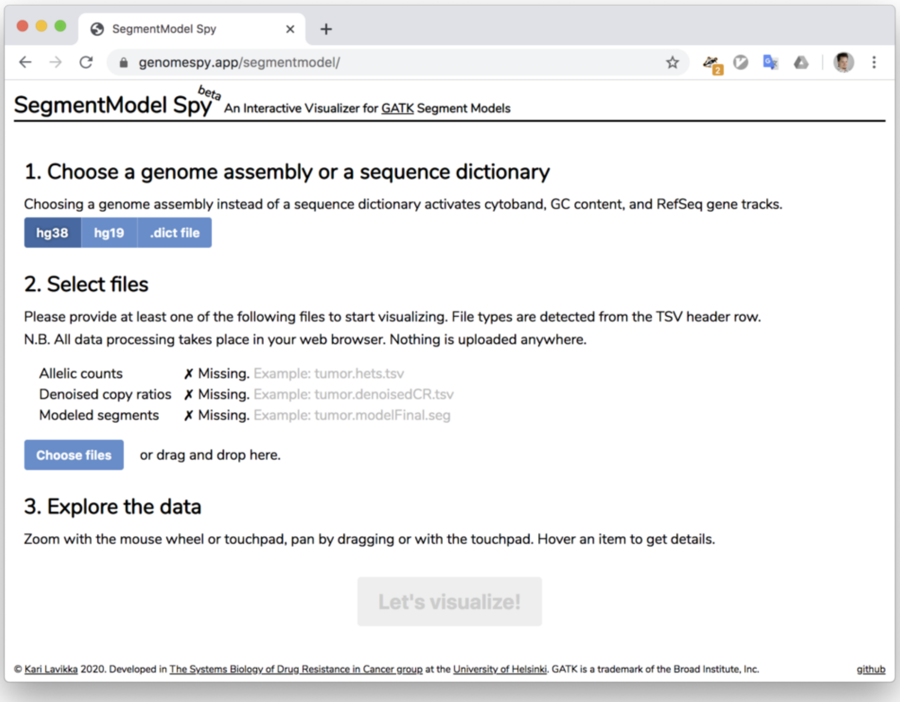
T3 Locate suspicious breakpoints to reveal problems in the segmentation.

T4 Compare the raw copy ratios to GC-content to ensure that no prominent correlation exists.

`PlotSegmentModels` supports the tasks 1-3 to some degree, but a static plot has insufficient resolution to reveal the critical details. However, a zoomable visualization would fully support these tasks.

4.2.4 Implementation

SegmentModel Spy is an easy-to-use frontend that uses GenomeSpy as a visualization library. The application presents the user with a welcome screen (Figure 4.3) that allows for loading the GATK result files for visualization. When the user loads the


SegmentModel Spy

genomespy.app/segmentmodel/

SegmentModel Spy beta An Interactive Visualizer for GATK Segment Models

1. Choose a genome assembly or a sequence dictionary

Choosing a genome assembly instead of a sequence dictionary activates cytoband, GC content, and RefSeq gene tracks.

2. Select files

Please provide at least one of the following files to start visualizing. File types are detected from the TSV header row.
N.B. All data processing takes place in your web browser. Nothing is uploaded anywhere.

Allelic counts	✗ Missing. Example: tumor.hets.tsv
Denoised copy ratios	✗ Missing. Example: tumor.denoisedCR.tsv
Modeled segments	✗ Missing. Example: tumor.modelFinal.seg

or drag and drop here.

3. Explore the data

Zoom with the mouse wheel or touchpad, pan by dragging or with the touchpad. Hover an item to get details.

© Kari Lavikka 2020. Developed in The Systems Biology of Drug Resistance in Cancer group at the University of Helsinki. GATK is a trademark of the Broad Institute, Inc. github

Figure 4.3: The welcome screen of SegmentModel Spy. The application inputs the same data files as `PlotModeledSegments`. If the user selects a genome assembly, the visualization will be augmented with annotation tracks.

data files by dragging them to the application window, the files are parsed and their types are detected automatically. Once the user has provided all the required files, the application generates a visualization specification and displays it with GenomeSpy in the same window (Figure 4.4). Although the application is web-based, all data processing takes place securely in the user's web browser. No external servers are involved. Figures 4.4, 4.5, and 4.6 demonstrate a use case where the user explores the genome by navigation (T1), assesses the segmentation quality (T2, T3), and observes GC-correlation (T4).

The full dynamically generated JSON view specification is too large to be included in this thesis. However, its structure is summarized in Figure 4.7. With four million data items, on a MacBook Pro (13-inch, 2018) and a Radeon Pro 580 eGPU, the visualization is able to maintain constant 60 FPS performance at all zoom levels, ensuring fludity with smooth and continuous zooming.

The application is accessible at `https://genomespy.app/segmentmodel/` and its source code is freely available at `https://github.com/tuner/segment-model-spy` with an open-source license.

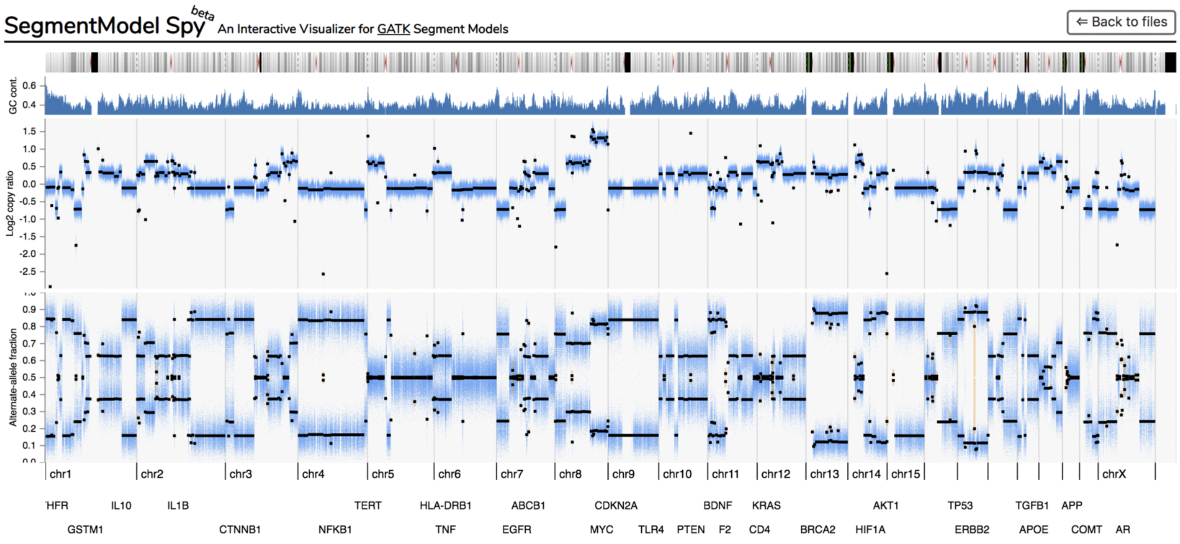


Figure 4.4: SegmentModel Spy after the user has chosen a reference genome and loaded the data files, which contain approximately four million data items. The visualization consists of six tracks: *Cytobands* aid in navigation. *GC-content* in 250 kilobase windows. *Log2 copy ratio* and *Alternate-allele fraction* are familiar from `PlotModeledSegments`. However, the copy ratio is log2-transformed. Some of the analyzed samples have segments with remarkably high copy ratios, which are difficult to display using a linear scale. *Genome axis* displays the chromosomal coordinates. *Gene annotation* displays RefSeq genes, allowing the user to see whether genes are affected by a breakpoint or segment. The user can navigate around the genome by using the mouse wheel or trackpad to zoom. Dragging by mouse or mouse gestures pan the viewport. The user can also double click chromosomes, cytobands, genes, or marks in the visualization to zoom in quickly.

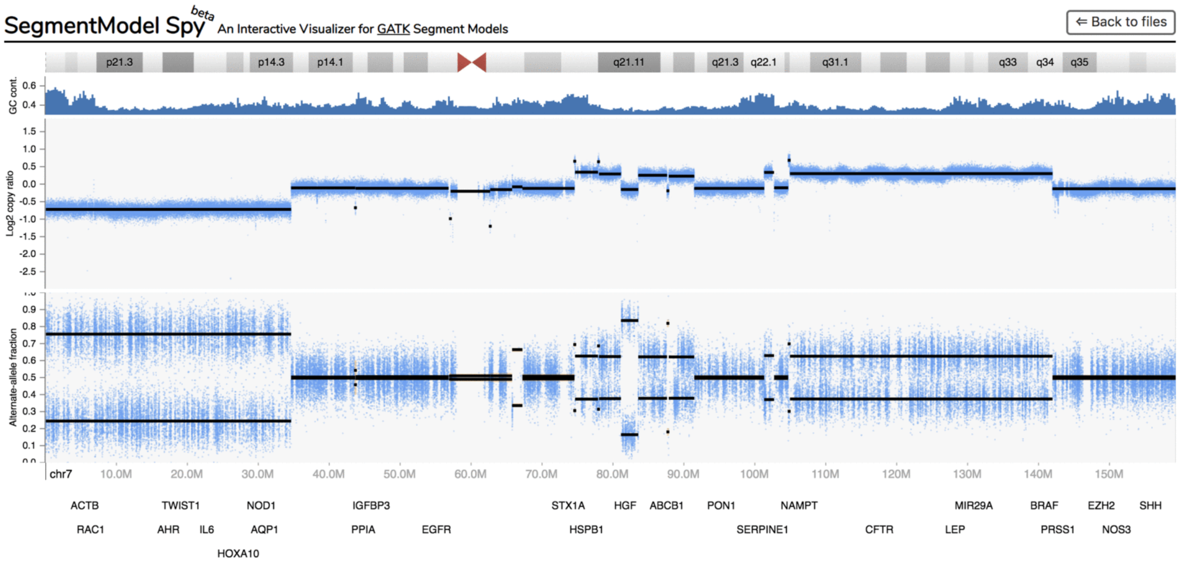
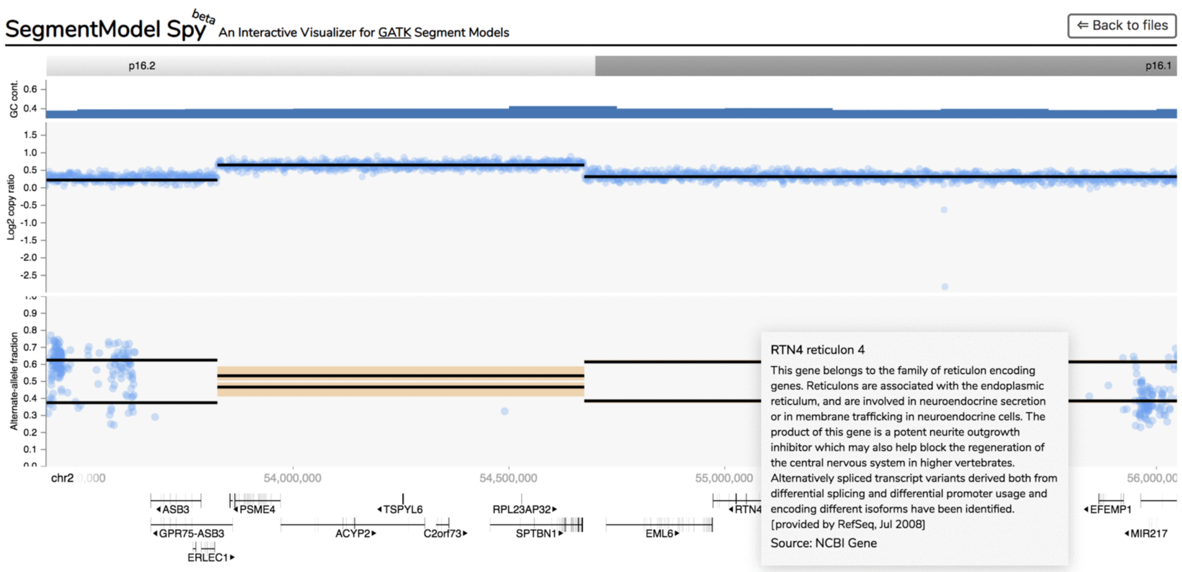


Figure 4.5: The user has zoomed into the chromosome seven to see the details more clearly. The GC-wave artifacts in the copy ratio are now more clearly visible. The user can compare the raw data to the GC-content and judge whether the waves are artifacts or represent a true copy-number alteration in the genome. Too sensitive segmentation parameters may result in breaks to the steepest parts of the waves. The user can easily discover these cases with the interactive visualization. New, less cited genes have appeared onto the gene annotation track.



RTN4 reticulon 4

This gene belongs to the family of reticulon encoding genes. Reticulons are associated with the endoplasmic reticulum, and are involved in neuroendocrine secretion or in membrane trafficking in neuroendocrine cells. The product of this gene is a potent neurite outgrowth inhibitor which may also help block the regeneration of the central nervous system in higher vertebrates. Alternatively spliced transcript variants derived both from differential splicing and differential promoter usage and encoding different isoforms have been identified. [provided by RefSeq, Jul 2008]

Source: NCBI Gene

Figure 4.6: The user has noticed a segment with a relatively large credible interval in the alternate-allele fraction and zoomed closer. The breakpoints look reasonable, but the status of heterozygosity is uncertain. The visualization uses semi-geometric zooming, and the points in the scatter plot are now clearly visible. The user could hover the points with the mouse cursor for exact attribute values. At this zoom level the gene annotation track displays gene bodies, exons, and direction in addition to the gene symbols. The user has been interested in the nearby *RTN4* gene and hovered it with the mouse cursor. A tooltip reveals details such as the full name of the gene and a concise summary.

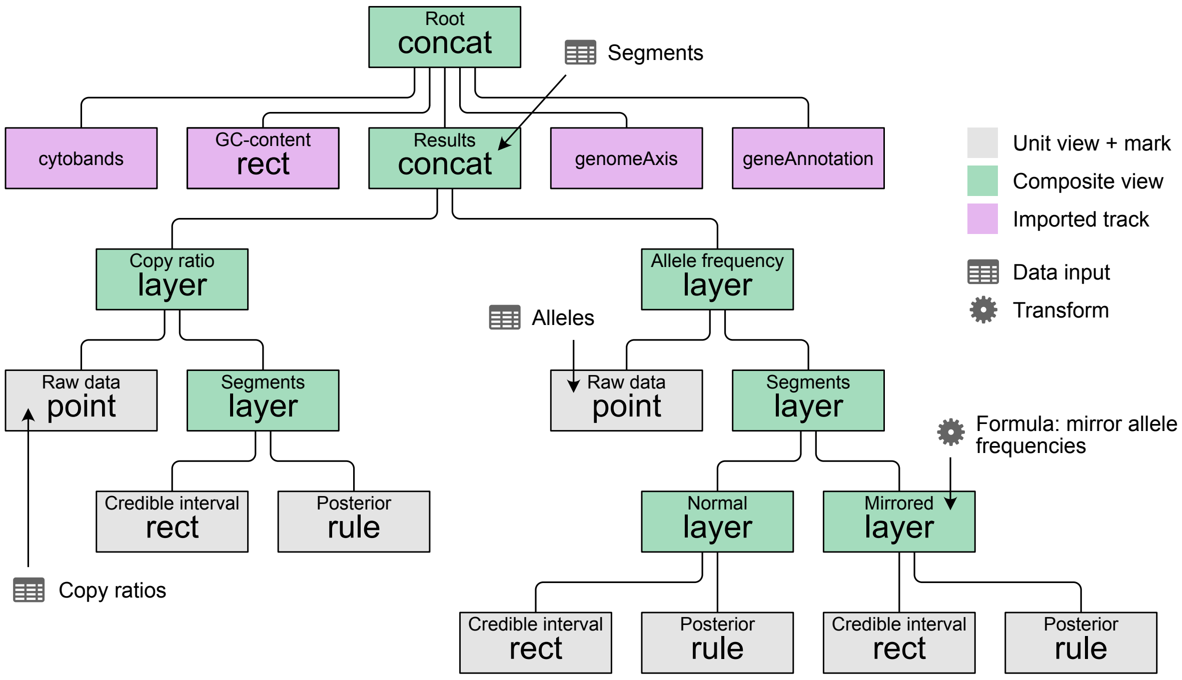


Figure 4.7: The view hierarchy of the SegmentModel Spy visualization. The segmented data and the x and $x2$ channels are defined in the *Results* view, from where they propagate down the tree. Scales under the layer views are shared by default. Thus, the raw data and segments have common y axes.

4.3 Case Study II: Multidimensional Sample Comparison

This case study validates the expressivity of the grammar with a novel visualization design that employs faceting to display multiple samples. Moreover, the implementation and interaction design of GenomeSpy are validated with a user study.

Many genomic aberrations have a joint effect on the phenotype. The most striking example in HGSOC is the co-occurrence of *TP53* mutation and loss of heterozygosity in chromosome 17 [20]. This case study presents a GenomeSpy-based visualization, which displays copy-number variation (CNV), loss of heterozygosity (LOH), and point mutations for multiple samples at the same time. The CNV and LOH dimension were analyzed with the pipeline explained in Section 4.2. Mutations were analyzed using another GATK pipeline and various annotation tools, such as CADD [19], which computes scores for the mutations. The score is based on allelic diversity, functional annotations, and various other sources, reflecting mutations' functional significance.

4.3.1 Task Abstraction

Based on discussions with domain experts, I identified the following tasks that guided the visualization design, and more broadly, the design of the interactions in GenomeSpy.

T1 **Explore the genome** to get an overview of the dataset.

T2 **Compare multiple data dimensions** to better understand their joint effect.

T3 **Compare multiple samples** to reveal outliers or patterns spanning the patient cohort or different subsets of the samples.

T4 **Compare genomic data to metadata**. The user might, for example, want to investigate whether a clinical attribute correlates with a genomic feature.

T5 **Manipulate the sample set** by filtering and sorting by metadata and the actual genomic data. Users may want to, for example, focus on individual patients and choose the samples that have the highest purity (a sort of quality metric).

T6 **Lookup features and coordinates** to quickly navigate to the focus of interest. The user may be interested in a specific gene or region. For example, the user wants to quickly check whether a reported finding is also visible in his/her data.

T7 **Compare genomic data to annotations** such as RefSeq genes and COSMIC Census. Genomic aberrations impact the disease through genes. Thus, users are interested to see how aberrations located with respect to them.

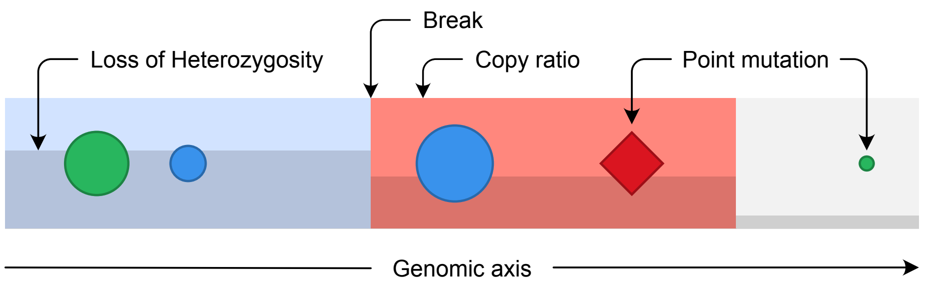


Figure 4.8: Layered visualization design for multiple data dimensions (T2). The design consists of three layers, which represent the different data dimensions: the two bottom layers consists of rectangles with left and right edges marking the start and end coordinates of the copy-number segments. The colors of the full-height bottom rectangles encode the *copy-ratios* using a conventional blue-white-red (deletion-neutral-amplification) color scheme. The *heights* of the dark, translucent rectangles in the middle layer encode the *loss of heterozygosity*. The point symbols represent *point mutations* and indels. Three visual channels have been utilized: *Color* encodes *functional category*, a nominal variable that indicates mutation's effect on transcription and translation. *Size (area)* encodes allele-frequency, a quantitative variable with the domain of $[0, 1]$. Reading an exact value from an area is difficult, but comparing the value to nearby data items is sufficient for the users. The exact value is available in a tooltip. *Shape* encodes whether the mutation is somatic (circle) or germ-line (diamond).

4.3.2 Visualization Design

This subsection describes the visualization designs I developed for faceted multidimensional samples (T2) and *COSMIC Cancer Gene Census* [40] annotations (T7).

Multidimensional Samples

Case Study I presented a visualization design that displays copy-number variation and B-allele frequency (BAF) using the positional y channel. However, such a design is inadequate when hundreds of samples are shown at the same time: heights of the facets become too small. Figure 4.8 displays the visualization design for T2. In addition to CNVs and BAF, the design incorporates third data dimension: point mutations and indels.

The bottom layer displays the CNVs using a convention that is familiar from the IGV and other existing tools. Color is perceptually less accurate than positional channels, but it allows for pre-attentive search of outliers and easily reveals patterns in the data [26].

The middle layer displays the BAF. However, instead of directly encoding the BAF, it is first transformed to loss of heterozygosity (LOH) using the formula:

$$LOH = 2\,|BAF - 0.5| \tag{4.1}$$

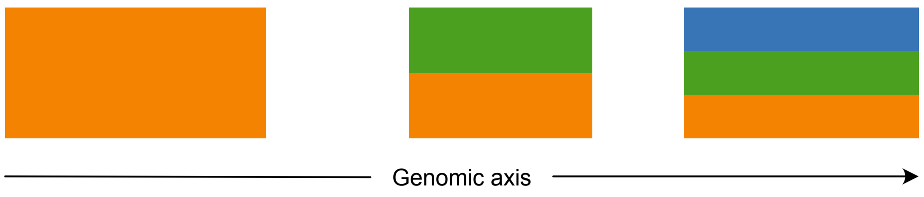


Figure 4.9: Visualization design for COSMIC Cancer Gene Census track (T7). The design consists of rectangles with left and right edges marking the start and end coordinates of the genes. The *color* encodes the *role in cancer*: orange for tumor suppressor genes, green for fusion targets, blue for oncogenes. If a gene has multiple roles, the colors are *stacked*.

LOH better highlights the deviation from the normal status. When a segment is fully heterogenous (BAF is 0.5), LOH is zero. Conversely, when it is fully homogenous (BAF is 0 or 1), LOH is one. In this design, LOH is displayed as a translucent rectangle with its height encoding the LOH. Height is a positional channel allowing for perception of the value with precision. When hundreds of samples are visualized, height becomes inaccurate. However, the translucent black rectangle influences the luminance of the underlying CNV rectangle, allowing for perception of LOH as a component of the color channel.

The top layer displays point mutations and indels. Because they only affect a single or a few nucleotides, they are displayed as points. As color allows for pre-attentive search, it is reserved for *functional category*, the most important attribute, which describes mutation's effect on the transcription and translation process. Size (area) is relatively inaccurate channel, but allows for approximate but sufficient perception of the allele frequency. When the user has found a mutation, they are interested to know whether it is somatic or germ-line. The shape channel does not support pre-attentive search, but is suitable for this purpose. Thus, the visual encodings have been chosen so that the most effective channels are used for the most important attributes.

COSMIC Cancer Gene Census Annotation

The Cancer Gene Census (CGC) is a catalogue of genes that have been associated with cancer. By comparing genomic features to the census genes, investigators may get evidence for feature's association with cancer. An evidence is not a proof, but it can steer the exploration process.

Figure 4.9 displays the visualization design for the census track. Color is used for encoding the role in cancer. Other attributes are accessible through tooltips.

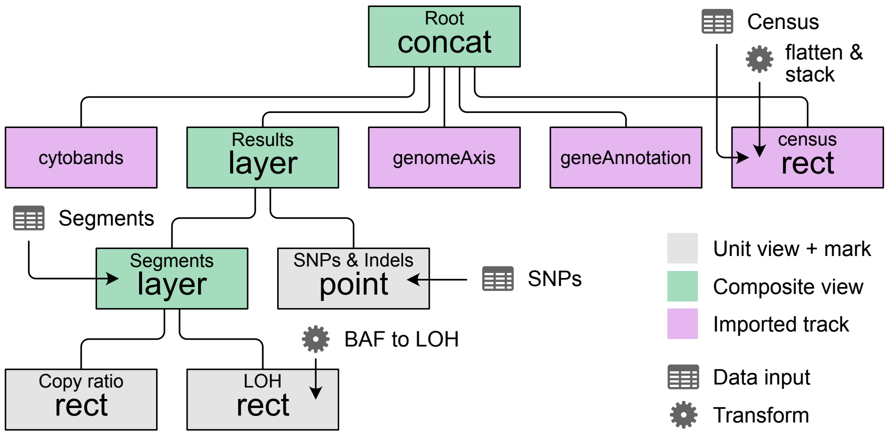


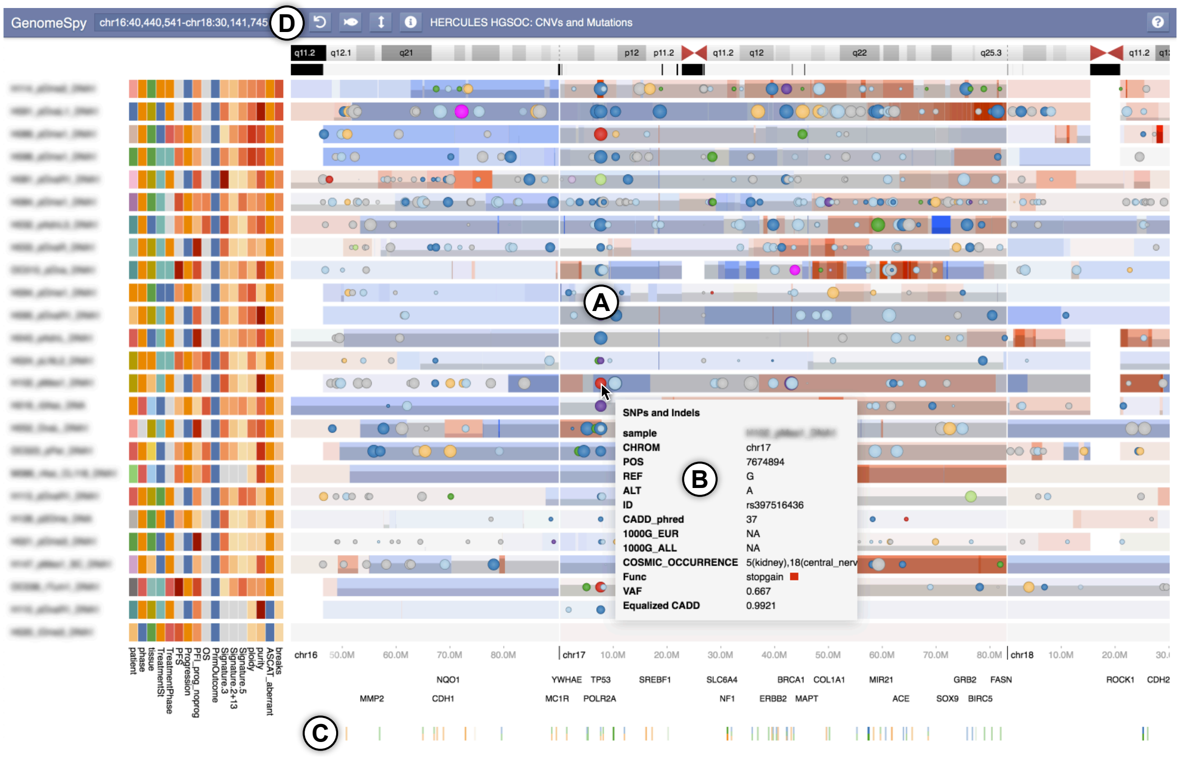
Figure 4.10: The view hierarchy of the multidimensional sample comparison. Some transforms have been omitted for clarity.

4.3.3 Implementation

Figure 4.10 displays the view hierarchy of the visualization specification. The *Results* view implements the visualization design for multidimensional samples. The nested *SNPs and Indels* view uses the CADD score for score-based semantic zooming: a dynamic filter adjusts the score threshold as the users zooms in or out, preventing excess overplotting. An approximately constant number of mutations are always visible even though the CADD score is exponentially distributed.

The *census* view implements the design for the Cancer Gene Census track. It uses the `fattenDelimited` transform to split rows having multiple roles in cancer to multiple rows. The `stack` transform computes a normalized stacked layout for the colored rectangles. The full view specification for the census track can be found in Section A.2.

Figure 4.11 displays the the final visualization with multiple samples (T3). It is implemented as a web page that uses the GenomeSpy JavaScript library to parse the visualization specification and embed it as a full-screen application on the page. Thus, the user only needs to browse to a URL and start exploring. Subsection 3.4.3 demonstrates the metadata and filtering interactions (T4, T5) using the same visualization.



SNPs and Indels

sample	[redacted]
CHROM	chr17
POS	7674894
REF	G
ALT	A
ID	rs397516436
CADD_phred	37
1000G_EUR	NA
1000G_ALL	NA
COSMIC_OCCURRENCE	5(kidney),18(central_nerv
Func	stopgain 🟥
VAF	0.667
Equalized CADD	0.9921

Figure 4.11: The multidimensional sample comparison visualization displaying HGSOC data. The user has navigated (T1) to chromosome 17 to view the co-occurence of *TP53* mutations (A, visible as a column of points) and loss of heterozygosity (dark translucent rectangles). (B) A tooltip provides details on demand for the data items. (C) The Cancer Gene Census track displays the locations of well-known cancer-related genes. (D) The toolbar displays the genomic coordinates of the current view and allows the user to lookup loci and genes quickly (T6). The toolbar contains a button for reverting the sorting and filtering actions. Other buttons allow for activating the fisheye and peek. This screenshot displays only a small subset of the samples to better illustrate the visualization designs. When hundreds of samples are visible, the user can use the fisheye and peek tools to quickly focus on specific samples (subsection 3.4.4). The user can also filter and sort the samples to better investigate more focused subsets (subsection 3.4.3).

4.3.4 User study

To validate the visualization and interaction designs I had created, I conducted a small user study. In practice, we organized a genetics workshop that consisted of two parts. We began by teaching the lab members some cancer genetics. Next, we instructed the participants to complete a number of exercises using the GenomeSpy visualization described in this case study. The participants were a heterogeneous group of post-docs and students from fields such as biology and bioinformatics. After the workshop, we collected feedback using Google Forms. At the time of the workshop, some features, such as semantic zoom and peek, were not yet implemented.

Workshop Exercises

First half of the exercises introduced the participants to the visualization. An example problem: “ Find the basic elements of the user interface / visualization: 1) Location / search bar, 2) Cytobands, 3) . . . ” The second half comprised of real biological questions that the participant had to work out by using the GenomeSpy visualization. Example problems: “ Look at *MYC* gene in patient A123, compare interval and primary samples. Do you think the difference is reliable? ” and “ Can you detect some patients, who seem to have or have had higher genomic instability than others? Is there any clues whether it happened earlier or is the process still ongoing? ” I formulated the user-interface related exercise. The domain experts (a senior scientist and a geneticist post-doc) created the biology exercise.

Feedback Results

Table 4.1 displays the feedback results. The answers reflect participants views on the user-experience of GenomeSpy in general and the effectiveness of the specific visualization described in this case study. Although not all participants felt that GenomeSpy has utility in their research, it was considered easy to use. The visualization design presented the underlying data effectively. However, several participants discovered that although the filtering functionality is practical, it has a limitation:

“ I would like to be able to filter samples on a range of values instead of only larger than or smaller than a threshold. In other words an easier way to keep the extremes for some attribute. ”

Currently, filtering by a quantitative attribute is only possible by choosing a single threshold value. Thus, removing values between two thresholds is impossible.

All the verbal responses can be found in Appendix B.

Table 4.1: Feedback questions and answers. n = 9. The scale is (1) disagree - (5) agree.

		Answers					
	Question	1	2	3	4	5	Mean
1	Navigation around the genome is easy	0	0	0	2	7	4.78
2	Sample sorting and filtering is easy	0	1	0	6	2	4.00
3	Overall, the tool feels intuitive to use	0	1	0	4	4	4.22
4	The visualization presents the underlying data effectively	0	0	0	6	3	4.33
5	The visualization is aesthetic	0	0	1	1	7	4.67
6	Observing multiple dimensions the same time, e.g. CNV, LOH, SNPs, is practical	0	0	0	6	3	4.33
7	The Fisheye feels more practical than a vertically scrollable viewport (which GenomeSpy does not have)	0	0	2	1	6	4.44
8	I plan to use GenomeSpy in my research	1	0	4	1	2	3.38
9	I plan to use GenomeSpy in my research if certain missing features are added. (Please specify below)	2	0	1	1	4	3.62

5. Discussion and Conclusions

GenomeSpy is a visualization software library that can be used as a building block in other applications. In the current form, it may be inaccessible to users of traditional genome browser applications that allow for opening a data file from the local disk and viewing it in a familiar, specific way. However, as the SegmentModel Spy (section 4.2) demonstrated, GenomeSpy could be used as a foundation for a higher-level application that provides a simple point-and-click interface and familiar visual encodings for common file formats.

When it comes to users wishing to explore the design space and author novel visualizations, they have to provide the specification as a JSON document. However, a designer application with a graphical user interface and drag-and-drop interactions could be built on top of the grammar [32]. Such an approach would make visualization authoring accessible to a broader audience.

5.1 Increasing Expressivity

Although the case studies demonstrated the grammar's expressivity, the current set of building blocks is rather limited: only three graphical marks and a few data transforms have been implemented. However, new building blocks can be added without altering the general structure of the grammar. For instance, introducing an `arc` mark generalized as a Bézier curve would enable visualization designs for structural variation. A `pileup` transform would allow designs for sequence alignment data (Figure 4.1), and a `coverage` transform would enable coverage computation directly in the visualization [47]. Although the building blocks could be built primarily with just one use case in mind, an adequate level of abstraction ensures that they can be adapted and repurposed for new visualization designs creatively.

The intended application domain of GenomeSpy is coordinate-based genomic data. However, to some extent, it can also be used as a general-purpose visualization tool. By implementing a zoomable vertical axis, the GPU-accelerated architecture of GenomeSpy could be used to visualize, for example, large scatter plots displaying dimensionally reduced data.

5.2 Shortcomings of the Architecture

Currently, GenomeSpy loads the whole dataset to memory when the visualization is launched. Later discussions with domain experts have clarified the requirements and indicated that dense, gigabyte-scale data needs to be supported. For instance, some usage scenarios require coverage plots with the resolution as high as a single base. However, the current design is not flexible enough. To support such use cases, zoom-level dependent lazy-loading of data needs to be implemented. Data processing must be asynchronous and multi-threaded to ensure the responsiveness of the user interface [36]. The implementation would necessitate redesigning of certain parts of the architecture and requires further research.

The architecture has one particular design flaw, which impedes lazy loading. Except for the horizontal genomic axis, all scale transformations are computed by the CPU. The approach has several consequences. Firstly, the data cannot be uploaded to the GPU before it is transformed. However, data cannot be transformed before the scale function's domain is known. If the domain is extracted from the data, all involved data must be loaded first. Thus, dynamic lazy-loading of data is incompatible with the current architecture. Secondly, changing the data domain dynamically – by the user, for instance – requires that all scale transformations are re-computed, and the results are re-uploaded to the GPU memory. A better approach would be to move all scale transformations to the GPU by using shader programs [30, 22, 31]. In other words, instead of uploading scale-transformed data to the GPU, the raw data should be uploaded. An added benefit is that the raw data could be used for efficient data filtering. The score-based semantic zoom uses such design already.

5.3 Insights from the User Studies

Although the user study (subsection 4.3.4) indicated that the implemented interactions support the identified tasks, it also revealed some severe shortcomings in the sample filtering interactions. It is currently virtually impossible to remove samples having a quantitative metadata attribute within a specific range, e.g., retain only cases in the top and bottom 10%. Current filtering interaction only supports subsetting the current set of samples by using a single threshold. A solution that embraces the direct manipulation principle needs further research. Moreover, later discussions have also revealed a need for freely sortable samples. The users would like to drag and drop the samples to arrange them in a specific order.

A geneticist in the research group considers the GenomeSpy-based multidimensional visualization (section 4.3) invaluable in her research and uses it daily. She appreciates the interaction design, which allows for quickly seeking answers to specific research questions. Although this work aimed to improve exploration, GenomeSpy is also applicable to specific questions.

The usability of the visualization grammar has not been evaluated rigorously. I arranged a mini user study by recruiting two members of the research group to visualize their own data with GenomeSpy. With some help, they succeeded. They also found some bugs and were able to push the tool to its limits. A real user study would likely provide valuable feedback to guide further development.

5.4 Towards Fluid Analysis

In the future, one possible research direction is the integration of elementary analysis support into GenomeSpy. Currently, when the user finds an interesting observation using the visualization, they have to load the data into external software for further analyses. By providing functionality for grouping and aggregating the samples and performing statistical tests between the groups, the user could conduct the initial analyses directly in GenomeSpy. Moreover, the analyses could be exploited in a recommendation engine, which finds regions with significant differences between groups in the genome. The system could guide the user in the exploration process.

The "fluid analyses" would benefit from recording provenance data, as it facilitates replicability, allows users to explore multiple paths in the analyses, and allows for more accurate communication of the results [13]. GenomeSpy already records a linear history of sample filtering and sorting actions, but it could be extended further.

5.5 Concluding Remarks

This thesis presented GenomeSpy, a genome-browser-like visualization tool that improves upon the existing state of the art by supporting more extensive and efficient exploration. Its visualization grammar allows for exploring the design space, and the interaction design supports more efficient exploration of the genomic space of multiple samples. Two case studies validated the expressivity of the grammar and the benefit of the GPU-accelerated interactions.

GenomeSpy is freely available at `https://genomespy.app/`.

Bibliography

[1] kepler.gl - Geospatial analysis tool for large-scale data sets. https://kepler.gl/.

[2] B. Alberts, D. Bray, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Essential Cell Biology.* Garland Science, oct 2013.

[3] E. Angel and D. Shreiner. *Interactive Computer Graphics with WebGL.* Addison-Wesley Professional, 7th edition, 2014.

[4] J. T. Behrens. Principles and procedures of exploratory data analysis. *Psychological Methods*, 2(2):131–160, 1997.

[5] M. Bostock, V. Ogievetsky, and J. Heer. D^3 Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, dec 2011.

[6] A. Cockburn, A. Karlson, and B. B. Bederson. A review of overview+detail, zooming, and focus+context interfaces. *ACM Computing Surveys*, 41(1):1–31, jan 2009.

[7] A. Cockburn and J. Savage. Comparing Speed-dependent Automatic Zooming with Traditional Scroll, Pan and Zoom Methods. In *People and Computers XVII — Designing for Society*, pages 87–102. Springer London, London, 2004.

[8] M. A. DePristo, E. Banks, R. Poplin, K. V. Garimella, J. R. Maguire, C. Hartl, A. A. Philippakis, G. del Angel, M. A. Rivas, M. Hanna, A. McKenna, T. J. Fennell, A. M. Kernytsky, A. Y. Sivachenko, K. Cibulskis, S. B. Gabriel, D. Altshuler, and M. J. Daly. A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nature Genetics*, 43(5):491–498, may 2011.

[9] S. J. Diskin, M. Li, C. Hou, S. Yang, J. Glessner, H. Hakonarson, M. Bucan, J. M. Maris, and K. Wang. Adjustment of genomic waves in signal intensities from whole-genome SNP genotyping platforms. *Nucleic Acids Research*, 36(19):e126–e126, nov 2008.

[10] N. Elmqvist, A. V. Moere, H.-C. Jetter, D. Cernea, H. Reiterer, and T. Jankun-Kelly. Fluid interaction for information visualization. *Information Visualization*, 10(4):327–340, 2011.

[11] J. Ferlay, I. Soerjomataram, R. Dikshit, S. Eser, C. Mathers, M. Rebelo, D. M. Parkin, D. Forman, and F. Bray. Cancer incidence and mortality worldwide: Sources, methods and major patterns in GLOBOCAN 2012. *International Journal of Cancer*, 136(5):E359–E386, 2015.

[12] M. J. Goldman, B. Craft, M. Hastie, K. Repečka, F. McDade, A. Kamath, A. Banerjee, Y. Luo, D. Rogers, A. N. Brooks, J. Zhu, and D. Haussler. Visualizing and interpreting cancer genomics data via the Xena platform. *Nature Biotechnology*, may 2020.

[13] S. Gratzl, A. Lex, N. Gehlenborg, N. Cosgrove, and M. Streit. From Visual Exploration to Storytelling and Back Again. *Computer Graphics Forum*, 35(3):491–500, jun 2016.

[14] C. Green. Improved alpha-tested magnification for vector textures and special effects. In *ACM SIGGRAPH 2007 courses on - SIGGRAPH '07*, page 9, New York, New York, USA, 2007. ACM Press.

[15] D. Hanahan and R. A. Weinberg. Hallmarks of cancer: The next generation. *Cell*, 144(5):646–674, 2011.

[16] J. Heer and G. G. Robertson. Animated transitions in statistical data graphics. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1240–1247, 2007.

[17] W. J. Kent, C. W. Sugnet, T. S. Furey, and K. M. Roskin. The Human Genome Browser at UCSC W. *Genome Research*, 12:996–1006, 2002.

[18] P. Kerpedjiev, N. Abdennur, F. Lekschas, C. McCallum, K. Dinkla, H. Strobelt, J. M. Luber, S. B. Ouellette, A. Azhir, N. Kumar, J. Hwang, S. Lee, B. H. Alver, H. Pfister, L. A. Mirny, P. J. Park, and N. Gehlenborg. HiGlass: web-based visual exploration and analysis of genome interaction maps. *Genome Biology*, 19(1):125, dec 2018.

[19] M. Kircher, D. M. Witten, P. Jain, B. J. O'Roak, G. M. Cooper, and J. Shendure. A general framework for estimating the relative pathogenicity of human genetic variants. *Nature genetics*, 46(3):310–315, 2014.

[20] S. I. Labidi-Galy, E. Papp, D. Hallberg, N. Niknafs, V. Adleff, M. Noe, R. Bhattacharya, M. Novak, S. Jones, J. Phallen, C. A. Hruban, M. S. Hirsch, D. I. Lin, L. Schwartz, C. L. Maire, J.-C. Tille, M. Bowden, A. Ayhan, L. D. Wood, R. B. Scharpf, R. Kurman, T.-L. Wang, I.-M. Shih, R. Karchin, R. Drapkin, and V. E. Velculescu. High grade serous ovarian carcinomas originate in the fallopian tube. *Nature Communications*, 8(1):1093, dec 2017.

[21] C. M. Lee, G. P. Barber, J. Casper, H. Clawson, M. Diekhans, J. N. Gonzalez, A. S. Hinrichs, B. T. Lee, L. R. Nassar, C. C. Powell, B. J. Raney, K. R. Rosenbloom, D. Schmelter, M. L. Speir, A. S. Zweig, D. Haussler, M. Haeussler, R. M. Kuhn, and W. J. Kent. UCSC Genome Browser enters 20th year. *Nucleic Acids Research*, pages 1–6, nov 2019.

[22] J. K. Li and K. L. Ma. P4: Portable Parallel Processing Pipelines for Interactive Information Visualization. *IEEE Transactions on Visualization and Computer Graphics*, PP(c):1, 2018.

[23] M.-A. Lisio, L. Fu, A. Goyeneche, Z.-h. Gao, and C. Telleria. High-Grade Serous Ovarian Cancer: Basic Sciences, Clinical and Therapeutic Standpoints. *International Journal of Molecular Sciences*, 20(4):952, feb 2019.

[24] A. Mayakonda, D.-C. Lin, Y. Assenov, C. Plass, and H. P. Koeffler. Maftools: efficient and comprehensive analysis of somatic variants in cancer. *Genome Research*, 28(11):1747–1756, nov 2018.

[25] L. Micallef, G. Palmas, A. Oulasvirta, and T. Weinkauf. Towards Perceptual Optimization of the Visual Design of Scatterplots. *IEEE Transactions on Visualization and Computer Graphics*, 23(6):1588–1599, jun 2017.

[26] T. Munzner. *Visualization Analysis and Design.* A K Peters/CRC Press, dec 2014.

[27] S. Nusrat, T. Harbig, and N. Gehlenborg. Tasks, Techniques, and Tools for Genomic Data Visualization. *Computer Graphics Forum*, 38(3):781–805, may 2019.

[28] N. A. O'Leary, M. W. Wright, J. R. Brister, S. Ciufo, D. Haddad, R. McVeigh, B. Rajput, B. Robbertse, B. Smith-White, D. Ako-Adjei, A. Astashyn, A. Badretdin, Y. Bao, O. Blinkova, V. Brover, V. Chetvernin, J. Choi, E. Cox, O. Ermolaeva, C. M. Farrell, T. Goldfarb, T. Gupta, D. Haft, E. Hatcher, W. Hlavina, V. S. Joardar, V. K. Kodali, W. Li, D. Maglott, P. Masterson, K. M. McGarvey, M. R. Murphy, K. O'Neill, S. Pujar, S. H. Rangwala, D. Rausch, L. D. Riddick, C. Schoch, A. Shkeda, S. S. Storz, H. Sun, F. Thibaud-Nissen, I. Tolstoy, R. E. Tully, A. R. Vatsan, C. Wallin, D. Webb, W. Wu, M. J. Landrum, A. Kimchi,

T. Tatusova, M. DiCuccio, P. Kitts, T. D. Murphy, and K. D. Pruitt. Reference sequence (RefSeq) database at NCBI: Current status, taxonomic expansion, and functional annotation. *Nucleic Acids Research*, 44(D1):D733–D745, 2016.

[29] R Core Team. R: A Language and Environment for Statistical Computing, 2016.

[30] D. Ren, B. Lee, and T. Höllerer. Stardust: Accessible and Transparent GPU Support for Information Visualization Rendering. *Computer Graphics Forum*, 36(3):179–188, jun 2017.

[31] A. Sarikaya and M. Gleicher. Using WebGL as an Interactive Visualization Medium: Our Experience Developing SplatterJs. *Proceedings of the Data Systems for Interactive Analysis Workshop*, 2015.

[32] A. Satyanarayan and J. Heer. Lyra: An Interactive Visualization Design Environment. *Computer Graphics Forum*, 33(3):351–360, jun 2014.

[33] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-Lite: A Grammar of Interactive Graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):341–350, jan 2017.

[34] A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer. Reactive Vega: A Streaming Dataflow Architecture for Declarative Interactive Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):659–668, jan 2016.

[35] V. A. Schneider, T. Graves-Lindsay, K. Howe, N. Bouk, H. C. Chen, P. A. Kitts, T. D. Murphy, K. D. Pruitt, F. Thibaud-Nissen, D. Albracht, R. S. Fulton, M. Kremitzki, V. Magrini, C. Markovic, S. McGrath, K. M. Steinberg, K. Auger, W. Chow, J. Collins, G. Harden, T. Hubbard, S. Pelan, J. T. Simpson, G. Threadgold, J. Torrance, J. M. Wood, L. Clarke, S. Koren, M. Boitano, P. Peluso, H. Li, C. S. Chin, A. M. Phillippy, R. Durbin, R. K. Wilson, P. Flicek, E. E. Eichler, and D. M. Church. Evaluation of GRCh38 and de novo haploid genome assemblies demonstrates the enduring quality of the reference assembly. *Genome Research*, 27(5):849–864, 2017.

[36] M. Schuetz. *Potree: Rendering Large Point Clouds in Web Browsers.* PhD thesis, Vienna University of Technology, 2016.

[37] B. Shneiderman. Direct Manipulation: A Step Beyond Programming Languages. *Computer*, 16(8):57–69, aug 1983.

[38] R. Sicat, J. Li, J. Choi, M. Cordeil, W.-K. Jeong, B. Bach, and H. Pfister. DXR: A Toolkit for Building Immersive Data Visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):715–725, jan 2019.

[39] S. P. Strom. Current practices and guidelines for clinical next-generation sequencing oncology testing. *Cancer Biology and Medicine*, 13(1):3–11, 2016.

[40] J. G. Tate, S. Bamford, H. C. Jubb, Z. Sondka, D. M. Beare, N. Bindal, H. Boutselakis, C. G. Cole, C. Creatore, E. Dawson, P. Fish, B. Harsha, C. Hathaway, S. C. Jupe, C. Y. Kok, K. Noble, L. Ponting, C. C. Ramshaw, C. E. Rye, H. E. Speedy, R. Stefancsik, S. L. Thompson, S. Wang, S. Ward, P. J. Campbell, and S. A. Forbes. COSMIC: the Catalogue Of Somatic Mutations In Cancer. *Nucleic acids research*, 47(D1):D941–D947, 2019.

[41] A. Thall. Extended-precision floating-point numbers for GPU computation. In *ACM SIGGRAPH 2006 Research posters on - SIGGRAPH '06*, number March, page 52, New York, New York, USA, 2006. ACM Press.

[42] H. Thorvaldsdóttir, J. T. Robinson, and J. P. Mesirov. Integrative Genomics Viewer (IGV): high-performance genomics data visualization and exploration. *Briefings in bioinformatics*, 14(2):178–92, mar 2013.

[43] E. L. van Dijk, H. Auger, Y. Jaszczyszyn, and C. Thermes. Ten years of next-generation sequencing technology. *Trends in Genetics*, 30(9):418–426, sep 2014.

[44] H. Wickham. A Layered Grammar of Graphics. *Journal of Computational and Graphical Statistics*, 19(1):3–28, 2010.

[45] H. Wickham. Tidy Data. *Journal of Statistical Software*, 59(10), 2014.

[46] L. Wilkinson. *The Grammar of Graphics*. Statistics and Computing. Springer-Verlag, New York, 2 edition, 2005.

[47] T. Yin, D. Cook, and M. Lawrence. ggbio: an R package for extending the grammar of graphics for genomic data. *Genome Biology*, 13(8):R77, 2012.

Appendix A. Code examples

A.1 Layering a Lollipop Plot

```
{
  "description": "Lollipop Plot Example",
  "layer": [
    {
      "name": "Baseline",
      "data": { "values": [0] },
      "mark": "rule",
      "encoding": {
        "y": { "field": "data", "type": "quantitative", "axis": null },
        "color": { "value": "lightgray" }
      }
    },
    {
      "name": "Arrows",

      "data": {
        "sequence": {
          "start": 0,
          "stop": 6.284,
          "step": 0.39269908169,
          "as": "x"
        }
      },

      "transform": [
        { "type": "formula", "expr": "sin(datum.x)", "as": "sin(x)" }
      ],

      "encoding": {
        "x": { "field": "x", "type": "quantitative" },
        "y": {
          "field": "sin(x)",
          "type": "quantitative",
          "scale": { "padding": 0.1 }
        },
        "color": { "field": "sin(x)", "type": "quantitative" }
      },

      "layer": [
        {
          "name": "Arrow shafts",

          "mark": {
            "type": "rule",
            "size": 3.0
          }
        },
        {
          "name": "Arrowheads",

          "mark": "point",

          "encoding": {
            "shape": {
              "field": "sin(x)",
              "type": "quantitative",
              "scale": {
                "type": "threshold",
                "domain": [-0.01, 0.01],
                "range": ["triangle-down", "diamond", "triangle-up"]
              }
            },
            "size": { "value": 500 },
            "strokeWidth": { "value": 0 }
          }
        }
      ]
    }
  ]
}
```

A.2 Cancer Gene Census Track

```json
{
    "name": "cosmic",
    "title": "Cancer Gene Census",
    "description": "COSMIC Cancer Gene Census, https://cancer.sanger.ac.uk/census",
    "styles": { "height": 18 },
    "data": { "url": "Census_allWed Apr 10 16_11_43 2019.tsv" },
    "transform": [
        {
            "type": "regexExtract",
            "field": "Genome Location",
            "regex": "^(X|Y|\\d+):(\\d+)-(\\d+)$",
            "as": ["chrom", "startpos", "endpos"],
            "skipInvalidInput": true
        },
        {
            "type": "flattenDelimited",
            "field": "Role in Cancer",
            "separator": ", "
        },
        {
            "type": "stack",
            "groupby": ["chrom", "startpos"],
            "sort": { "field": "Role in Cancer" },
            "offset": "normalize"
        }
    ],
    "mark": {
        "type": "rect",
        "tooltip": {
            "skipFields": [ "chrom", "startpos", "endpos", "y0", "y1", "opacity" ]
        },
        "minWidth": 2.0,
        "minOpacity": 0.4
    },
    "encoding": {
        "x":  { "chrom": "chrom", "pos": "startpos", "type": "quantitative" },
        "x2": { "chrom": "chrom", "pos": "endpos", "offset": 1 },
        "y":  { "field": "y0", "type": "quantitative", "axis": null },
        "y2": { "field": "y1" },
        "color": {
            "field": "Role in Cancer",
            "type": "nominal",
            "scale": { "scheme": "category10" }
        },
        "opacity": {
            "expr": "datum.Tier == 2 ? 0.2 : 1",
            "type": "quantitative",
            "scale": { "type": "identity" }
        }
    }
}
```

Appendix B. Verbal Workshop Feedback

Which features did you find very useful or practical?

1. NA
2. Region/gene selection, sorting, filtering
3. NA
4. All the features related to the progression of the disease
5. The genome navigation is really intuitive. Culling by significance is very useful.
6. NA
7. fisheye, zooming, filtering etc.
8. Sorting and filtering is always very useful, given tha number of samples as well as their variable usefulness (e.g. purity, or number of samples per patient). Being able to zoom-in to and search by genes is also a welcome addition.
9. Sample sorting and filtering.

Which features were confusing or hard to use?

1. NA
2. NA
3. Lack of threshold to find for example top 5 percent of good or bad responders. I did it yesterday just by removing other samples to keep the best and worst responders for example. (I mean the columns which are not representing continuous values like sample, patient columns)
4. There are two marker called phase and this confuse me
5. Doing more elaborate filtering etc. is tricky or impossible.
6. NA
7. Hovering over small points such as the annotations or snps was hard. Some filtering options were not implemented yet, like filtering based on cnv amplification or deletion, so that was hard. Hand picking genomes one by one was hard since the annotations were so small.

8. PrimOutcome seemed to have multiple colours for what was essentially the same value (but with different capitalisation), e.g. pink "Partial Response" and purple "partial response". A togglable legend would also help with the high number of data types sharing the same colour codes, as well as a glossary in case there are any entries not immediately clear, such as PFI_prog_noprog. The top- and bottom-most patients are slightly more difficult to hover over when in fish-eye view. This is only a problem before extensive filtering. The mismatching names of same samples made a few samples to have missing data, but I suppose this will be addressed later with the sample renaming.

9. NA

What new data would you like to see integrated into GenomeSpy? Why?

1. NA

2. DNA methylation and CHIP-Seq

3. NA

4. NA

5. Expression data would be useful to see if the genomic aberrations have an effect on the phenotypes, but I'm not sure how I would want it to be visualized.

6. NA

7. methylation

8. Sample-specific attributes: Later on it might be interesting to add CNV signatures, double-base and indel as well as structural variation signatures on the left-hand side. Some of them are associated with HRD or platinum treatment. If there are too many columns of data, please consider adding an option to toggle them (would also make the view less cluttered).

   Viewport: I expect that multiple base substitutions and indels will also be visualised (provided they have a high enough CADD). You may have to truncate REF or ALT for >20bp indels as the hover box would start stretch too wide otherwise.

9. methylation data and maybe expression data.

What new features (related to user interface or analysis) would you like to see?

1. NA

2. Visualization of additional features when zooming in, possibility to select/drop ranges

3. NA

4. NA

5. Most useful for me would be showing custom data, e.g. input using a BED file with genomic coordinates + score from 0 to 100, which maps to the bar color/height. I usually want to visualize something that is not standard and experiment with it.

6. NA

7. Selecting genomes by hand one by one could be made easier with some kind of mouse select-tool. Small point mutations and annotation boxes could be highlighted when hovered over to make it easier to hit them. Filtering based on just amplified/deleted cnvs to study only samples with similar patterns etc. Auto-completed suggestions for genes in the search bar if that is possible? I think these would make the tool more fluidic, easier to use.

8. I would like to be able to filter samples on a range of values instead of only larger than or smaller than a threshold. In other words an easier way to keep the extremes for some attribute. Furthermore, filtering by segMean, BAF, ASCAT copy number (nMajor + nMinor) of a locus as well as sorting by ASCAT copy number. Box-selection for selecting genomic location or choosing samples to filter would be a very intuitive and simple method for the user to navigate or filter as opposed to using multiple clicks. A redo to go with undo or a possible drop-down list of previous history states would help (in case of undoing too much). Later it may be nice to let the user control the CADD threshold with a slider. Hotkeys for sorting attributes or filtering samples would be nice to have.

9. More flexible sample filtering, e.g. set whatever threshold on one feature or combination of features can filter out samples on the ends and retain samples in between.

General feedback

1. NA
2. NA
3. NA
4. I like GenomeSpy but there is a problem in manage a lot of data, in particular it would be useful the possibility to analyze only the selected samples and not only belonging to the one sample or the first for each patient.
5. NA
6. NA
7. Overall a nice tool with lots of potential. When perfected it will probably make a lot of things easier and more intuitive. The workshop was also good.
8. I was unsure whether it is the area or the radius of the SNV circles that is linearly associated with VAF.
9. GenomeSpy is a very flexible efficient tool and I will definitely use it in my research.