Interactive Semantic and Aesthetic Guidance for Multi-View Visualization Design

Yngve Sekse Kristiansen

Thesis for the degree of Philosophiae Doctor (PhD)
University of Bergen, Norway
2023

UNIVERSITY OF BERGEN

Interactive Semantic and Aesthetic Guidance for Multi-View Visualization Design

Yngve Sekse Kristiansen

Thesis for the degree of Philosophiae Doctor (PhD)
at the University of Bergen

Date of defense: 09.01.2023

Year: 2023

Title: Interactive Semantic and Aesthetic Guidance for Multi-View Visualization Design

Name: Yngve Sekse Kristiansen

Print: Skipnes Kommunikasjon / University of Bergen

Scientific Environment

The work presented in this thesis was conducted as a part of my PhD studies at the Department of Informatics, University of Bergen. In addition, I have been enrolled in the ICT Research School at the Department of Informatics, University of Bergen.

UNIVERSITY OF BERGEN

Research School In ICT
Information and Communication Technology

VISGROUP
UNIVERSITY OF BERGEN

Acknowledgements

I want to thank Stefan Bruckner for being a great main supervisor. He provided guidance and help in all situations when I needed it, and encouraged me to pursue ideas I found interesting. Many times during this thesis, things did not always go as expected, and Stefan helped me to reach the goal and get our works published. Without his guidance and influence, I would not have been able to finish the papers within this thesis. His love for ideas and concepts, Red Bull, and bad puns is on a level I have never seen before, ever, and I will never forget.

I also want to thank my co-supervisor Manuela Waldner for helpful advice, insights and related works into ideas pursued during this PhD.

And thanks to the others at the UiB VisGroup who made a great work environment. Thanks to Helwig Hauser and his surprisingly good table tennis skills, his stories of hikes he's done (that I have not done). Noeska Smit made the work environment at the VisGroup really great by always making some time for social arrangements such as coffee breaks, and in general being there to listen and give advice. I also wish to thank the "Island Expert" and former office mate Fabian Bolte for sometimes arriving later than myself. With him, I discussed so many ideas back and forth, which ultimately helped shape the final ideas that ended up in the papers. Furthermore, without him I would not have the title "Meat Expert" on my door sign, or started spending way too much time playing table tennis at work. I thank Laura Garrison and Ollie the dog for giving me much needed sanity checks for the last two papers, and for having the patience to help me clarify the stories behind the papers. My last two papers would not have been the same without her insights and designer skills. I also want to thank Chaoran Fan for all the countless games of table tennis, surprising me and the rest of the group with really delicious special chinese snacks such as dried squid and cod. I thank Fourough Gharbalchi for her hospitality, and for on occasion joining the table tennis, and for bringing her little Instagram-star friend Barney to work. I want to thank Thomas Trautner for deep discussions about profound topics, his shared enthusiasm for fishing, and for helping arrange super nice trips and dinners together, and for taking the time to listen to rants about encountered challenges during the course of my studies. I also thank Eric Mörth for being a part of these things, and for teaching me the ways of flying drones, photography and building bamboo bikes. I also wish to thank Sergej Stoppel for arranging all kinds of social things before he left us, and for suggesting a "wear pajamas to lunch day", even though it never came to fruition I still think it is a great idea. I thank Eduard Gröller for coming to see all of us in the VisGroup, several times a year, and for his even more surprisingly strong table tennis skills. Related to this, I also wish to thank the table tennis group who I spent probably way too much time with: Torstein, Emmanuel, Emmanuel, and Chaoran. And I also thank Emmanuel Sam for helping me spend work time not working. I also thank Sherin Sugathan for sometimes joining for table tennis, and sharing knowledge and insights into his own work experience.

I also want to thank my family for supporting me through this period of my life. And I wish to thank Silje Sofie for supporting me through the ups and downs of this work, and helping me forget about work and just relax when I needed it the most.

Abstract

In recent times, the amount of data has been increasing massively. Analyzing this data with the help of visualization is highly useful, but also difficult for non-experts. Multiple views are typically used in conjunction to visualize more complex insights into data. Such multi-view visualizations are more difficult to design, more prone to design errors, and involve a more complicated design process. When designing a multi-view visualization, there are several aspects to consider: (1) how the visualization is *specified*, (2) means for extracting useful and actionable information, i.e., *reasoning* about the visualization, and finally (3) means for achieving a *refined*, or more *aesthetic* design. A user may be an expert in one or two of these aspects, but rarely all of them. In many cases, the user is not an expert in any of these aspects. With the increasing amount of data that needs to be analyzed and communicated, it is essential to make these aspects of multi-view visualization design more available to non-expert users through interactive, automatic and semi-automatic approaches.

In this thesis, we introduce approaches to make typically "expert-only" techniques more available to non-experts. Therefore, we first introduce an approach that lets users use simple charts as building blocks to create more complex charts by nesting them. This approach is powered by our novel data structure for specifying, editing, and rendering a nested visualization by editing a hierarchy of charts. Within this approach we also provide mechanisms for flexible data mappings, which allows for rapidly toggling between different visual representations, and for specifying visualizations less dependent on specific underlying data arrangements. The expressive power of these contributions was integrated into our visual builder, which ultimately enables non-experts to express highly detailed and aesthetic nested visualizations as demonstrated by a wide range of results generated with our approach.

Then, we proposed means for guiding users towards creating more consistent and compact multi-view visualization designs with our novel approach dubbed *semantic snapping*. With semantic snapping, the user is able to selectively apply operations to resolve potential design conflicts, such as inconsistencies, ambiguities, or redundancies. The core idea of our method is that each design can be positioned in a semantic space according to its degree of compactness and degree of consistency. We then define design anti-patterns as algebraic relations that are automatically detectable. Given these algebraic relations, we propose corresponding operations to remove relations from the design by making changes to individual elements. Based on these relations and operations, we provide a workflow wherein operations are selectively made available to the user, enabling incremental refinement of a multi-view visualization design. We demonstrate the utility and value of our approach with case studies where we show incrementally refined and ultimately improved multi-view visualization designs.

We then address the problem of multi-view visualization layout with our approach for making multi-view visualization layouts “content-driven”. From an existing grid layout, our approach generates an aesthetic, seemingly “hand-crafted” layout where elements are positioned according to their contents rather than bounding rectangles. This is achieved by using a force-directed layout to generate attraction and repulsion between elements. In this layout, attractive forces model the original arrangement of the grid layout, and are thus based on a minimal set of central elements derived from the original layout in an algorithmic fashion. The contents of the elements are modeled by repulsive forces, which are derived from the distance transforms of individual elements. This use of distance transforms for content-to-content repulsion handles highly irregular shapes, which allows for a better utilization of unused white space around the shapes. We demonstrate the value of our approach with case studies where our content-driven layout approach refines grid layouts with a high degree of unused white space into content-driven layouts that effectively utilize white space around irregular shapes, thus better capturing aesthetic qualities of an artistic design.

Abstract in Norwegian

I det siste har mengden data i omløp blitt stadig større. Det er nyttig å analysere denne dataen ved hjelp av visualisering, men det er også vanskelig for brukere som ikke er eksperter i dette feltet. For å vise mer innblikk i datasett brukes ofte flere visualiseringer på samme skjerm. Slik bruk av flere visualiseringer er vanskeligere å designe gitt en mer komplisert designprosess, og er derfor mer utsatt for designfeil. Det er flere spørsmål som må besvares som en del av designprosessen: (1) Hvordan visualiseringen er *spesifisert*? (2) Hvordan kan man automatisk *resonnere* om visualiseringer? Dvs. finne nyttig informasjon som kan brukes i praksis? Og (3) Hvilke måter kan vi automatisk gjøre et design *bedre* og mer *estetisk*? En person har kanskje ekspertise til å besvare ett eller to av disse spørsmålene, men sjelden alle. I mange tilfeller vil en person ikke inneha ekspertise til å besvare noen av spørsmålene. Gitt den økende mengden med data som trenger analyse er det viktig å gjøre slik designekspertise tilgjengelig til gjennomsnittlige brukere gjennom programvare og mer automatiske løsninger.

I denne avhandlingen introduserer vi metoder som gjør ekspertkunnskap tilgjengelig for brukere som ikke er eksperter. Først introduserer vi en metode som lar brukere bygge et hierarki av enkle visualiseringer som resulterer i mer detaljerte hierarkiske visualiseringer. Denne metoden er muliggjort av vår nye datastruktur for å spesifisere, redigere og tegne hierarkiske visualiseringer ved å redigere et enkelt hierarki av enklere diagrammer. Vår metode inkluderer også mekanismer for mer fleksible måter å oversette et datasett til en visualisering. Denne fleksibiliteten gjør det mulig å skifte hurtig mellom forskjellige visuelle representasjoner, og å spesifisere visualiseringer mer uavhengig av hvordan den underliggende dataen er arrangert. Vår metode gjør det mulig å uttrykke et bredt spekter av forskjellige visualiseringer som ellers krever mye ekspertise å lage. Dette er demonstrert av vårt program som lar ikke-eksperter uttrykke detaljerte og estetiske hierarkiske visualiseringer.

Vi skifter så fokuset over på metoder for å veilede brukere mot å lage mer konsistente og kompakte samlinger av visualiseringer med vår nye metode kalt *semantic snapping*. Med semantic snapping kan brukeren selektivt utføre operasjoner for å løse potensielle designkonflikter som for eksempel tvetydiget eller overflødighet. Hovedideen bak vår metode er at hver samling av visualiseringer kan plasseres på et plan i henhold til graden av kompakthet og konsistens. Vi definerer disse designkonfliktene som algebraiske relasjoner, dvs. automatisk detekterbare algebraiske predikater som beskriver forhold mellom flere diagrammer. Basert på disse predikatene introduserer vi operasjoner for å fjerne disse relasjonene, noe som lar brukere forbedre det helhetlige designet et steg av gangen. Vi setter så disse relasjonene og operasjonene sammen til en arbeidsflyt, hvor brukeren kan utføre en operasjon av gangen, og derved forbedre et design et steg av gangen. Nytten av vår metode er demonstrert gjennom våre eksempelstudier, hvor vi viser stegvis forbedring av en eksisterende samling av visualiseringer.

Til slutt ser vi på *layout* av en samling av visualiseringer, dvs hvordan de individuelle visualiseringene er posisjonert. Vi presenterer en automatisk metode for å posisjonere visualiseringer etter sitt innhold, heller enn en firkant rundt innholdet. Fra et eksisterende layout hvor elementene er plassert i rektangler, lager vår metode automatisk et nytt tilsynelatende "håndlaget" layout hvor elementene er posisjonert i henhold til sitt faktiske innhold. Vi oppnår dette ved å kjøre en fysikksimulering med attraktive og frastøtende krefter mellom elementene. Attraktive krefter emulerer de relative posisjonene i det originale layoutet. De frastøtende kreftene er regnet ut ved hjelp av Euklidske avstandsfelt, som støtter elementer uavhengig av form, som igjen fører til bedre bruk av ubrukt tomrom i det endelige designet. Vi demonstrerer vår metode med flere eksempelstudier hvor vi begynner med layout med en høy grad av ubrukt tomrom, og ender opp med mer kompakte layout som gjør bedre nytte av skjermplassen, og resulterer i et mer "håndlaget" design.

List of Papers

This thesis is based on the following publications:

(A) **Yngve S. Kristiansen** and Stefan Bruckner. **Visception: An Interactive Visual Framework for Nested Visualization Design**. In *Computers & Graphics (vol. 92)*, pages 13–27, 2020. (doi: https://doi.org/10.1016/j.cag.2020.08.007)

(B) **Yngve S. Kristiansen**, Laura Garrison and Stefan Bruckner. **Semantic Snapping for Guided Multi-View Visualization Design**. In *IEEE Transactions on Visualization and Computer Graphics (vol. 92, no. 1)*, pages 43–53, 2021. (doi: https://doi.org/10.1109/TVCG.2021.3114860)

(C) **Yngve S. Kristiansen**, Laura Garrison and Stefan Bruckner. **Content-Driven Layout for Visualization Design**. In *Proceedings of the International Symposium on Visual Information Communication and Interaction.*

The manuscripts presented in this thesis were written during the PhD studies of the main author. Stefan Bruckner is the main supervisor of the main author, and is thus a collaborating last author of all papers. Stefan Bruckner was a great inspiration for turning ideas into scientific contributions. His guidance and advice was essential for creating and publishing the works presented in this thesis. For all contributions, the main author was the driving force behind the realization and implementation. For paper B and C, Laura Garrison aided with inspiration and generation of results, as well as illustrations of presented ideas.

Contents

Part I

Overview

«I think I am, therefore, I am... I think.»
George Carlin

Chapter 1

Introduction

In recent times, the production and storage of data has increased, and continues to do so. To make better use of this data, many new methods and techniques have been developed across many research fields. One way to make use of data is to analyze it, and derive information which hopefully leads to better decisions. Such information can be derived by the use of machine learning, statistics, spreadsheets, or visualization. Visualization is a tool for analyzing and communicating data by taking advantage of human perceptual capabilities. For example, a line chart is both faster to read, and easier to understand than a table of numbers. Such visualizations are created and consumed by both experts and non-experts, leading to an overall increase in the use of visualization for data analysis. This increase calls for more accessible visualization techniques, especially for non-experts.

Visualization is comprised of several fields, focusing on different kinds of data, or different kinds of visual representations. One of these fields is called information visualization, and deals with visual representations of abstract data, i.e., data that does not have an inherent spatial component to it. For example, a table or a hierarchy is abstract data, while an X-ray scan is not. Such abstract data can be turned into visualizations in several ways. For example, a single row in a spreadsheet may be rendered as a dot in a scatter plot, as part of a line chart, or as a single bar in a bar chart. Design choices in this process are often subjective, and in many cases not ideal when the designer is a non-expert.

Design expertise is often not necessary to discover and communicate simple data insights with basic charts such as line charts, bar charts, and scatter plots. These basic charts are often expressible by non-expert users through most spreadsheet and business intelligence programs. However, it is difficult to express more complex insights with these solutions. Such complex insights can in most cases be visualized by one of several techniques proposed by information visualization researchers. Unfortunately, these techniques are often available to experts only. They are often available through programming libraries, or in some cases within visual authoring tools. Techniques available as programming libraries often abstract over some intricate details, but still require programming expertise. Thus, techniques are sometimes made available through visual authoring tools, which require only user interaction. Visual authoring tools are easier to use, but also complicated to create, as they must provide sensible, ideally simple interactions to specify complex charts. It is therefore common that visual builders are narrowly scoped to a specific way of interaction for specifying charts. This narrow scoping often comes at the expense of the flexibility and design freedom found in programming libraries.

Complex insights into data are not only expressed with complex charts, but also by using multiple charts in conjunction. This is often done within business intelligence environments such as Microsoft PowerBI and Tableau [97], where users can create multiple charts and arrange them in a grid-layout dashboard. While some means for making multi-view visualizations are available, multi-view visualizations are far less explored and researched than single visualizations. This gap has been partially addressed by recent research efforts to integrate novel visualization techniques and knowledge into existing environments [61, 82], so that they can be used in conjunction. However, most techniques are still reserved for experts only.

In this thesis, we aim to make novel multi-view visualization techniques and design expertise more available to non-expert users through user interfaces exposing automatic and semi-automatic techniques. First, we explore the idea of using nesting as a first-class operator, and contribute a novel data structure for simplifying the specification of nested visualizations. Along with this structure, we propose four approaches for flexibly adapting visualization specifications to different underlying data formats, which unburdens users from tedious data wrangling efforts. We then expose the flexibility and expressiveness of our data structure and flexible data mappings through a visual builder whose expressiveness is demonstrated by our examples. Then, we provide means for guiding users through the visualization design process with our novel approach of *semantic snapping*, which is a visual “grammar checker” that lets the user incrementally refine a multi-view visualization design. Design anti-patterns are defined as a set of algebraic relations which are automatically detectable. Based on these relations, we propose corresponding operations which resolve design anti-patterns and thus improve the design. We demonstrate the power of our approach with step-by-step case studies where bad designs are turned into good designs with incremental refinements generated by our approach. Finally, we address multi-view visualization layout with our approach for turning the ubiquitously used grid layout into a *content-driven* layout that better makes use of previously unused white space within the grid layout. Our content-driven version of the layout is powered by a force-directed approach, where the forces are inferred from the original layout and its elements. The original layout arrangement is emulated by selective application of attractive forces derived from the original grid layout topology. Precise content-to-content repulsion computation is inferred by utilizing the distance transform of elements to infer accurate distances and directions of repulsion. We demonstrate the utility of our approach by applying it to existing grid layouts with high degrees of unused white space, resulting in seemingly artistic “hand-crafted” layouts.

1.1 Problem Statement

The research presented in this thesis is motivated by challenges and problems arising when designing multi-view visualizations. Especially for non-expert users, recent developments in information visualization are hard to take advantage of as they require either design expertise, or programming expertise. More precisely, we address the following challenges of empowering non-expert users to: (1) flexibly specify both individual and nested visualizations, (2) design multi-view visualizations that communicate unambiguously and concisely to the audience, and (3) create aesthetically pleasing multi-view visualization layouts that appear similar to manual artist designs. Most designers are non-experts in at least one of these aspects of visualization design, since they are most likely not aware of, or able to properly overcome all of these challenges without guidance or expertise.

Business intelligence systems such as Tableau and PowerBI are frequently used to express arrangements of simple charts such as bar charts and scatter plots. More expressive or flexible charts remain expressible only for expert users. For example, most users are able to create a set of bar charts and scatter plots, and arrange them as a dashboard. However, it is typically much more difficult to express the same data insights with more unusual charts such as a treemap or a stream graph. Furthermore, users are rarely able to use multiple charts in conjunction by nesting, juxtaposing, or overlaying without tedious manual work such as coding it by hand, or using design tools such as Adobe Illustrator to achieve the desired result. Enabling such expression of both complex and conventional single visualizations, and for arranging such visualizations in a multi-view visualization without manual coding or design efforts remains a challenging task.

While a user may be very fluent in creating highly detailed custom charts, that user is not necessarily an expert in visual communication. For example, the design may be what the user intended, but also be misleading or confusing to the audience by having ambiguous mappings. Current bodies of work on design guidelines, quality measures, and pitfalls are spread across many different individual contributions. These contributions are in many cases targeted towards specific tasks and domains. Efforts have been made to consolidate this knowledge to make it more accessible to non-experts. Yet, this remains a challenge that still needs solutions.

In other cases, aesthetics of a visualization are highly important. Such aesthetic visualizations are typically hand-crafted by expert designers or artists, often with a high level of visual communication expertise. However, with the use of visualization design software among non-experts, there is a potential for bad design choices which may lead to confusing or misleading visualizations of data. This poses the research challenge of granting non-expert users expert capabilities through automatic or semi-automatic design tools.

1.2 Scope and Contributions

This thesis expands the state of the art on several aspects of multi-view visualization design. The contributions of this thesis can be summarized as follows:

1. **Specification.** Based on the core idea of using nesting as a first-class operation, we contribute the Visception Tree (shorthand: VC-tree) data structure that allows for precise control data mappings at different hierarchical levels, as well as implicit handling of nesting and deformation behaviors. Furthermore, we contribute means for making visualizations less dependent on a specific tabular data

arrangement by providing four different ways of expressing a chart's data grouping. We expose the expressive power of the VC-tree and our flexible data mappings through our visual builder, which has a correspondingly high degree of design expressiveness as shown by our results.

2. **Reasoning.** We enable the transformation from one chart into another by specifying equivalences between visual channels. Furthermore, we contribute means to extract useful, actionable knowledge about an existing multi-view visualization design by specifying algebraic relations between individual views.
3. **Aesthetics & refinement.** We enable gradual removal of potentially ambiguous or redundant design conflicts from a multi-view visualization with our novel semantic snapping algorithm, which enables the user to remove automatically detected problems by performing a corresponding operation with a single click. Furthermore, we provide means for making the layout of a multi-view visualization content-driven, i.e., positioning elements closely with regards to their contents, rather than bounding rectangles.

1.3 Thesis Structure

This thesis consists of two parts. First, we show an overview of our research contribution, then we present individual publications. The individual publications are presented verbatim, with only adjusted formatting to fit the layout of this thesis. The bibliographies of the individual papers are merged into a single, unified bibliography.

The overview part of our thesis is structured in the following way: Chapter 1 provides an introduction to the topic of the thesis, the general problem to be solved, and provides the scope for our achieved contributions. In Chapter 2, we discuss related work on multi-view visualizations, including visualization specification, multi-view visualization, and layout strategies. Based on this related work, Chapter 3 outlines the contributions of this thesis, along with results to demonstrate the utility and value of each contribution. Chapter 4 concludes the thesis by providing a direction for future work on multi-view visualizations.

«If I have seen further it is by standing on the shoulders of Giants.»
Isaac Newton

1

Chapter 2

State of the Art

In this thesis, we present novel methods for guiding multi-view visualization design. Our first contribution is based on the core idea of using nesting as a first-class operation, which leads to significant design flexibility and freedom. Then, we contribute an approach for guided multi-view visualization design, where users are able to review and resolve automatically detected problems. Finally, we introduce an approach for generating *content-driven* layouts from existing grid layouts that are often used for information visualization dashboards. Our work contributes to the field of information visualization.

In this chapter, we discuss the state of the art for multi-view visualizations. We begin by reviewing approaches for specifying visualizations, both general and more specific approaches. First, we detail approaches for specifying visualizations, namely visualization grammars and toolkits. We then focus on techniques for hierarchical visualizations, including layout techniques and grammars for hierarchical visualizations. We continue to discuss interactive visualization specification, i.e., visual builders. Then, we discuss means for evaluating and generating visualizations, namely visualization quality metrics and recommender systems. We then shift our focus more specifically towards multi-view visualizations, discussing works on visualization dashboards, and semantics specific to multi-view visualizations. Finally, we discuss layout strategies used in the context of multi-view visualizations in two parts: First, we discuss content-aware strategies, where approaches for positioning multiple views by their contents are discussed. Secondly, we discuss techniques that fully or partially use automation to generate layouts. We provide a more detailed discussion of the relations between previous work and our own contributions in the individual related work sections of paper A, B, and C.

2.1 Visualization Specification

Before computers were used to display pixels on a screen, data visualizations were often crafted by hand. Specifying such visualizations is rarely trivial. This problem of *visualization specification* has been addressed by many researchers through a wide range of grammars, toolkits, techniques, and visual builders. These approaches share the goal of our thesis, which is to enable users to specify visualizations of abstract, non-spatial data. However, this act of specifying a visualization can be partially or fully automated. Automation provides less human control over details, but also re-

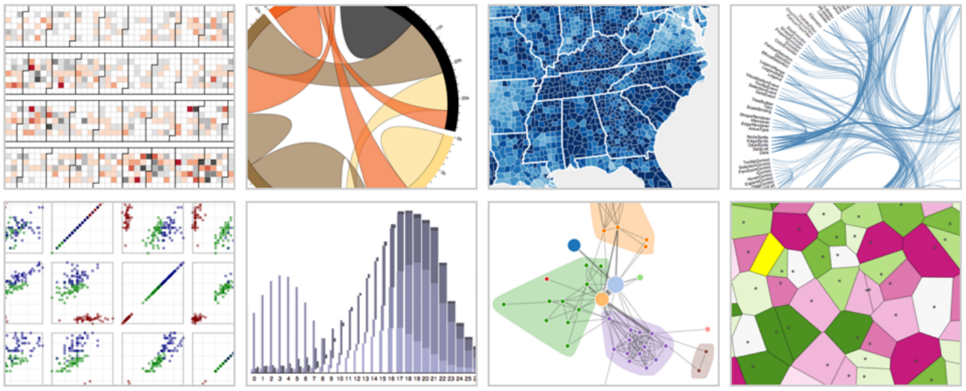


Figure 2.1: Here, the expressive power of the well-known D3 [14] library is illustrated with a few examples, all of which are specified mapping data to DOM elements.

quires less effort and involvement. Thus, a large body of research exists on the topic of *visualization recommendation*, focusing on automating more of the design process. When a human is involved in the design process, subjective evaluations and decisions steer incremental refinements towards the final design. If a computer is doing the design, it is necessary to replace these subjective evaluations and decisions with *quality metrics* that can be automatically computed. In this section, we discuss approaches that enable for specification and evaluation of visualizations, with varying degrees of human involvement. The focus of this thesis is on interactive multi-view visualization design, and moves towards increasingly automated approaches. Similarly, the focus on these related works begins with fundamental works for visualization specification, moving towards works involving a higher degree of automation. We first introduce fundamental works on *visualization grammars and toolkits*, before describing more specific techniques and grammars for specifying *hierarchical visualizations*. We then move our focus to *visual builders* that allow users without coding expertise to interactively specify visualizations. Finally, we focus on techniques for automatic generation and evaluation of visualizations, namely *visualization recommender systems* and means for *evaluating visualization quality*.

2.1.1 Grammars and Toolkits

One of the most basic ways to express a visualization with a computer is through a visualization grammar or toolkit. Such grammars and toolkits allow users to express visualizations of data that would otherwise be difficult to create. For example, a grammar might let a user express “Create a bar chart where each bar represents one age group, where the height is proportional to the income”. Ideally, grammars are are easy to understand and demonstrate a high freedom of expression. The earliest, most basic grammars were created to support basic statistical graphics such as scatter plots, bar charts, line charts, and box plots. With the development of programming languages and computer-generated graphics, grammars were adapted into *visualization toolkits*, allowing expert programmers to proficiently and concisely specify graphics from data. While these toolkits were less accessible to non-experts, they were also highly expressive. This was very useful for expressing charts with many components and mappings, and made it easier to express such charts. Such grammars and toolkits lay the foundation for further research on visualization specification, and are thus also fundamental to the contributions of this thesis.

Some of the most well-known **grammars of graphics** are Bertin's Semiology of Graphics [10] and Wilkinson's Grammar of Graphics [109]. These grammars describe graphics in terms of different units and visual variables that make up the resulting visualization. For example, Bertin's Semiology of Graphics describes a chart as a set of basic graphical units, and styling applied to these. If we consider a scatter plot, the basic graphical unit is a circle, and the position, texture, color of the circles are described as visual variables. Munzner [73] made further efforts to consolidate existing grammars. She proposed that charts can be specified in terms of *marks* and *channels*, where marks denote the basic graphical units of the chart, and channels control the appearance of these marks. For example, the marks of a bar chart describe the rectangles that make up the bars, while its channels describe how the height and position and of the bars are generated from data. These grammars facilitate discussion of both existing and novel visualization techniques. With the development of programming languages and computer-generated graphics, a new set of visualization toolkits emerged. These toolkits were often based on fundamental grammars, and provided constructs to be used in a more proficient and concise manner by people with programming experience. Among the first of these toolkits is Prefuse [39], a programming library for interactive information visualization specification. It is a software framework for creating dynamic visualizations. Within this framework, programmers can string together reusable components to create their visualizations. Such abstractions are useful for specifying a wide range of visualizations. However, Bostock and Heer [13] found that there was a gap between low-level graphical systems typically used by designers, and high-level visualization systems often used by analysts. Designers would typically be *visually thinking*, i.e., think of the visualizations in terms of shapes, for example "create one bar per age group, and adjust the height according to income", whereas the high-level visualization system abstraction might be "create a bar chart of age group and income". In response to this gap, they introduced ProtoVis [13], a toolkit for specifying visualizations by means of visual thinking. Based on the underlying constructs of ProtoVis, Bostock et al. [14] later introduced Data-Driven Documents (D3), a well-known and widely used framework for creating web visualizations in the browser by directly mapping data to DOM-elements. This direct mapping enables leveraging existing capabilities within the web browser for creating animations and interactions, as well as further customizing the visualization after creation as exemplified in Figure 2.1.

While these toolkits allow for visual thinking, they still require manual coding of features like interaction and view composition. Manually re-implementing such features for every chart can be tedious and time consuming, and is rarely necessary. Declarative, high-level visualization grammars such as Vega [112] simplify this process by enabling users to simply declare a visualization along with its desired mappings and interactions at a high level. A Vega chart is made up of input data, a mark type, and visual encodings. Each encoding specifies how a channel is mapped to data. Vega-Lite [87] provides further functionality by adding the option to declaratively specify interaction, and providing a view composition algebra with four operations for composing several views in different ways as seen in Figure 2.2. These grammars and toolkits

```
{
  "repeat": {"column": ["hour", "delay", "distance"]},
  "spec": {
    "select": {
      "region": {
        "type": "interval",
        "project": {"channels": ["x"]},
        "resolve": "intersect_others"
      }
    },
    "data": {"url": "data/flights-2k.json"},
    "transform": {
      "filterWith": "region",
      ...
    },
    "mark": "bar",
    "encoding": {
      "x": {"field": {"repeat": "column"}, "bin": true, ...},
      "y": {"aggregate": "count", "field": "*", ...},
      "color": {"value": "steelblue"}
    }
  }
}
```

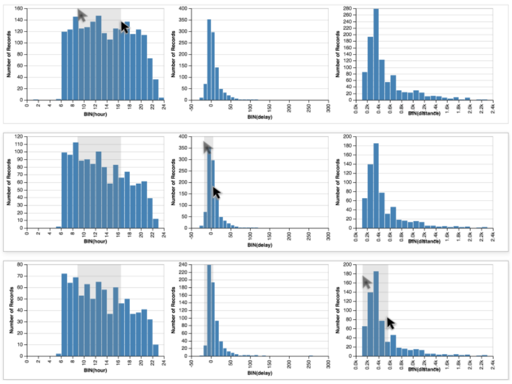


Figure 2.2: This figure shows a Vega-Lite [87] specification on the left, and a corresponding visualization on the right. The code that is highlighted in green creates the interactive brush as seen on the leftmost bar charts, which filters what is seen on the charts to their right.

greatly simplify the visualization creation process, but fall short when it comes to more specific and complex charts. While it is possible to specify complex hierarchical visualizations with visual thinking based toolkits such as D3 [14], the complexity of the specification is often too high for most non-expert users. Declarative grammars may support some specific complex hierarchical charts, but are not able to express such complex charts at a general level. Paper A [56] aims to enable users to interactively specify such charts at an arbitrary level by utilizing nesting as a first-class operator, which both simplifies the design process while also allowing for expressing highly complex designs.

2.1.2 Hierarchical Visualizations

Expressing elaborate visualizations of hierarchical nature is complicated, and is thus often not fully covered by generalist grammars such as Vega [112], or easily expressible with visual thinking frameworks. Therefore, specific grammars and techniques for creating such hierarchical visualizations have been proposed. For this category of visualizations, we focus on works that address at least one of the following: (1) specific layout techniques for hierarchical visualizations, (2) grammars for declaring hierarchical visualizations, (3) specialized grammars and techniques for composing and combining visualizations by using high-level operations.

Hierarchical layout techniques are often described in terms of several layout stages or operators that can be combined. The use of such operators can be seen throughout several works. For example, ZAME [28] (Zoomable Adjacency Matrix Explorer) allows for the exploration of large datasets by embedding detail information into glyphs displayed within cells of an interactive adjacency matrix. The interactivity of this adjacency matrix comes in the form of interactive zooming and panning functionality, which enables closer inspection of detail information within each cell. NodeTrix [40] shows more data and relations with less visual space by combining the node-link diagram and adjacency matrix into one technique, where each node is an adjacency matrix of its own, and links are drawn between cells of different adjacency matrices to

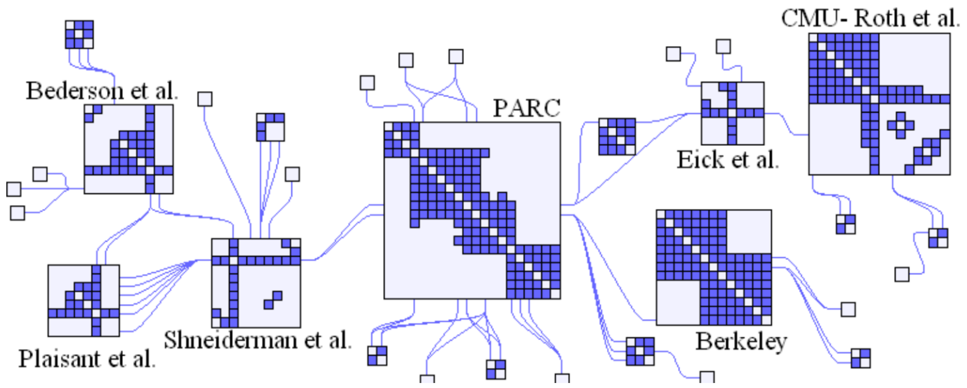
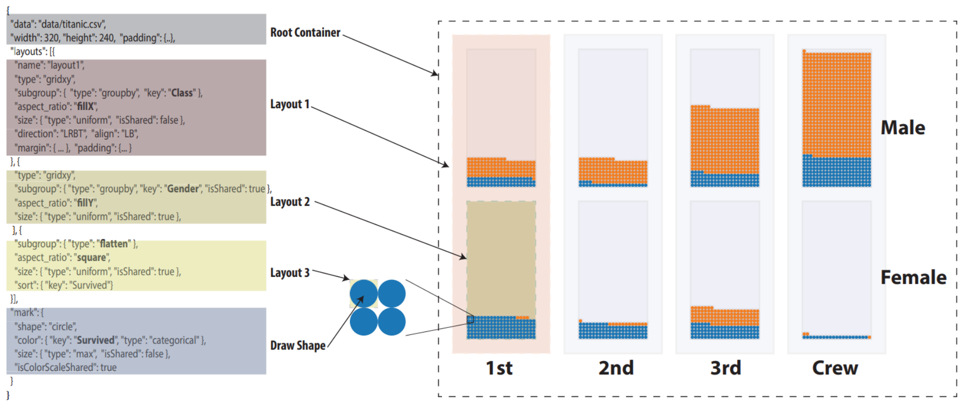


Figure 2.3: Here we see Nodetrix, a hybrid visualization combining a node-link diagram with adjacency matrices in order to better illustrate relationships in social networks.

illustrate relations efficiently as seen in Figure 2.3. Similarly, Domino [37] combines different techniques to achieve better comparison of data across subsets of multiple datasets. More specifically, they define individual charts as *blocks* representing rectangular regions corresponding to subsets of data. These blocks can be combined or linked together to further expose relationships within the data.

Schulz et al. [91] surveyed the design space of 2D and 3D hierarchy visualization techniques, and exposed unexplored parts of the design space. Furthermore, they contributed an implementation for generating visualizations within this design space, which proved highly useful for comparing different designs. View-composition techniques typically used for non-hierarchical data also prove to be useful when visualizing hierarchies. Javed and Elmqvist [49] unified existing coordinated multiple view techniques with other strategies for composing visualizations. From existing literature, they derived four operators: juxtaposition, superimposition, overloading, and nesting. Throughout their survey, they illustrate how these four operators are used to generate different designs. LeBlanc et al. [57] introduced the technique of dimensional stacking, backed by the core idea of displaying multiple dimensions aggregated within existing dimensions. This ultimately lets users map high-dimensional data to a smaller 2D space. Such stacking of dimensions can be used with different layouts. For example, the treemap [17] layout recursively subdivides a rectangular space into data-proportional sections, and is well-suited for showing hierarchies. Such treemap techniques were explored further by Baudel et al. [8], who captured the treemap design space with an algorithm for expressing different existing treemap layouts by using a few basic operations. Similar to treemaps, the circle packing layout [106] creates circles nested within other circles, corresponding to data hierarchies of arbitrary depth. Such packing can also be achieved by nesting circles with a force-directed layout [33], which simulates circles as physical entities that repulse each other upon collision. Nested hierarchical structures have also been explored by using large 3D nested graphs as proposed by Parker et al. [77]. This large 3D graph made it easier to understand complex relationships within software. Schulz et al. [90] applied the idea of composing existing techniques to explicit node-link diagrams, and proposed a generative approach for expressing tree and node-link visualizations in terms of operators that can be applied to the layout at a certain stage.

When novel visualization techniques are introduced, they are often unfamiliar to non-expert users. Loorak et al. [63] addressed this problem by enabling the *extension* of well-known, familiar visualizations with their component dubbed HEDA (Heterogeneous Embedded Data Attributes). This component allows for extending existing visualization techniques, while respecting an original layout. Similarly, Slingsby et al. [95] proposed the nesting of different layouts to display hierarchies of data with their proposed language HiVE (Hierarchical Visualization Expression Language). This language provides operations for editing, deleting, inserting and swapping at different hierarchical levels. Specific grammars for hierarchical visualizations have also been proposed. Li et al. [58] proposed a declarative tree visualization grammar where users can express visualizations both explicitly and implicitly. With their visual builder, users are also able to combine different tree layout algorithms and adjust finer aspects which would otherwise have to be coded by hand. ATOM [76] is a grammar for specifying unit visualizations, i.e., visualizations where every row in a tabular dataset is represented as one graphical unit. This grammar allows for subdividing a space as multiple levels, filling in one unit per datum as shown by Figure 2.4. Wickham and Hofmann's Product Plots [107] similarly combine 1D primitives to express area-based visualizations. With the three base primitive (bars, spines, and tiles) it is possible to express a wide range of both simple and complex visual representations of data. Schulz and Hadlak [92] also proposed means to represent visualizations by blending together existing visualizations defined as presets. This blending allows for smoothly interpolating between different visual representations, allowing for the expression of smooth animations. Vuillemot and Boy [105] proposed a tool for rapidly prototyping complex visualization designs by compositing and nesting visualizations regardless of data. They do this with a visual grammar made up of partitioning patters and data transformation operations.



```
{
 "data": "data/titanic.csv",
 "width": 320, "height": 240,  "padding": {..},
 "layouts": [{
  "name": "layout1",
  "type": "gridxy",
  "subgroup": {  "type": "groupby",  "key": "Class" },
  "aspect_ratio": "fillX",
  "size": { "type": "uniform",  "isShared": false },
  "direction": "LRBT",  "align": "LB",
  "margin": { ... },  "padding": {... }
 }, {
  "type": "gridxy",
  "subgroup": { "type": "groupby", "key": "Gender", "isShared": true },
  "aspect_ratio": "fillY",
  "size": { "type": "uniform", "isShared": true },
  }, {
  "subgroup": { "type": "flatten" },
  "aspect_ratio": "square",
  "size": { "type": "uniform", "isShared": true },
  "sort": { "key": "Survived"}
 }],
 "mark": {
  "shape": "circle",
  "color": { "key": "Survived", "type": "categorical" },
  "size": { "type": "max",  "isShared": false },
  "isColorScaleShared": true
 }
}
```

Figure 2.4: To the left, we see an ATOM [76] specification of the chart seen on the right. The chart illustrates survival rates by gender and class on the Titanic, where each individual circle represents a single person.

These works all aim to visualize hierarchies of data, and enable for simple expression of otherwise complicated hierarchical layouts. They show the trend of techniques moving towards flexible composition of existing approaches, rather than inventing custom methods for particular purposes and datasets. Furthermore, we see that grammars tailored towards certain categories of visualizations have been developed. These works use nesting and composition of different techniques as a means to an end, but never as a first-class operator. In paper A [56], we address this gap by making nesting a first-class operation, putting the user in control of the hierarchy of data as well as the corresponding hierarchy of charts. This core idea makes it possible to create nested visualizations similar to those expressible by specialized grammars in an interactive manner.

2.1.3 Visual Builders

While grammars and toolkits allow for specification of visualizations, they also typically require coding skills. Even declarative approaches can be confusing to users with little coding experience. For such users, the use of a graphical user interface is a better option as it requires no programming expertise. Such systems for designing visualizations interactively are often referred to as visual authoring tools, or visual builders.

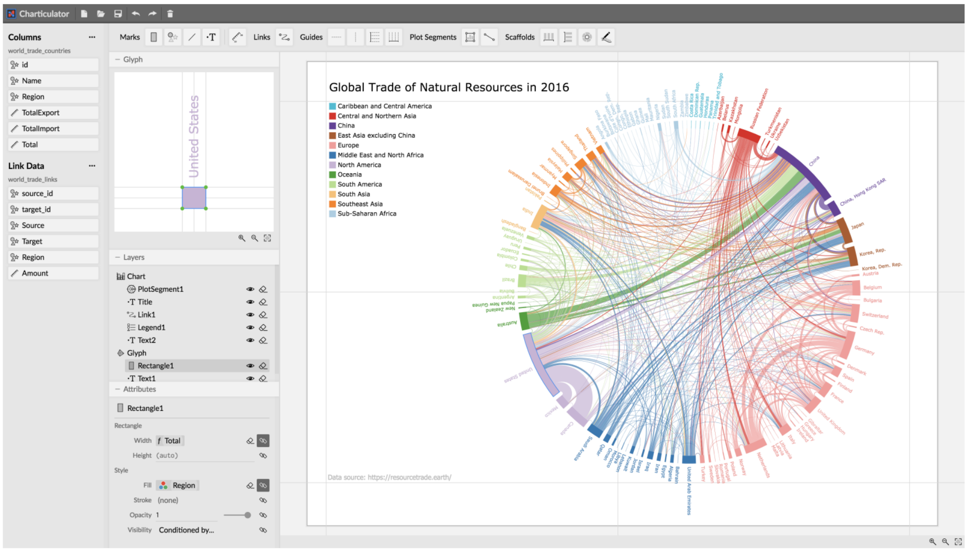


Figure 2.5: Here we see an overview of the Charticulator [82] user interface, where users can combine different glyphs and marks to specify highly flexible visualizations.

The first visual builders were primarily focused on the exploration of data, rather than design and aesthetics. IVEE [2], Visage [83], and Tioga2 [3] were among the first systems that enabled visual building of database queries with resulting visualizations. Polaris [97] by Stolte et al. was later commercialized as Tableau, and enables the rapid exploration of large multidimensional datasets by using a table algebra to display a variety of charts. With the improvement in computer performance and the development of new techniques, new kinds of visual builders were explored by researchers. These builders are not primarily focused on showing insights into the data, but also consider on *how* the insights into the data are shown. In other words, these new builders provide more design freedom and expressiveness than previous data exploration oriented tools.

For example, Lyra [85] lets users interactively create highly customized visualizations with drag-and-drop operations. It also lets users express advanced layouts and data transformations through its graphical interface. iVisDesigner [81] is another example of a visual builder, which also enables the expression of a wide range of different visualizations by providing a high degree of conceptual modularity. Following a similar line of thought, Charticulator [82] enables the specification of custom shapes by specifying data-driven compound marks, glyphs, and links through a visual builder as seen in Figure 2.5. Likewise, Data Illustrator [61] aims to provide a "data-driven" design process similar to the workflow of Adobe Illustrator. It lets users specify data-driven vector shapes within vector design tools by leveraging new concepts and operators for binding vector-based components to data. Kim et al. [52] proposed Data-Driven Guides, a technique for visually building embellished charts with highly customized shapes as seen in Figure 2.6. While authoring systems let users specify layouts interactively, iVolver [74] allows for interactively extracting data from visualizations.

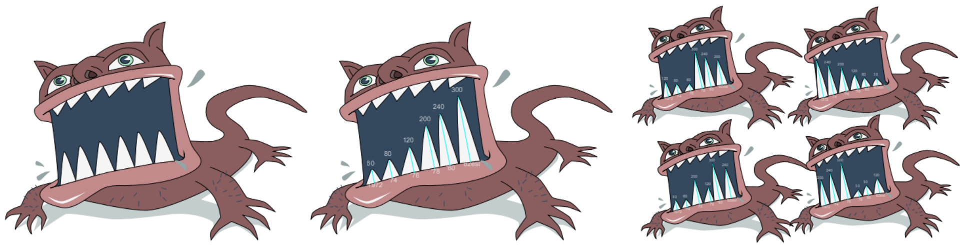


Figure 2.6: Here we see Nigel Holmes' *Monstrous Costs* chart, recreated with the Data-Driven Guides [52] technique for creating embellished visualizations. The chart is recreated by first importing the monster on the left, and then mapping data to the height of the triangles representing its teeth.

Such visual builders often address very or slightly different problems, and are thus highly difficult to compare. To address this difficulty in comparing systems, Satyanarayan et al. [86] assessed three recent systems, namely Data Illustrator [61], Charticulator [82], and Lyra [85]. From this study, they proposed a set of criteria to evaluate such authoring systems on a general level.

These builders all provide innovative ways to specify a wide range of charts, yet provide limited support when it comes to nesting of charts. Paper A [56] addresses this by making nesting available as a first-class operator in a visual builder, letting users express highly complex layouts by nesting of simple, understandable layouts.

2.1.4 Visualization Recommendation Systems

Heavy focus on expressive power provides great design flexibility, but also relies on user expertise in creating good designs. For users without such expertise, guidance systems embedded into design tools would be of great benefit. However, such guidance systems for refining and evaluating existing designs are most developed within visualization recommendation systems. We will now discuss such works that recommend, refine, or suggest improvements for existing designs. These works are important since they can effectively make expert knowledge and design guidelines available to users in an automatic or semi-automatic fashion. Such tools for automatic generation of infographics and visualization recommendations were surveyed by Zhu et al. [118], where they classified approaches into a set of application categories including network and graph visualization, annotation visualization, and storytelling visualization. Such automatic approaches lower the barrier of entry for novice users. Grammel et al. [36] found that novice users would greatly benefit from a tool that supports iterative well-explained refinements to existing visualizations. In paper B, we address this need with an approach that provides users the ability to detect potential problems in a multi-view visualization, and resolve them with corresponding operations. One of the first tools to automatically generate visualizations from data is MacKinlay's APT (A Presentation Tool) [64]. In this work, he proposed a composition algebra for automated visualization design based on Cleveland and McGill's effectiveness metrics [22]. These metrics judge how effective a human is at evaluating different aspects of 2D visualizations such as positions on a common scale, length, area, angle, and curvature. Wongsuphasawat et al. later introduced CompassQL [111], a general language for querying the space of visualizations, which has been used to power visual recommendation systems. Voyager [110] is an example of such a system powered by CompassQL, and enables data exploration by browsing of automatically generated visualizations. Voyager 2 [113] improves further on the state of the art by letting users partially specify what data should be shown, resulting in a set of automatically generated charts showing related data. Other approaches to recommend visualizations have also been explored. For example, Data2Vis [26] formulates visualization generation as a language translation problem, where a data specification is mapped to a Vega-Lite [87] specification. Within the visualization design environment of Tableau, Show Me [65] provides means for displaying additional data attributes on a chart with a single click, and serves as a set of high-level commands for showing multiple fields within a single view. Draco [72] enables the specification of visualization design guidelines as precise constraints accessible through an Answer Set Programming environment. In this framework, single visualizations are modeled as sets of logical facts, and design guidelines are represented as hard and soft constraints over these facts. Dziban [60] extends Draco further by providing anchored recommendations, i.e., where recommended charts are perceptually similar to a provided "anchor" chart.

These approaches all focus on helping users more easily create visualizations without having to do detailed design work manually. However, they only propose different visualizations given a dataset and some information about what is shown. The approaches are often focused on what data is shown, and less so on exactly how it is shown. Furthermore, they often only generate single visualizations, and rarely consider problems that are caused by multiple visualizations. In paper B, we provide an approach for generating and recommending incremental refinements to multi-view visualizations. These increments come in the form of operations that remove potentially undesirable relations from the design.

2.1.5 Evaluating Visualization Quality

The quality of a visualization can be gauged by several aspects. Examples of such aspects include aesthetics, correspondence to underlying data, or that the visualization correctly communicates the intended message to the audience. Due to visualizations being used for many applications across different domains, quality measures are similarly often targeted towards different domains and use cases.

Several works have proposed a wide range of different quality measures to evaluate visualizations. These measures provide means for reasoning about different aspects of a visualization, such as its effectiveness or level of ambiguity. For example, Behrisch et al. [9] categorized quality measures from about 250 papers, most of which were specific to a certain combination of underlying data, task, and visualization technique. Such specifically targeted measures complicate the process of making them available on a general level. This problem has been recognised, and efforts have been made towards unification of these scattered works, aiming to make them available, understandable, and actionable to users.

Bolte and Bruckner [11] surveyed measures by which aspect of the visualization process they target: Perceptual characteristics, task-oriented quality measures, structure-oriented measures, and meta-perceptual processes. Zhu [119] found in a literature review that definitions of visualization effectiveness are often incomplete, and in some cases conflicting. He pointed out that measures usually take either a data-centric view or task-centric view on what constitutes an effective visualization. Data-centric measures consider how accurately a visualization corresponds to underlying data, while task-centric measures consider how well-suited a visualization is for the intended task. For example, Tufte's data-to-ink ratio [102] exemplifies a data-centric metric since it describes a desirable relationship between the data and its visual representation. Kindlmann and Scheidegger's algebraic framework [54] is another example of a data-centric framework for measuring visualization effectiveness as it only considers symmetries between changes in data space and resulting changes in visualization space. Partially based on this algebraic framework, McNutt and Kindlmann [67] proposed a mechanism for detecting and linting potential problems in a single design. With better, more universal quality metrics, such linting of existing visualizations can be highly helpful for users. However, this kind of automatic detection of problematic designs has only been realized for single visualizations, and not for multi-view visualizations. In paper B, we provide an approach to automatically detect and resolve problems in multi-view visualization designs.

A visualization can also be measured by how it is perceived by the observer. Cleveland and McGill's graphical experiments [22] revealed how humans perform in elementary visual tasks such as comparing positions on a common scale. Correll et al. [23] discuss how designs may appear to be showing data completely, while still hiding important details. They further propose a set of actions to remedy such vulnerabilities across different chart types.

Other measures consider both task and data. Cantu et al. [18] presented an approach for identifying relationships between visualization challenges and representation components. They argued that these relationships can increase our understanding of mechanisms behind visualization components, which could be leveraged to build visualization recommendation tools.

As shown by Zhu [119], different sets of guidelines and measures are often disjoint and sometimes conflicting. Efforts to bring these different viewpoints closer have been made by Diehl et al. [27] through the VisGuides forum which facilitates both collection and discussion of visualization guidelines and knowledge in general. While there is much knowledge in the visualization community in terms of formulating and creating

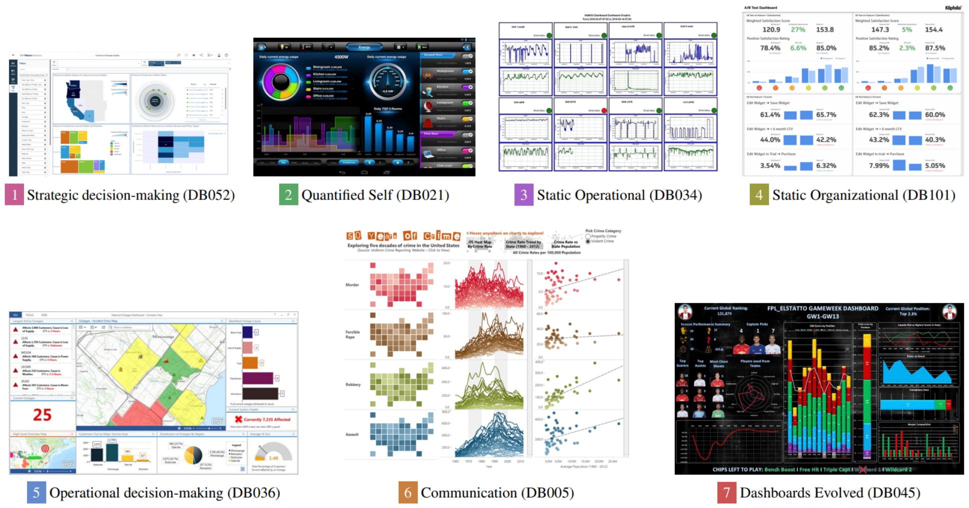

1 Strategic decision-making (DB052)
2 Quantified Self (DB021)
3 Static Operational (DB034)
4 Static Organizational (DB101)
5 Operational decision-making (DB036)
6 Communication (DB005)
7 Dashboards Evolved (DB045)

Figure 2.7: This figure shows seven clusters of dashboards for different uses, identified by Sarikaya et al. [84] in their survey on dashboards.

such design guidelines, Engelke et al. [30] point out that there is a great gap between the visualization community, and users who are most in need of such guidelines. To remedy this, they provide VISupply [30], a conceptual model that highlights problems and opportunities for bringing visualization guidelines to non-experts. Paper B contributes towards this gap, as it automatically detects problems and solutions that are presented to the user in an easily understandable and actionable manner.

2.2 Multi-View Visualizations

While many established quality metrics can be applied to both multi-view and single-view visualizations, multi-view visualizations are subject to different design challenges and problems. In addition to single-view visualization metrics, a multi-view visualization can be evaluated by its layout and relationships between several views. Potential problems in a multi-view design are in many cases not isolated to a single view, as there may be a problematic relationship between multiple views. Furthermore, the layout, i.e., the positioning of elements relative to each other in a multi-view visualization greatly impacts the design. For example, in the case of UML diagram layouts, Störrle [98] found that good layouts were much easier to understand for novices.

Dashboards are used for several purposes. They provide a visual overview of key insights into data that inform analysts and decision makers rapidly. Describing exactly what constitutes a dashboard, and enumerating the different kinds of dashboards and how they are to be used remains a large and difficult task. However, researchers have made several efforts to consolidate and describe this design space. In order to enable for specification, reasoning and refinement of such dashboards, knowing the design space and its possibilities is essential.

Sarikaya et al. [84] analyzed multiple dashboards found "in the wild" to construct a design space of dashboards. In this design space, they derived seven clusters of dashboards as seen in Figure 2.8. Furthermore, they mention that dashboards are venturing more and more into the realm of infographics. One implication of this is that dashboards will have more artistic and flexible layouts.

Ondov et al. [75] conducted a series of graphical perception experiments to find which compositions of multiple charts are effective for different tasks. Chen et al. [20] investigated multi-view visualization design patterns from 360 multi-view visualizations collected from IEEE VIS, EuroVis and PacificVis publications from 2011 to 2019. They identified common multi-view visualization practices, view layouts, view types, and correlations between view layouts and view types. They made these patterns available through a multi-view visualization recommendation system where users can interactively browse different designs. Hoffswell et al. [41] proposed a prototype system for previewing and editing multiple visualization versions simultaneously. This system is motivated by the analysis of 231 responsive news visualizations and interviews with journalists. Most of the existing work on dashboards presume the use of a grid layout. However, with the onset of more "infographics-like" dashboards [84], more flexible layout techniques are required.

Semantics must also be considered when evaluating a multi-view visualization design. Defining such semantics formally allows for automatic reasoning which may be used to partially or fully aid the design process. For example, in graphic design tools, conventional snapping creates a "gravity field" around geometric objects, making it easier to place elements closely together. This snapping can also be done with regards to semantics, which is exemplified by Hudson's [43] introduction of semantic snapping as an interaction technique for geometrically snapping objects together only if the objects are *semantically* related. Such semantic relations were also explored more closely by Shadoan and Weaver's [93] flexible interactive query language that allows for exploring semantic relations in multi-dimensional data by cross-filtering on attribute relationship graphs. Kosslyn [55] proposed a framework for evaluating the semantic clarity of a visualization with five specific criteria. Furthermore, Kosslyn recommended this framework for both single-view and multi-view visualizations. However, very few examples of its application to multi-view visualizations exist. Qu and Hullman [79] were among the first to discuss ways to operationalize Kosslyn's principles for multi-view visualizations. They derived two constraints: C1 (encode the same data in the same way), and C2 (encode different data in different ways). In their following work [80], they conducted a Wizard-of-Oz study where Tableau users often and unknowingly respected their proposed constraints C1 and C2. In this study, they found that participants were highly positive to having a consistency checker tool to inform about violations of these constraints. While these works explore the use of semantics to aid and guide visualization design, they do not yet provide a realized approach to reducing semantic conflicts between multiple views. In paper B, we provide such an approach that is able to both detect and resolve potentially undesirable semantic relations between views by making targeted changes to individual views.

2.3 Layout Strategies

With improvements in technologies and computing power, multi-view visualizations are becoming ubiquitous. A multi-view visualization may be a simple visualization dashboard, or several views scattered across multiple windows, screens, or even separate heterogeneous devices. For all these use cases, the multi-view visualization must have a *layout strategy*. This strategy has a high impact on the aesthetic qualities of the design, and emphasis on certain parts of data.

Simple strategies such as juxtaposition [101], where elements are simply placed side by side do not always suffice when there are multiple heterogeneous chart types or devices. For such cases, layouts are often specified manually. This manual layout specification process was simplified by Feiner [31] who proposed the use of a grid. To this day, using an underlying grid to position views is widely used in many areas. Automatic generation of layouts for heterogeneous views is a difficult problem. Lok and Feiner [62] investigated several techniques for automatically generating such information presentation layouts. They categorize layout techniques as either constraint satisfaction or machine learning techniques. While there are layout strategies for single visualizations, we will focus on only layout strategies for multi-view visualizations.

2.3.1 Content-Aware Strategies

Multi-view visualization layouts typically position views with regards to their bounding geometries. This is most commonly found in grid layouts used in most applications such as Tableau [97]. However, some approaches attempt to position views according to their contents, rather than bounding boxes. The technique dubbed content-aware layout by Ishak and Feiner [44] aims to position several windows efficiently according to their contents. Steinberger et al. [96] proposed a dynamic window management technique that positions windows based on coherency between the information shown. Similarly, Haraty et al. [38] optimizes windowing layouts, but with a genetic algorithm that considers the specific task at hand. Zheng et al. [117] proposed an approach to generate high quality, content-aware magazine designs with a deep learning generative model trained on a large magazine layout dataset. While these approaches offer means to position elements by contents, they do not explore such content-aware layouts applied to small-scale information visualization dashboards. In paper C, we address this problem with an approach that transforms a dashboard with an underlying grid layout into a dashboard with a content-driven layout that positions elements by their contents, rather than proxy geometries.

A layout that positions elements with regards to their contents will often have less superfluous white space than bounding geometry aware layouts. The problem of making a layout that positions elements according to contents can be seen as a problem of white space minimization. This problem was explored in a different context than multi-view visualizations. In order to reduce loss of physical fabric while cutting, Albano and Sapuppo [4] explored heuristic methods for positioning irregular 2D shapes with a minimal amount of white space. Bouganis and Shanahan [15] used computer vision techniques to minimize white space in layouts with varying shapes, on both regular and irregular surfaces. The two latter approaches focus on optimal packing, but do not provide means for controlling the final layout and the positions of elements in relation

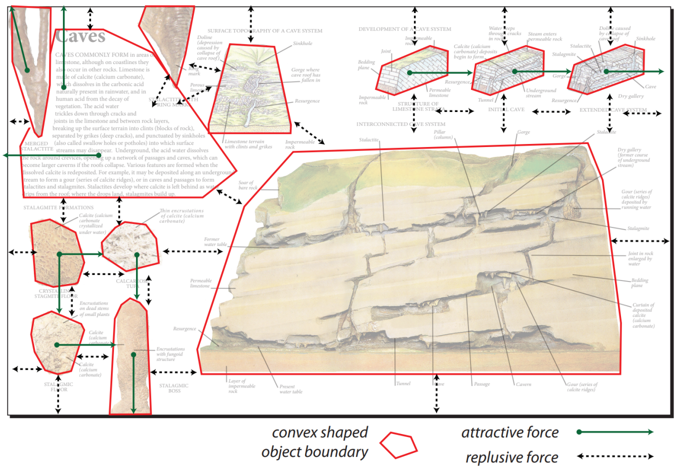


Figure 2.8: Here we see how an illustration has been compacted by its content with the force-directed adaptive approach by Ali et al. [5], where each element is modeled by convex bounding geometry.

to each other. In paper C, we allow for such control of element placements by letting the designer first specify a grid layout which implicitly contains a desired ordering of elements. This grid layout is then automatically transformed into a content-driven layout that respects the original arrangements.

Content-aware constraint based layouts have been proposed in several works. These layouts typically simulate all shapes as physical objects colliding with each other. In other words, the common force-directed layout approach [33] is a well-suited candidate for running such a simulation. This layout is typically used in conjunction with other constraints to achieve a desired result. While the most common application of a force-directed layout is graph drawing, it is not surprising that it has a much wider range of applications, such as visualizing biological pathways [34], improving Euler diagrams [69], rendering Lombardi-style graphs [21], targeting network spatialization [46], and rapidly visualizing large networks [16]. Dengler et al. [25] utilized a simple force-directed layout to generate diagram layouts satisfying both geometric and aesthetic/perceptual constraints. Furthermore, Ali et al. [5] utilized a force-directed layout to generate a content-aware layout of illustrations, by modeling each element by its convex hull. While modeling each element as a convex hull is convenient for a force-directed layout, it is also imprecise and unreliable for more irregular shapes, especially with concave regions. In paper C, we address this shortcoming by using an image-based collision detection and avoidance scheme in a force-directed simulation, enabling collision handling between even highly irregular shapes.

2.3.2 Automatic and Semi-Automatic Strategies

Other strategies for layouts typically apply higher-level aspects modeled as constraints to generate a desirable layout. Such higher-level constraints can be used to automatically achieve greater levels of aesthetics and refinement in layouts. These higher-level aspects represent high-level conceptual information about an overall design. For example, Jahanian et al. [48] quantified concepts from arts and aesthetics into a system for automatically generating magazine cover designs. These quantified concepts were leveraged further to create a recommendation system [47] adhering to intuitive higher-level cues such as "formal" or "sporty". Yang et al. [115] proposed a system to automatically generate layouts by leveraging expert-designed, topic-dependent templates, and a computational framework for integrating and harmonizing high-level aesthetics with low-level image features. Such constraints can also be modeled to be used within the context of machine learning, as done by Li et al. [59] who provided LayoutGAN, a generative adversarial network that allows for synthesis, modeling, and editing of geometric relations between 2D elements. In recent times, the use of sketching gestures in interactive design has increased. Xu et al. [114] proposed an interface for beautifying layouts by making relationships visible and editable through sketching gestures. Following a similar line of thought, Sketchplorer [100] allows for sketch-based design where each sketching gesture is automatically translated into potential local and global improvements.

Due to people using a myriad of different screen sizes and devices, it is in many cases expected that visualizations can automatically adapt their size and layout to fit the target media, i.e., be responsive. Consequently, there is a body of work aiming to make visualizations responsive. There are many different strategies to achieve this responsiveness. Kim et al. [51] analyzed 378 pairs of large and small screen visualizations and proposed a characterization of their resizing strategies. This characterization is based on observed changes in single elements upon changes to the overall design. Andrews and Smrdel [6] leveraged principles of responsive web design to make visualizations responsive. Jacobs [45] proposed an approach for adapting grid-based magazine layouts to different screen sizes. Schrier et al. [89] presented a system for assembling documents from different sources into a size-responsive grid-based magazine design. Dayama et al. [24] proposed means for conveniently and interactively transferring the layout of one user interface to another.

Such automation of layout strategies is highly useful towards making more intricate and aesthetic layouts available in visualization design environments. Existing research in this field often focus on generating a good layout with certain characteristics from scratch. However, the focus is rarely on improving existing designs by transforming and improving an existing layout. In paper C, we enable this by providing an approach for transforming the widely used, well-known grid layout into a more compact content-driven layout with a similar placement of elements.

2

«If we hit that bulls eye, the rest of the dominoes should fall like a house of cards. Checkmate.»
Zapp Brannigan

Chapter 3

Contributions

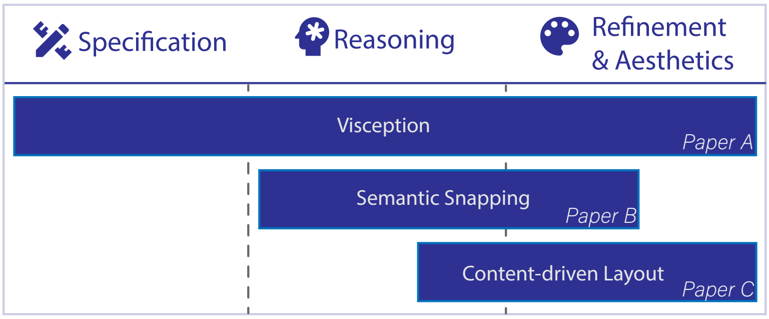


Figure 3.1: Overview of how we contribute to different aspects of multi-view visualization design. The focus of this thesis starts with specification and reasoning, before progressing towards refinement and aesthetics. Our work consists of approaches that have both theoretical and implemented contributions.

This thesis presents approaches for designing multi-view visualizations, where each individual approach contributes towards at least one of the following aspects of multi-view visualization design: specification, reasoning, as well as aesthetics and refinement as shown in Figure 3.1.

Our first contribution, namely Visception, focuses on all mentioned aspects of multi-view visualization design, with a main focus on the specification of nested visualizations for non-experts. Based on our novel underlying Visception Tree data structure, we present its implications and an accompanying visual builder. We then shift the focus towards reasoning and refinement as we introduce semantic snapping, where the aim is to guide non-expert users towards unambiguous multi-view visualization designs by algebraically inferring and resolving potential design problems. Finally, we introduce an approach that helps non-designer users achieve aesthetic, seemingly "hand-crafted" multi-view visualization layouts by transforming an existing grid layout into a content-driven force-directed layout.

In the remainder of this chapter, we outline the main achievements of this thesis in three sections, covering our approaches for making expert multi-view visualization design more accessible to non-experts.

3.1 Nested Visualization Design for Non-Experts

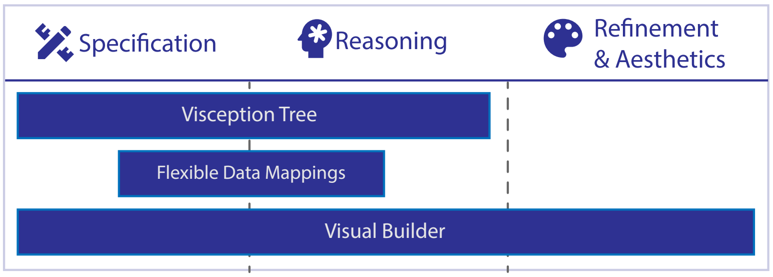


Figure 3.2: Here we see a more detailed overview of how our first work on nested visualization design addresses different aspects of multi-view visualization design.

In this first contribution, we address the full scope of this thesis with a framework and accompanying visual builder for specifying nested visualizations as illustrated by Figure 3.2. The Visception Tree data structure provides means for specifying and reasoning about nested visualizations. Our flexible data mappings makes a visualization less dependent on specific data arrangements by including data wrangling operations as part of a visualization specification. Finally, our visual builder incorporates the underlying specification and reasoning mechanisms into an interactive environment in which users can create and refine high-fidelity aesthetic nested visualizations.

In Figure 3.3 we see the "At the National Conventions, the Words They Used" visualization created by Mike Bostock in 2012 [70], using the D3 [14] library. If we look closer and consider the dataset, which contains the number of mentions per word per political party, we see that it consists of one circle per word, sized according to the

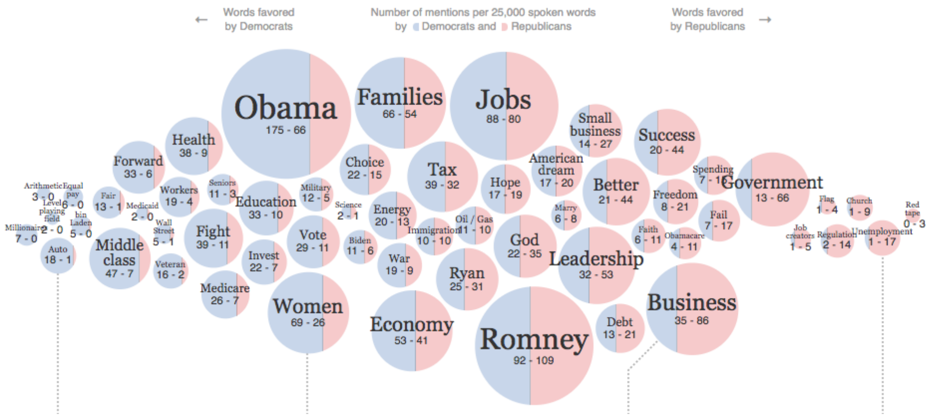


Figure 3.3: Here we see a visualization created by Mike Bostock in 2012 for The New York Times [70]. It consists of circles – one per word, in a force-directed layout. The horizontal positions of the circles are determined by how "democratic" the corresponding word is. Within each circle, there are two squares, clipped by their surrounding circle, with their widths proportional to the number of mentions of the word.

total number of mentions. Each circle is positioned and subdivided according to how much its corresponding word is mentioned by democrats (blue) and republicans (red). Most people would not be able to create this visualization by their own effort. Even for experts, manual coding efforts with libraries such as D3 [14] would be necessary. If the user is not able to code with such libraries, other options are necessary.

A user could instead try an existing visualization grammar to specify this chart. However, these grammars are typically reserved for only specific kinds of charts, or only conventional charts. Therefore, finding the correct grammar to express this can be challenging, especially for non-expert users. Furthermore, to the best of our knowledge, highly customized charts with the embeddings seen in Figure 3.3 are not expressible within existing grammars. Some grammars do utilize nesting, but are locked to specific kinds of visualizations. For example, ATOM [76] leverages nested layouts to express unit visualizations, while Product Plots [107] allow for composing layouts with a limited set of primitives. Furthermore, HIVE [95] allows for nesting different layouts within one another, with a main focus on composing space-filling rectangular layouts for incremental exploration of large multivariate datasets. However, these approaches all require a certain degree of coding expertise. Thus, a user may need a solution that does not require coding or visualization expertise.

Software programs for interactively expressing visualizations, namely visual authoring tools are highly useful for users without the design expertise and coding skills required to express charts with a visualization library or grammar. Such visual authoring tools provide support for generating visual elements that correspond to data, as well as capabilities for flexible configuration of several aspects of a visualization. Within the field of information visualization, several visual authoring tools have been proposed. However, all visual authoring tools have limits to their flexibility and expressiveness, and often introduce their own set of concepts to be learned and utilized by the user. For non-expert users, such concepts are more difficult to learn. Visual authoring tools such as Data Illustrator [61], Charticulator [82], and Lyra [85] enable the specification of a wide range of visualizations, but still fail to express nested charts such as the chart in Figure 3.3, or the unit visualizations produced by the ATOM [76] grammar.

We noticed the following gap in the current body of work: there were no visual frameworks, grammars, or authoring tools able to flexibly express nested visualizations on a general level. To fill this research gap, we set out to create a framework and a visual builder that would enable non-expert users to express such complex visualizations by re-using the same concept, namely nesting. Thus, the first contribution of this thesis is a framework for expressing such nested visualizations. We made our framework accessible to non-expert users by providing an accompanying visual builder, and demonstrated its utility with examples created with our visual builder.

3.1.1 Nesting as a First-Class Operation

Before explaining the underpinnings of our contribution, we believe it is worthwhile to first consider what is possible with nesting as a first-class operator. We begin by reverse-engineering the visualization in Figure 3.3 into a nested visualization. First, we see that each circle has an embedding within itself. This inner embedding always contains two clipped rectangles, where the width ratio between them corresponds to the ratio between the number mentions by democrats and republicans. The first step is to

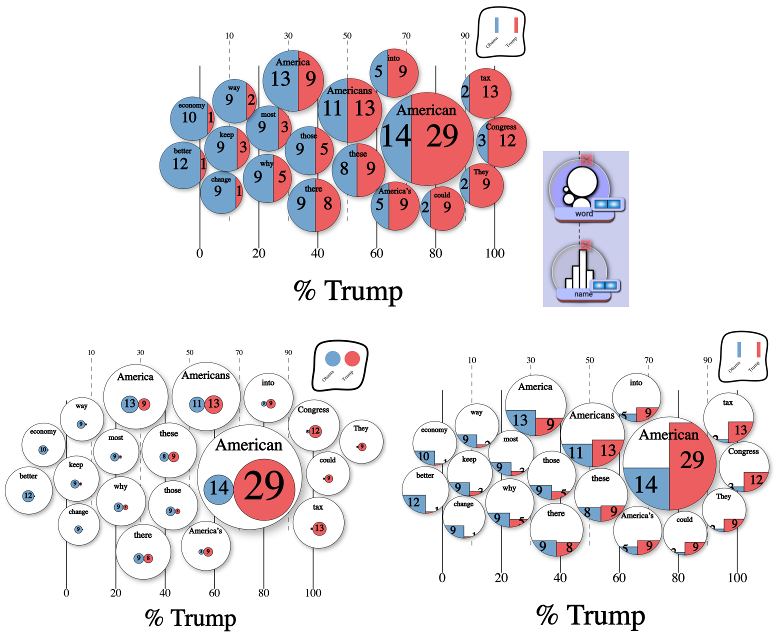


Figure 3.4: This figure shows a modern recreation, and alternate versions of "At the National Conventions, the Words They Used" [70], created with our visual builder.

create this as a nested visualization. First, we create one circle per word, sized by the total number of *mentions*, positioned along the x-axis by how "democratic" it is, i.e., number of mentions by democrats divided by total number of mentions. Furthermore, these circles are positioned according to a force-directed layout to avoid overlapping elements. The next step is to create two rectangles within each circle – one per political party. In other words, a bar chart grouped by the attribute *party*. These bars are then made width-proportional by mapping their widths to the number of *mentions*. From this reverse-engineering process, we can see that this chart can be expressed by nesting a bar chart within a plot of circles, followed by individual editing of mappings. Such use of nesting as a first-class operator yields highly expressive and flexible chart definitions. For example, in Figure 3.4 we see a modern recreation, and two alternate versions that are generated only by editing the mappings of one chart in the hierarchy.

3.1.2 The Visception Tree

In order to realize nesting as a first-class operator, we proposed the Visception Tree data structure (shorthand: VC-tree). This structure encapsulates a hierarchy of charts, which provides precise control over data mappings at arbitrary hierarchical levels, and implicit handling of nesting and deformation behaviors. Thus, the VC-tree enables for concise expression of arbitrary hierarchies of charts, which can then be rendered as corresponding nested visualizations. The most basic element of a VC-tree is a VC-node (Visception Node), which encapsulates a chart with a set of VC-channels that control style and layout parameters. For example, if we consider the circles of Figure 3.3, we see that the chart type is a *plot* of circles grouped by *word* since each circle represents a word. Furthermore, when we consider the inner elements of each circle, we see that they can be expressed as a *bar chart* grouped by *party*, where the *fill color* VC-channel is also mapped to *party*, and the *bar width* VC-channel is mapped to the number of mentions. We divide VC-channels into style channels shared across all charts, and layout channels which affect the layout of specific charts or families of charts. Some style channels within our framework are shown in Figure 3.6. Furthermore, a selection of charts and accompanying layout channels realized within our framework can be seen in Figure 3.5.

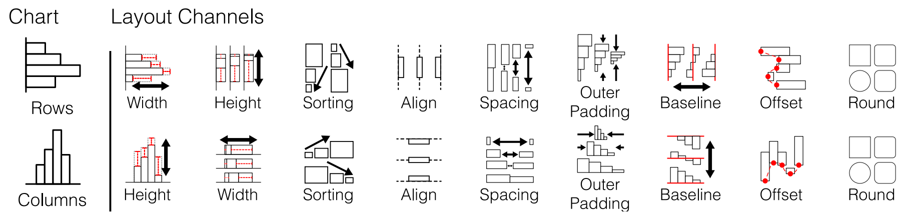


Figure 3.5: Here we see a selection of charts and accompanying layout channels within our framework. A more comprehensive overview of all VC-channels can be seen in paper A.

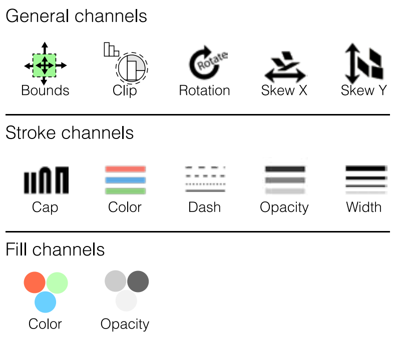


Figure 3.6: This figure shows a selection of style channels within our framework, controlling the stroke, and fill configurations of charts.

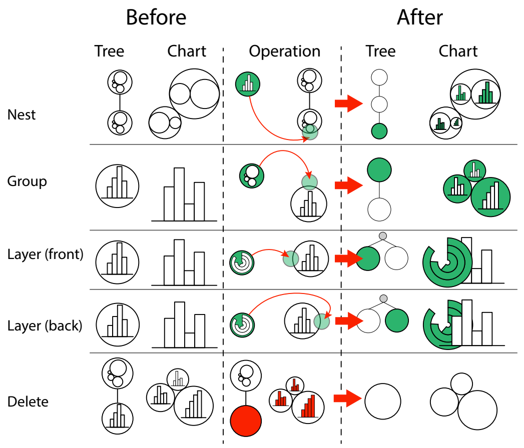
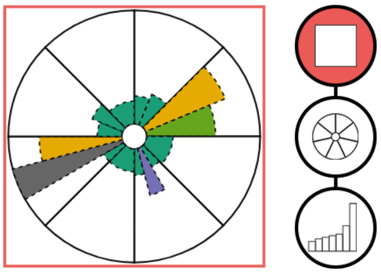


Figure 3.7: This figure displays all operations to modify a VC-tree.

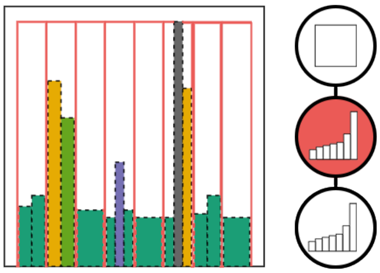
Visception Tree Specification: Our proposed VC-tree data structure makes it possible to specify a nested visualization. In order to make it editable and suitable for non-expert interaction, it is necessary to introduce means for modifying the hierarchical structure of the tree itself. We introduced five operations to edit the hierarchy, which are all illustrated and exemplified in Figure 3.7. A central underlying idea of this structure is that the hierarchy corresponds to the rendered chart. We define that a chart B is nested within chart A, if A is the parent of B in the corresponding VC-tree. Thus, the act of adding a child to a VC-node corresponds to the *nest* operation as seen in

Figure 3.7. Conversely, if a VC-node A is added as a parent of an existing VC-node B, that corresponds to *grouping* B by A as shown in Figure 3.7. Furthermore, the left-to-right order of VC-nodes in the VC-tree correspond to the front-to-back order of the charts, meaning that the leftmost VC-nodes are always layered on top of their rightward neighbors. This rendering order is again illustrated by the *layer* operation shown in Figure 3.7. It is also possible to *delete* a VC-node from the hierarchy, which removes the corresponding chart and its children from the visualization, as depicted in Figure 3.7.

Visception Tree Rendering: The rendering of the VC-tree as a nested visualization is done by rendering the VC-tree hierarchy in a top-down fashion. In other words, the process always starts with rendering the root VC-node such as the highlighted VC-node in Figure 3.8a. We then render its descendants recursively in a top-down fashion until all VC-nodes are rendered. When a VC-node is rendered, it always has one *parent space* per graphical shape of the chart it is nested within. For example, if we consider the bar chart in the hierarchy seen in Figure 3.8a, its parent spaces are the eight arcs of the pie chart corresponding to its parent VC-node. Since the rendering of all charts is done on a purely geometric level, some charts are specified to be deformable (for example, the bar charts seen in Figure 3.8a and 3.9), while others are fitted as the pie charts in Figure 3.9. This geometric handling of nesting behavior ensures that our VC-tree data structure is not limited in terms of depth of nesting, but only in terms of what is practically possible to render on a computer.



(a) Since the immediate parent space of the *columns* chart is an arc, the bars are deformed into arcs that fit within the parent arcs.



(b) As the parent's space is Cartesian, the bars are fit into the Cartesian coordinate system of the parent marks.

Figure 3.8: Two examples of nesting with different types of parent spaces. If the parent space is deformed, each bar of a *columns* chart is also deformed as seen on the left.

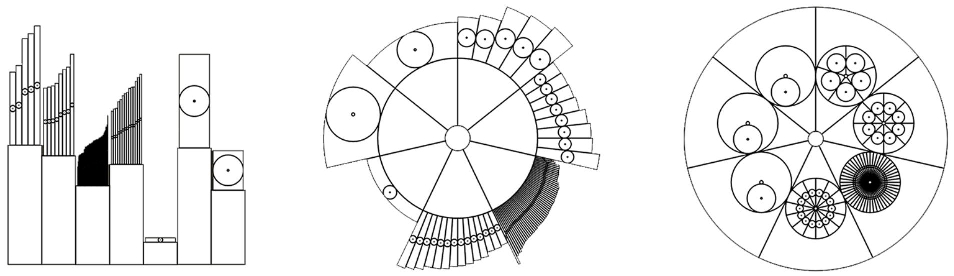


Figure 3.9: Here we see another example of how deformation and nesting works within the Visception framework. Notice how all the pie charts are not deformed, but fitted, i.e., they preserve their shapes but are positioned to fit within the shapes. On the other hand, bar charts are deformed by their parent shapes as seen in the chart in the middle.

3.1.3 Flexible Data Mappings

Here, we detail four approaches for flexibly adapting visualizations to data, as opposed to adapting and rearranging data to fit the visualization environment. There are many opportunities and challenges when it comes to mapping data to visual elements, some of which we address with flexible data mapping mechanisms within our framework. Visualization environments are typically centered around the creation of visualizations, and assume that the data format is as expected. Data wrangling, i.e., the steps involved in making data ready for analysis has long been an elephant in the room of data analysis, tediously consuming extraordinary amounts of time [50].

In our framework, we address data mappings from four different perspectives: (1) We provide more mapping opportunities in the context of nesting by expanding the definition of what constitutes a data-mappable channel, and (2) enable for rapidly changing the visual representation while preserving data shown with our proposed set of VC-channel equivalencies. Furthermore, we (3) make it possible to specify charts less dependent on a specific tabular arrangement by providing four kinds of different data mappings to group charts by. Finally, we (4) facilitate handling of missing data in the context of nesting, by enabling for specification of mapping sparseness.

Using the Parent Datum: Within the context of nested visualizations, chart attributes which are typically considered as "global" become mappable to the parent datum. A typically "global" attribute can be mapped to data dimensions its ancestors are grouped by. This makes it possible to achieve new mappings which are typically not considered for non-nested visualizations, leading to greater design flexibility. Examples of this include using a categorical dimension to enable or disable effects such as drop shadows, or adjusting the drop shadow color of a chart based on its parent datum as seen in Figure 3.10

VC-channel Equivalencies: There are many different ways to show the same data, and it is not always easy to know beforehand which representation is the best. It is therefore ideal to be able to rapidly toggle between different designs. To enable this, we establish a set of VC-channel equivalencies to ensure that mappings are preserved upon changing of the chart type. To illustrate this with an example, consider the two lower charts of Figure 3.4. The only difference between these two charts is the chart

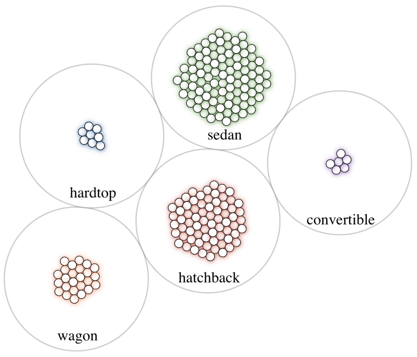


Figure 3.10: Here we see a nested chart, with circles grouped by *body-style*, and then again one circle per row in the dataset. With this grouping, it is therefore also possible to map *body-style* to the drop-shadow color of the inner circles.

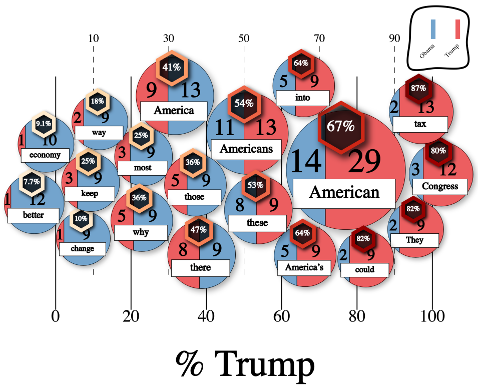


Figure 3.11: This figure illustrates an alternate design of “At the National Conventions, the Words They Used”, and the utility of the *one* data mapping, where it is used to nest a proxy element with a hexagon shape within each circle, and nest a rectangle which is used as a label.

type of the inner elements, which is circles to the left, and a bar chart to the right. It is therefore possible to go from one representation to the other by only changing the chart type.

Arrangement-Agnostic Mappings: Most visual authoring tools and visualization libraries are tied to a specific kind of tabular data arrangement. Typically, examples shown are created from cleaned datasets, arranged in the way the authoring tool or visualization library expects it. However, datasets found “in the wild” are often not arranged in this manner. For example, consider how a dataset including the three dimensions: *income(0-20), income(21-40), income(40+)*, could instead have the following two dimensions: *income, age group*. Within the context of nested visualizations, it might be desirable to have a certain kind of grouping at one level, and a different grouping below. For example, consider a streamgraph showing some tabular data. Here, it is possible to either create one stream per distinct value of a dimension, or for a selection of dimensions, create one stream per dimension. In response to these challenges, we introduce four different kinds of data mappings by which a chart can be grouped: *dimension*, *all*, *identity*, and *monolith*. The *dimension* data mapping corresponds to the SQL command **GROUP BY**, and thus produces one datum for every distinct value of that dimension in the dataset, and amounts to aggregating the data by that dimension, which is very frequently done in both visualization and data analysis. The data mapping *all* produces one datum for every row in the dataset, and is useful when it is desirable to create one graphical mark per row as done with for example unit visualizations [76]. *Identity* simply produces one datum, and is useful especially in the context of nesting, as it allows for creating dummy elements, which can be styled and used to embellish the chart as seen in Figure 3.11. Finally, the *monolith* mapping lets the user specify a selection of dimensions, where one mark is created for each dimension. This mapping is especially useful when the dataset contains multiple numerical dimensions

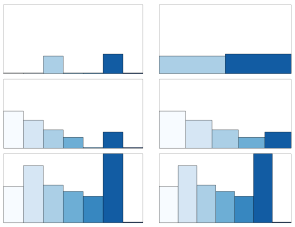


Figure 3.12: This figure illustrates the effects of using a *sparse* and *non-sparse* mappings. To the left, we see a *non-sparse* mapping, which makes the visualization include the missing bars. On the right, these bars are not visible due to the mapping being *sparse*, which makes it difficult to compare across dimensions.

which are to be represented as individual streams in a streamgraph. These mappings allow for flexible specification of nested visualizations with varying data mappings.

Dealing with Missing Nested Data: In the context of nesting, there is a more frequently occurring problem of missing data. When creating a chart grouped by a certain dimension, each graphical mark of that chart represents a certain slice of data. For example, a dataset may be grouped by *gender*, resulting in two graphical marks. However, if there are only males with the hair color *blue*, the hair color *blue* is missing from the female subset of the dataset. Thus, when a dataset is grouped by a dimension, there may be missing data within each subset. It is then an open question whether that missing data should be shown or not. In response to this problem we introduced additional mapping mechanisms to specify whether such missing data should be shown. Thus, we specify that the *all* and *dimension* groupings can be either *sparse* or *non-sparse*. This distinction only makes sense if the chart is nested within another chart. A *non-sparse* data mapping is where all missing values are included. If the data mapping is *sparse*, it excludes missing data items. This is illustrated by Figure 3.12, where the result of a non-sparse mapping is shown to the left, and a sparse mapping is shown to the right. When setting up visualizations such as small multiples, it is especially useful to include missing items to allow for proper comparison as shown on the left in Figure 3.12.

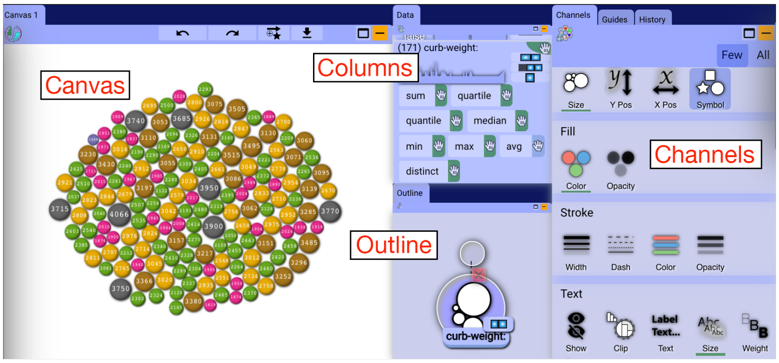


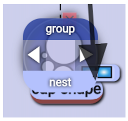
Figure 3.13: A screenshot of all views within Visception. These views enable the user to express and modify arbitrary hierarchies of nested visualizations.

3.1.4 Visual Builder

Having specified the VC-tree data structure and its accompanying flexible data mapping and rendering mechanisms, we set out to make it available as an interactive visual builder. Mapping data to charts or channels is the most basic operation of our framework. Since all drag operations then originate from the *columns view*, we placed it centrally, in direct contact with every other view as seen in Figure 3.13.

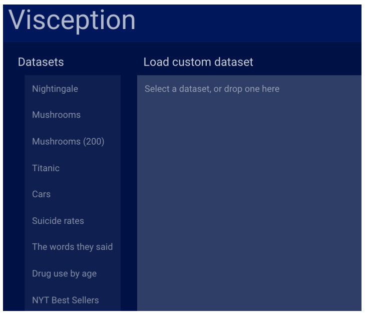
Mapping data to charts and channels is done by dragging data that is a potential chart grouping or channel mapping from the columns view, and dropping it on an eligible target that can properly use the mapping. For example, the *all* mapping can not sensibly be applied to a channel, but it can be applied to a node as a chart grouping. On the top of the columns view we see the *all* and *identity* mappings (see top of columns view in Figure 3.14b), which are only mappable as chart groupings. When dragged, they can only be dropped as chart groupings on the canvas view or outline view. Each column in the columns view can be clicked and expanded to show one tile per aggregation, which can be dragged and accordingly mapped to a channel in the *channels view*. Individual columns expose: the *monolith* mapping if a drag is initialized on the corresponding icon (▣), a *non-sparse* mapping if a drag is initialized on the non-sparse mapping icon (▬▣▣), a *sparse* mapping if a drag is initialized on the column itself, or the sparse mapping icon (▣▣).

Grouping, nesting, and layering can be specified by dropping a data grouping onto the outline view, or by direct interaction with the outline view itself. In the case of modifying the VC-tree structure by dragging of data, a data grouping can be dragged and then dropped on the side of a node in the outline view. A drop on the top of the node results in a *group* operation, which creates a node and sets it as the parent of the target. Our Titanic result demonstrates this operation both hierarchically and visually (Figure 3.16 and 3.17). A drop on the bottom of a node triggers the *nest* operation, which adds a new child node to the parent node. Visually, this corresponds to creating a new nested chart within every graphical mark of the chart receiving the drop. Finally, a drop on the left or right side creates a new node which is a sibling of the target node, where the leftmost nodes are always layered on top of their right neighbors. These operations are illustrated in the figure on the right side of this page.

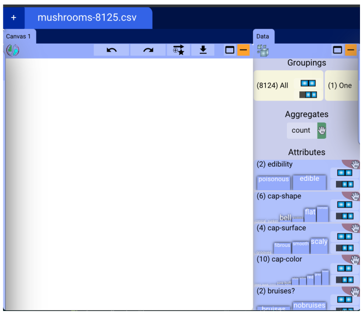


Editing of individual channels is done by clicking on a channel in the channels view, and interacting with the accompanying pop-up widget. For example, if the stroke width channel is clicked, a slider shows up. Another example is the fill color channel, which provides a color picker if the channel is not mapped to data, and a color range picker if the channel is mapped to data.

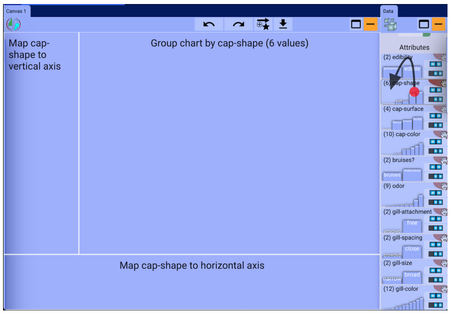
Workflow: The workflow of our builder can be summarized as follows: First, the user selects a custom or pre-defined dataset as shown in Figure 3.14a. Then, only the canvas view and data view are made visible to the user (Figure 3.14b). The initial chart is then created by initiating the drag of a chart grouping as shown in Figure 3.14c. A chart is then created, and the outline view and channels view are made visible as (see Figure 3.14d). The workflow then consists of making incremental edits to the hierarchy in the outline view, or configuring the chart type, or individual channels of a single node, until



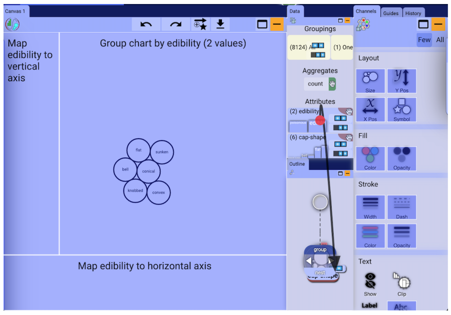
(a) The user can select, or load a template csv dataset.



(b) After loading a dataset, the user can see the data dimensions, and an empty canvas.



(c) When initializing a drag, only the canvas is highlighted as a drop zone.



(d) After creating a chart, the outline and channels view are made visible. When dragging a data dimension, nodes in the outline view and channels in the channels view, are highlighted as drop zones.

Figure 3.14: Visception, getting started step by step.

a desired result is achieved. Examples of this workflow are illustrated in the following section.

3.1.5 Results

Having created the framework and accompanying builder, we set out to generate a set of results to demonstrate the expressiveness of our approach. In this section, we present two selected examples that demonstrate how a relatively simple hierarchy of charts can be used to express an otherwise complicated highly detailed visualization. The charts in these examples are difficult to specify with other approaches, but are concisely expressible by nesting of charts.

Suicide rates: In this example we explore the following dimensions of Kaggle's suicide rate dataset: *country*, *year*, *generation*, *sex*, and *suicides per 100,000*. The goal of this visualization is to show global trends over time across several countries and generations, focusing on the countries with the highest suicide rates. In Figure 3.15, we see a chart with stacked bar charts enclosed within circles. This is achieved by first creating a single chart with one circle per country, and mapping the circle size to the sum of *suicides per 100,000*. Then, we utilize the *nest* operation to create a bar chart within each circle, and size each bar according to the sum of *suicides per 100,000*. Finally, we embed a vertically stacked bar within each bar of the bar chart. These three operations result in the hierarchy displayed by the tree on the lower right of Figure 3.15.

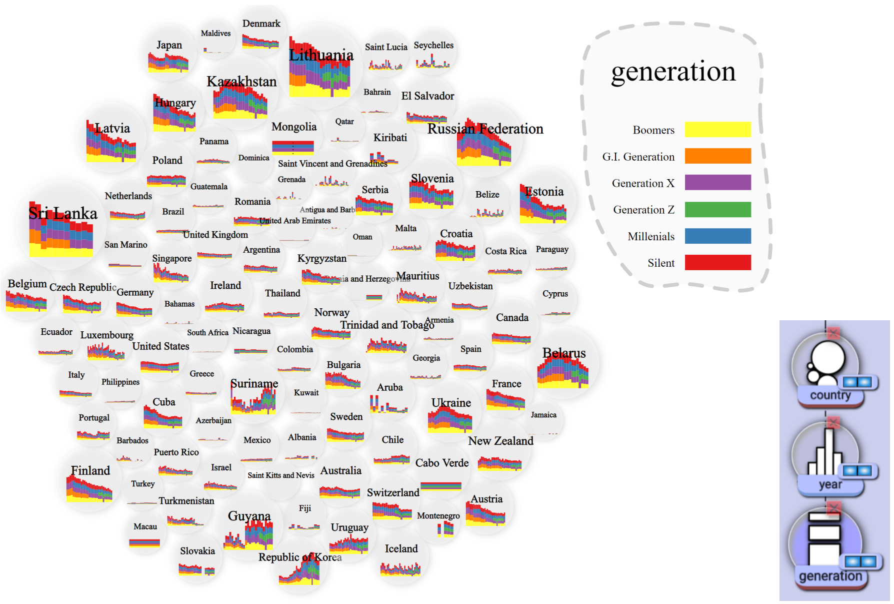


Figure 3.15: Suicides per country over time, by generation, over time.

Titanic: In this example we investigate the survival rates on the Titanic by sex and age group by expressing a unit visualization with the nesting mechanisms of our approach. The dataset contains the following dimensions: *age group*, *sex*, and *survival status*. We leverage the *unit* chart type to show both global patterns and individual details. First, we create a bar chart grouped by *age group*, where every bar is the same height due to no height mapping being specified. Then, within that chart, we nest a *unit* chart, creating one unit per row in the dataset. This unit chart is again sorted by *sex* and *survival status* by applying the corresponding mapping to the *sorting* channel. In order to generate custom shapes within each unit, we nest a *plot* chart within them, and map the shape type and opacity to *survival status*. Finally, we map fill color to *sex*. The final result appears as a “unit stream” and can be seen in Figure 3.16. Next, we wish to look at the results *within* each gender group by separating the “unit stream” into two – one per gender. This is easily achieved with our approach, as we only need to insert a *rows* chart grouped by *sex* into the hierarchy, which is achieved by using the *group* operation on the unit chart, leading to the two separate “unit streams” shown in Figure 3.17.

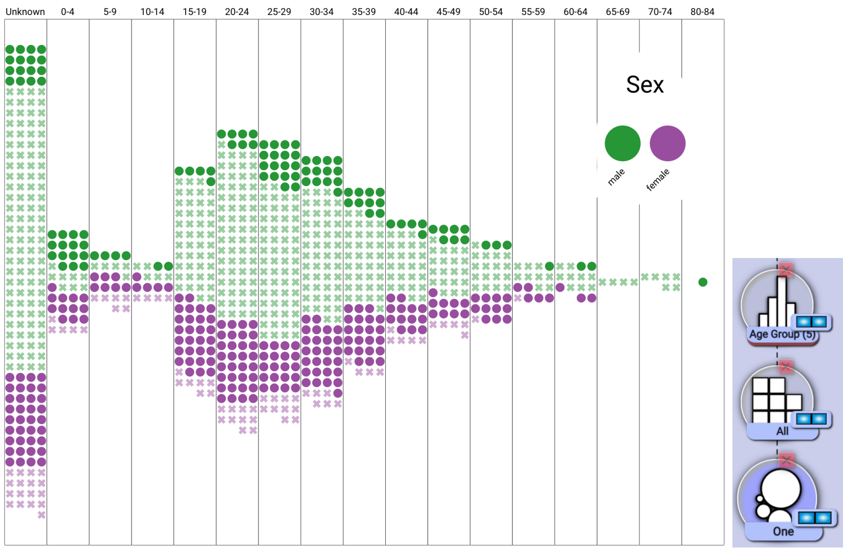


Figure 3.16: A "unit stream" by age group and gender.

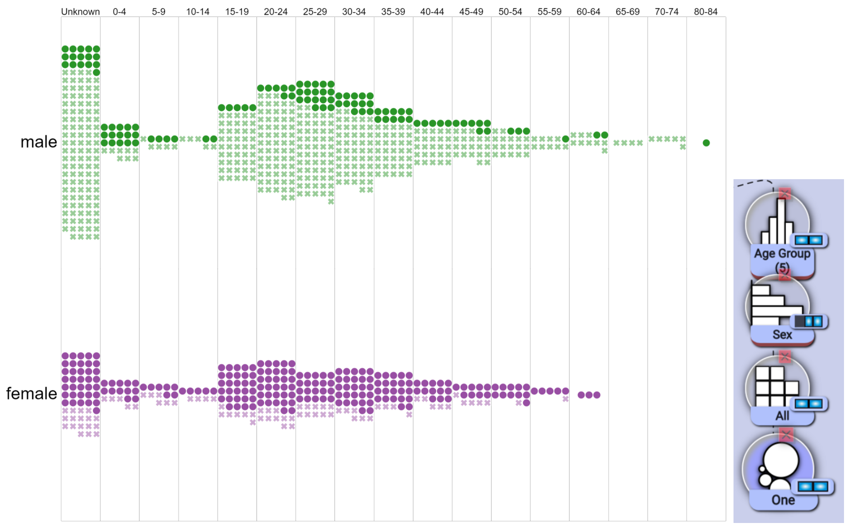


Figure 3.17: Two "unit streams" by gender, then age group.

3.1.6 Discussion and Limitations

The framework of Visception is based on the use of a VC-tree to encapsulate a nested visualization. The top-down nature of our nesting mechanism, where each chart is fitted within the shapes of its parent chart introduces some limitations. More specifically, it does not support layouts where the shapes of the parent charts are influenced by their child charts. As our approach only targeted tabular data, we did not yet explore means to incorporate explicit node-link diagrams. Furthermore, we have not explore means to provide linking between elements and accompanying charts such as parallel coordinates, parallel sets, and matrices. However, we believe our overall program architecture could be molded to support such constructs. The feature these charts have in common is that they have a potentially very large set of data groupings, which is challenging to embed within our architecture and visual builder. For example, a parallel set chart has a potentially infinite set of axes, each with its own grouping and mappings. The intricacies of these different charts were not explored in full in this work.

While we provide flexible data mappings that support different arrangements of tabular data, we did not explore means to integrate these mappings with different data representations such as network and set data.

Not surprisingly, we experienced more performance challenges with more complex hierarchies of data and visual elements. Deep data aggregations on large datasets are costly, and may require an independent server to properly process. Similarly, if the resulting visualization had too many SVG elements, especially with embellishments such as drop shadows, interaction and zooming became increasingly choppy. Although great efforts have gone into optimizing the SVG rendering engines built into most modern browsers, we believe that using GPU-accelerated HTML5 canvas for rendering could remedy some of these performance issues.

3.2 Expert Visualization Design for Non-Experts

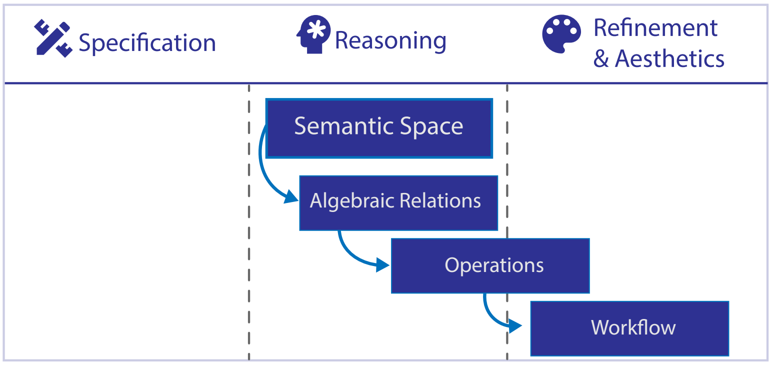


Figure 3.18: Here we see a more detailed overview of how our second work on guided multi-view visualization improvement addresses different aspects of multi-view visualization design. The core idea of our approach was to enable for automatic reasoning and refinement of multi-view visualizations design.

Our second contribution focuses on guided multi-view visualization refinement, addressing the latter two aspects of multi-view visualization design as shown in Figure 3.18. The initial idea of our work was to establish a semantic space in which a multi-view visualization design could be placed. Its position in this space is defined by our algebraic relations, which indicate a certain degree of potential "badness" in a design. As a consequence of these well-defined relations, we also defined operations to remove them, thus improving the design. We made these relations and operations available to users by packing them into a workflow involving only simple user interaction.

Most visual authoring tools focus mainly on ease of use and range of expression. The focus is rarely on guiding the design process according to visualization design knowledge and guidelines to avoid potentially bad design decisions. Such visualization design knowledge and guidelines are often presented on their own, and are seldom packaged into interactive approaches more accessible to non-expert users. Furthermore, such guidelines are rarely known by non-experts who are perhaps most in need of them [30]. An example of such a guided approach is the mechanism of linting applied to visualization, introduced by McNutt [67]. Such automatic guidance has also been explored in the context of multi-view visualizations by Qu and Hullman [79, 80], who found through interviews that users would appreciate a tool to surface warnings in the case of potential design violations. While a single visualization linter is analogous to a spell checker, a multi-view visualization linter that detects problems *between* multiple visualizations is analogous to a grammar checker. To the best of our knowledge, there were no realization of such a visual “grammar checker” that detects design problems in a multi-view visualization, beyond the hypothetical linter discussed by Qu and Hullman [80].

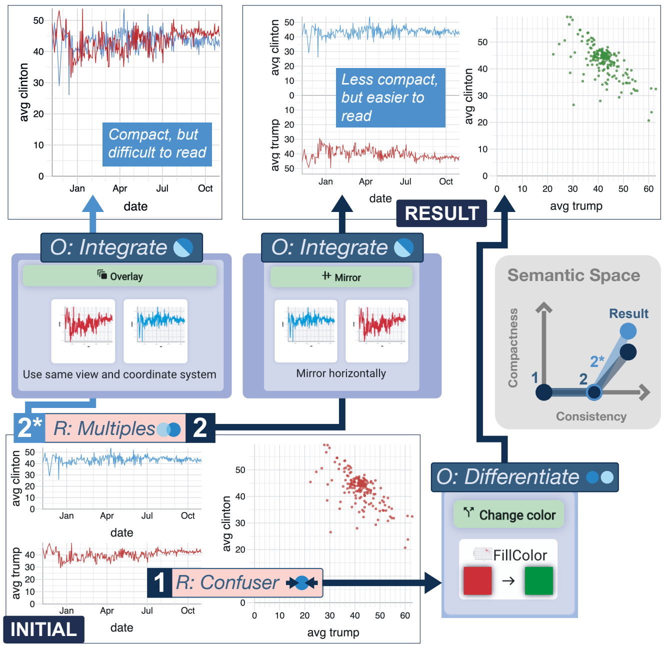


Figure 3.19: Semantic snapping is the process of iteratively improving the compactness and consistency of a multi-view visualization. In this example we see the 2016 US Election poll percentages and pollsters. With our underlying approach, we identify (1) a *confuser* relation since the two views are using the same color to show different data, and (2) a *multiples* relation between the two line charts as they are only different in terms of their y-axis data mapping. These relations are then resolved through corresponding operations of resolving the color conflict between the bottom line chart and right scatter plot, and integrating the line charts into a single multi-line chart, leading to the final design seen on the upper right. The semantic map to the right illustrates the path through the semantic space.

3.2.1 Semantic Snapping

In response to this research gap, we created an approach that helps users make better design choices by informing about potential design conflicts between views, and providing resolutions for violations accessible through a single click. This work is inspired by the above mentioned works, and aims to establish an algebraic model similar to that of Kindlmann and Scheidegger [54], leveraging it to find and resolve potential problems within a multi-view visualization design as demonstrated in Figure 3.19. Our approach can be seen as analogous to *snapping*, which typically helps users *snap* slightly misaligned shapes into perfect alignment. Similarly, our approach of *semantic snapping* corrects semantic misalignments between visualizations. We illustrate this concept in Figure 3.20, where the leftmost bar charts are showing the same quantity, but with different y-axes. These charts are initially not aligned semantically since they are showing the same quantity on the y-axis with different scales. We then *semantically snap* them together by making the y-axes use the same scale, as shown by the top right charts. To enable the user to do this in a guided fashion, it is necessary to define how to detect such potential problems, and how to resolve them.

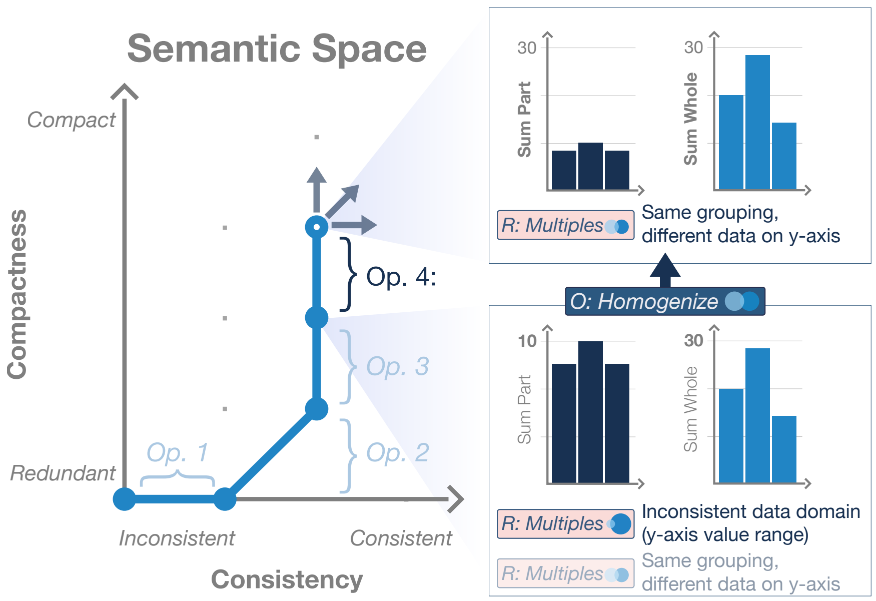
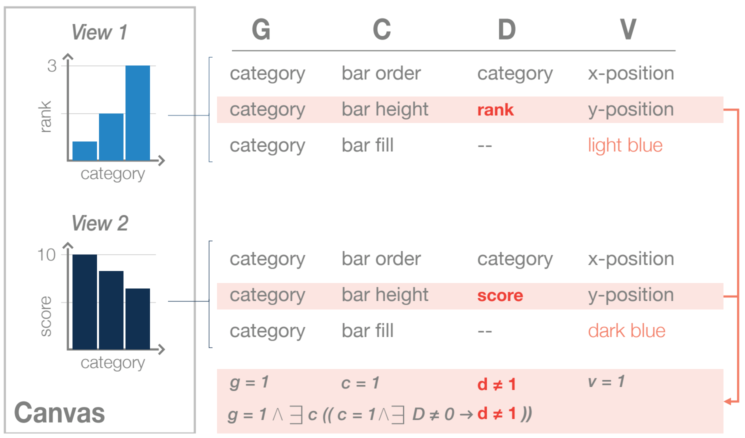


Figure 3.20: Here we see a conceptual overview of our semantic snapping approach. To the left, we see a semantic space in which a multi-view visualization design is placed according to its degree of compactness and consistency. Operation 4 displays the *homogenize* operation made available as a result of a *multiples* relation (the two y-axis scales are different, even though their underlying data represents the same quantity).

3.2.2 Semantic Space

Our approach detects problems by searching for *relations* between views in a multi-view visualization, and resolves them with *operations* that make appropriate modifica-



G	C	D	V
category	bar order	category	x-position
category	bar height	**rank**	y-position
category	bar fill	--	light blue
category	bar order	category	x-position
category	bar height	**score**	y-position
category	bar fill	--	dark blue
$g = 1$	$c = 1$	$d \neq 1$	$v = 1$

$g = 1 \wedge \exists c\, ((c = 1 \wedge \exists D \neq 0 \rightarrow d \neq 1))$

Figure 3.21: Here we see two views and their corresponding (G, C, D, V) tuples for the three channels representing the y-axis (bar height), x-axis (bar order), and fill color. At the bottom, we see the comparison tuple of the two highlighted tuples, which results in these two views being identified as *multiples (1) same grouping* due to satisfying the corresponding predicate on the bottom.

tions to the design. We describe this process in terms of a *semantic space*, in which a multi-view visualization is placed according to its levels of consistency and compactness, which are derived from detected relations, i.e., inconsistencies and redundancies between views. For example, if two views are showing the same data in the same way, there is a *redundancy* relation between them. If we remove one of these views, we also remove the redundancy relation from the design. The removal of this redundancy implicitly moves the design along the compactness axis in the semantic space. Thus, a revision may improve the consistency and/or compactness of the design. We model potential problems as algebraic relations between attributes of individual views. Similarly, solutions to these potential problems are modeled as modifications to attributes of at least one of the views involved in the relation, such that the relation ceases to exist.

3.2.3 Detecting Potential Problems as Relations

The core idea behind our approach was inspired by Qu and Hullman's evaluation constraints [80] and Kindlmann and Scheidegger's algebraic model [54]. In our work, we aimed to make such constraints and rules applicable within the context of multi-view visualizations. Thus, we set out to establish a detection mechanism based on the two following assumptions: (1) problems can be defined as relations between views, and (2) these relations can be identified by comparing mappings across different views.

In order to compare sets of visual encodings across views, we represent each encoding as a tuple (G, C, D, V), where G is the chart's grouping, C is a channel, D is the data shown by the channel, and V is the visual result of the channel's mapping. Each view is consequently represented as a set of such tuples, i.e., one tuple *per channel*. For example, consider Figure 3.21, where we see how each of the charts are associated with three tuples. View 1 in this figure has the three tuples *(category, bar order,*

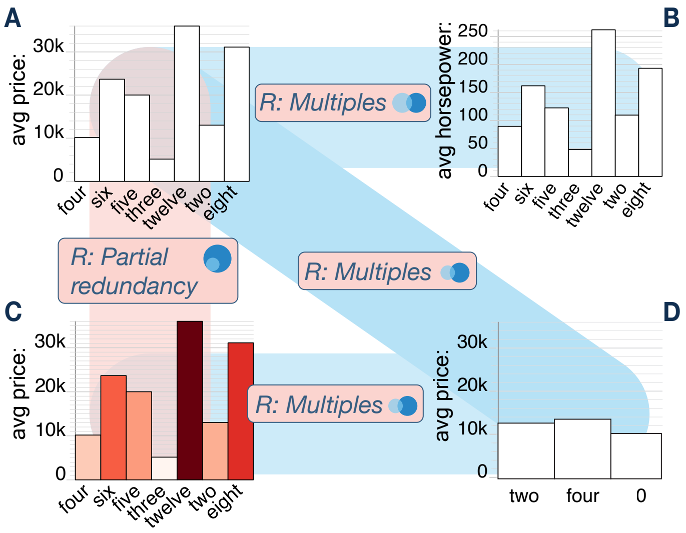


Figure 3.22: Here we see four redundancy relationships: There is a partial redundancy between A and C, since A is the same as C, but without the fill color mapping. (2-3) A and D, as well as C and D are multiples with different groupings, but all showing *avg price* along the y-axis. Furthermore, (4) A and B are multiples with the same grouping, showing different data along the y-axis.

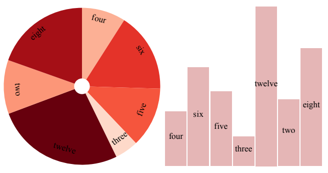
category, x-position), *(category, bar height, rank, y-position)*, and *(category, fill color, 0, light blue)*, where “0” (marked as “--” in the figure) denotes that the mapping is non-existent.

With each single view defined as a tuple, we define relations as comparisons between sets of tuples across multiple views. Each comparison is represented as a *comparison tuple* (g, c, d, v), where each lower case g, c, d, v corresponds to the comparison of the two tuples between the views. Matching values are represented as 1, while mismatching values are represented as 0. To illustrate this, consider the two highlighted tuples in Figure 3.21: (category, bar height, rank, y-position) and (category, bar height, score, y-position). If we compare these two, we see that they have the same values for G, C, and V, but different values for D. Hence, the comparison tuple evaluates to $(g = 1, c = 1, d = 0, v = 1)$.

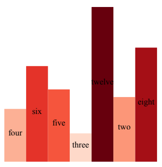
We can now define relations based on the existence or non-existence of a certain comparison tuples across two views. Consider the two highlighted tuples and corresponding comparison tuple on the bottom of Figure 3.21. This comparison tuple $(g = 1, c = 1, d = 0, v = 1)$ satisfies the formula below, which is a specification of the *multiples* relation. These relations inform about the degree of inconsistency and redundancy in a multi-view visualization. The first four relations seen in Table 3.1a-d represent different degrees of redundancy. A *full redundancy* is present if two charts are exactly the same, while a *partial redundancy* is present if all the data shown by one chart, is also shown by another chart which is showing more. Two charts are *multiples* if they have the same grouping, but are showing different data with the same channel, or if they are differently grouped, but are showing the same data with the same channel.

	Relation	Specification	Illustration	Possible Operations	Illustration
(a)	Full redundancy	$g = 1 \land \forall c(c = 1 \rightarrow d = 1)$	D1 D1	Delete one	D1
(b)	Partial redundancy	$g = 1 \land \forall c((c = 1 \land d \neq 1) \rightarrow \exists! D = 0)$	D1 D2 D1∈D2	Integrate, or delete D1 view	D2
(c)	Multiples (1) same grouping	$g = 1 \land \exists c((c = 1 \land \exists D \neq 0) \rightarrow d \neq 1)$	D1 D2	Integrate or homogenize	D1 D2
(d)	Multiples (2) same data	$g \neq 1 \land \exists c((c = 1 \land \exists D \neq 0) \rightarrow d = 1)$	D1 D1	Homogenize	D1 D1
(e)	Hallucinator	$g = 1 \land \exists c(c = 1 \land d = 1 \land \exists D \neq 0 \land v \neq 1)$	←D1→	Homogenize	←D1→
(f)	Confuser	$\exists c(c = 1 \land d \neq 1 \land v = 1)$	D1→ ←D2	Differentiate	D1 D2

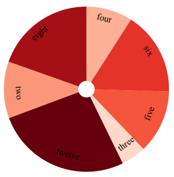
Table 3.1: Here we see all relations and operations specified in terms of our model. Relations a-d represent different degrees of redundancy, while the relations e-f represent respectively when the same data is shown differently, and when different data shown the similarly.



(a) Here we see two charts with a partial redundancy relation. The pie chart on the left is showing all the data shown by the bar chart on the right, in addition to showing data via the fill color channel.



(b) Result of an *integrate* operation wherein the fill color mapping is transferred from the pie chart to the bar chart, before also removing the pie chart.



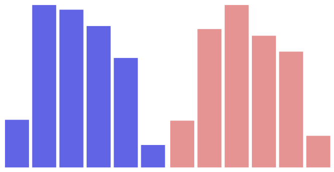
(c) Result of an *integrate* operation where we remove the chart showing the least data, so that only the pie chart remains.

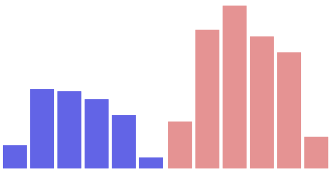
Figure 3.23: This figure shows two ways to *integrate* charts when there is a *partial redundancy* present. This operation ensures that all the same data is still shown, but only once.

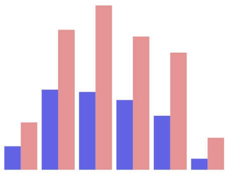
The *hallucinator* relation (Table 3.1e) corresponds to Kindlmann and Scheidegger's hallucinator for single view visualizations, and is present if two charts are showing the same data differently. Conversely, the *confuser* (Table 3.1f) occurs when two charts are showing different data in the same way, and is a multi-view equivalent of Kindlmann and Scheidegger's definition of a confuser.

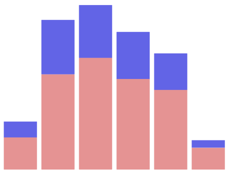
3.2.4 Resolving Potential Problems with Operations

Since our relations describe potential anti-patterns, operations are potential solutions to those anti-patterns. Thus, we proposed the following understandable and predictable operations:
O1: Delete is made available if there is a full or partial redundancy. In other words, if one chart is showing only data that is shown by another chart, one of them can be


(a) Here we see two charts that are similarly grouped multiples.


(b) Result of a homogenize operation which makes the y-axes use the same scale.


(c) Result of an integrate (group) operation, which creates a grouped bar chart.


(d) Result of an integrate (stack) operation, creating a stacked bar chart.

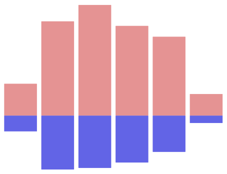
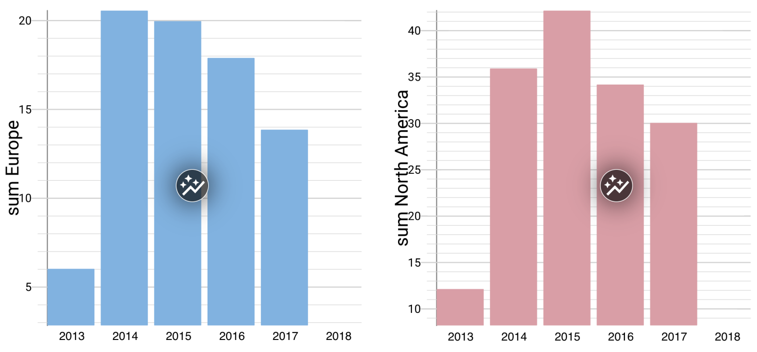

(e) Result of an integrate (mirror) operation, creating a mirrored bar chart.

Figure 3.24: This figure depicts four different operations that can be performed in response to a multiples relation. (a) shows the original two charts, while b-e shows the outcome of performing the operations on the original chart.

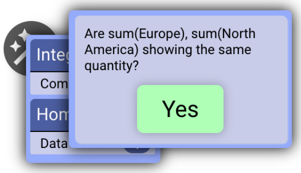
deleted as shown in Table 3.1a and Figure 3.23.
O2: Homogenize makes dissimilar views more similar, and can be used in response to *multiples* (shown in Figure 3.24b-e) and *hallucinator* relations as seen in Table 3.1c-e.
O3: Differentiate makes similar charts more dissimilar, and is useful for resolving *confuser* relations (See Figure 3.1f). For example, if a chart is using the same color scheme to show different data, the *differentiate* operation proposes alternate non-conflicting color schemes to the user.
O4: Integrate can be used to combine several views into a single view, and applies if the views are *multiples*, or in the case of a *partial redundancy*. In the case of multiples with the same grouping (Table 3.1c), there are four ways of integrating the charts: *overlay*, *group*, *stack*, and *mirror*. The operations *group*, *stack*, and *mirror* are exemplified in Figure 3.24c-e, while we can see an example of the *overlay* operation to the left in Figure 3.19. In response to a partial redundancy, where one chart is showing only a subset of data shown by another, two options are available: We can either delete the chart showing the *least* data and keep the other, or transfer mappings such that they are showing the same amount of data, and delete the chart originally showing the most data.

3.2.5 Demonstration and Workflow

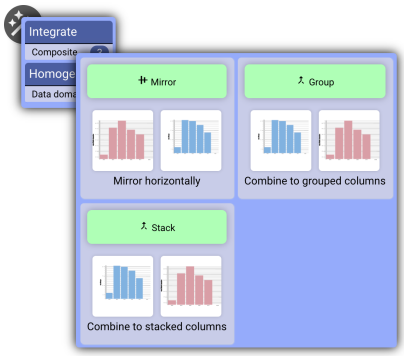
We demonstrated the viability of our approach with several case studies. Our aim was to make non-expert users informed and aware of potential problems, and enabled to resolve them if desired. In Figure 3.25, we (a) see a button on top of each chart, when



(a) Here we see a multi-view visualization consisting of two bar charts, showing sum of video game sales in Europe and North America. The buttons on the middle of each chart expose potential operations when clicked.


Integrate
Composite
Homogenize
Data
Are sum(Europe), sum(North America) showing the same quantity?
Yes

(b) Here, the user is asked if the domains of the two y-axes are showing the same quantity. If the user clicks "Yes", the scales are equalized.



(c) Here we see three potential integrate operations available to the user. If the operation is clicked, it applies the according revision to the design.

Figure 3.25: These figures provide an overview of how the process of semantic snapping is made available to the user through our interface, which consists of a single clickable button attached to each individual view.

this button is clicked, the user is presented with available operations (b-c) as seen in Figure 3.25. Thus, we show an initial design with several problems, and describe how our relations and operations help guide the user towards a refined, more consistent and compact design. In the design environment, all relations and accompanying operations are pre-computed upon any change to the multi-view visualization design.

COVID-19 in Germany

In this case study, we show an initial problematic design of a COVID-19 dashboard showing six charts. The initial layout of Figure 3.26 consists of two bar charts, two pie charts, and two stacked area charts. The two bar charts show the number of deaths, and number of cases by age group, while the two stacked area charts are showing the number of deaths and cases over time. The two pie charts are showing the number of

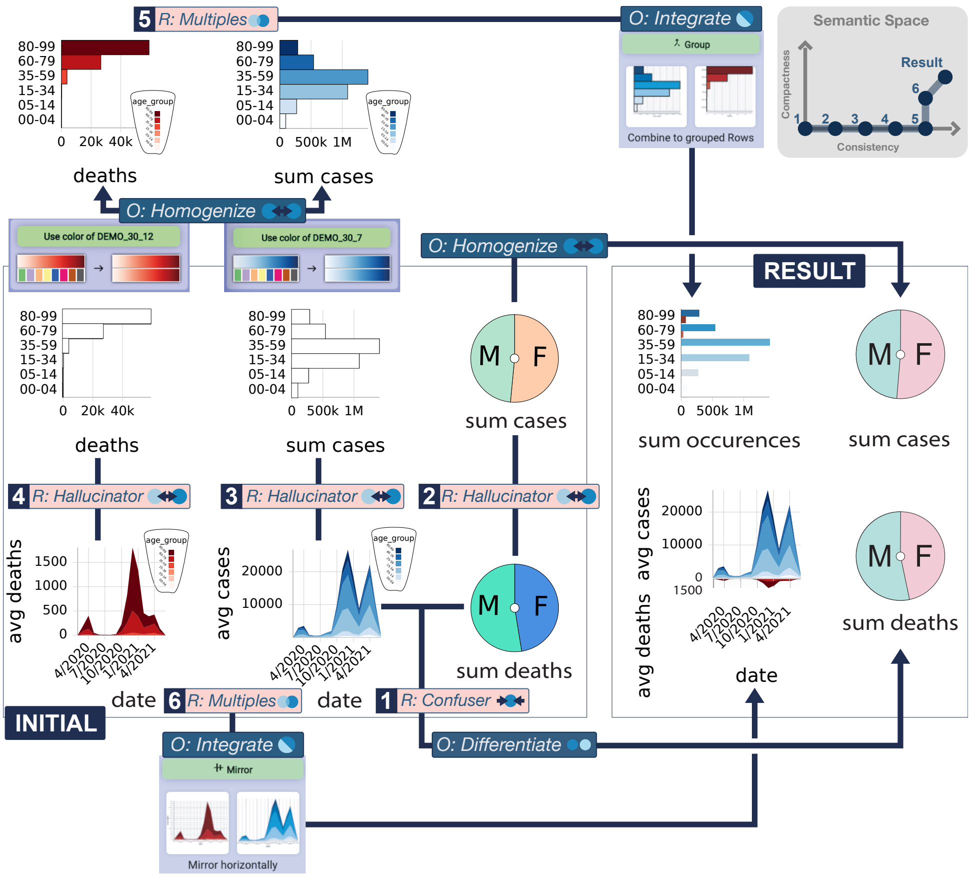


Figure 3.26: Case study workflow demonstrating semantic snapping to resolve a COVID-19 dashboard with a *confuser*, several *hallucinators* and two *multiples* relations. We resolve these relations with a series of operations that include *homogenize*, *differentiate*, and *integrate*.

deaths and cases across genders.

Looking at this initial design, there are several problems which are automatically detected by our system. (1) There is a confuser between the middle stacked area chart and the lower pie chart, since they are showing different data with a similar blue color. (2) There is a hallucinator between the two pie charts, as the charts are color-mapped to gender with two different color schemes, i.e., they are showing the same data differently. (3) There is a hallucinator between the bar chart showing the cases by age group, and the area chart showing the cases over time as they are both grouped by age group, but are using different color schemes. (4) Similarly, there is a hallucinator between the two leftmost charts as they have the same grouping showing the same data, but with different color schemes. (5) The two bar charts are multiples as they have the same grouping, and same y-axis. (6) Similarly, the two area charts are multiples due to their identical groupings by age group, and x-axis mapping to date.

With our approach, the user is able to resolve all these relations with the following operations: First, the pie charts have their colors equalized with the homogenize operation, which also resolves the confuser between the lower pie chart and area chart. Next, the color mappings of the two lower area charts are applied to the two upper bar charts as a consequence of two homogenize operations. Finally, the two bar charts are integrated to a grouped bar chart, similarly to the two stacked area charts which are integrated to a mirrored stacked area chart. The final result can be seen on the right side of Figure 3.26.

3.2.6 Discussion and Limitations

While our approach was built based on the Visception [56] environment, it is possible to use within other environments if a few prerequisites are met. Firstly, these environments must have mechanisms for retrieving information about basic mappings. Since such mechanisms are fundamental for specifying charts, we believe them to be present in most environments. Secondly, the environment must provide means for specifying "combined" charts such as stacked, mirrored, or grouped bar charts. Our approach is therefore dependent on the expressive range of its underlying environment, and is thus also most expressive if the environment supports a wide range of expression.

There are several aspects that we did not address, but that could be incorporated. For example, operating on multiple datasets would be possible, but it would also be more difficult to establish equivalences between data dimensions. The detection of *different* data shown in the same way, and corresponding differentiation of charts would work exactly the same. However, manual user input would be necessary to establish connections across different datasets, and thus infer relations on these similarities. While our current operations have no complicated inputs and require only a single click, we believe that more customizable operations could lead to a greater range of design options. Especially in the currently unaddressed case of layout customization, user inputs specifying which views are important, related, or unrelated would greatly help in computing alternate multi-view visualization layouts.

Finally, our approach does not take the user task into consideration, which would be a great challenge of its own. We believe that this is a topic worth exploring in future work, and that general user tasks could guide our evaluation of relationships and corresponding solutions.

3.3 Artist Designs for Non-Artists

In this work, we contributed an approach for turning rectangle-based grid layouts into aesthetic, artistically "hand-crafted" content-driven layouts. The main focus of our work is refinement and aesthetics as seen in Figure 3.27, backed up by a force-directed layout with forces derived from an original grid layout.

Our previous contribution addressed different ways to refine multi-view visualizations through operations. However, we did not address the *layout* aspect of multi-view visualization design. The most widely used layout in dashboard design is the grid layout, which is based on repeated subdivisions of rectangular spaces. This layout does not only appear within visualization design environments, but also in code editors such as Visual Studio Code, and dashboard software such as Tableau [97] and PowerBI.

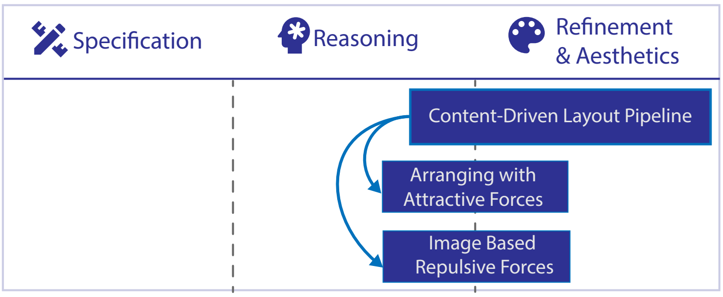

Specification
Reasoning
Refinement & Aesthetics
Content-Driven Layout Pipeline
Arranging with Attractive Forces
Image Based Repulsive Forces

Figure 3.27: Here we see a more detailed overview of how our third work on automatic multi-view visualization layout refinement addresses different aspects of multi-view visualization design. We emulate an original grid layout topology with our attractive forces, while we repulse elements with an image-based repulsive force.

In response to ubiquitous use of grid layouts in the context of information visualization dashboards, we created an approach that transforms such grid layouts into more aesthetic content-driven layouts.

Highly aesthetic layouts typically have a "hand-crafted" quality to them. For example, Minard's visualization of Napoleon's invasion of Russia seen in Figure 3.28 positions elements with regards to their contents, as opposed to their bounding boxes. Such layouts that position elements by their contents may be more compelling and visually pleasing, but also require significant manual fine-tuning. Existing approaches to automate this process are often based on the force-directed layout [33]. Ali et al. [5] adopted this approach in order to position elements according to their contents. However, their work does not properly handle concave regions of highly irregular shapes since it uses convex hulls as proxy geometries for collision detection. To the best of our knowledge, there were no such layout mechanisms that managed collision avoidance between arbitrary shapes used for multi-view visualization layouts.

Based on this research gap, we proposed a mechanism for turning a grid layout into a similarly arranged force-directed layout where elements are positioned according to contents rather than bounding geometries. As an alternative to such bounding geometries, we utilize the Euclidean distance transform [12] of elements to detect and avoid collisions.

3.3.1 Content-Driven Layout Pipeline

The goal of our contribution was to enable users to refine the layout of an existing multi-view visualization. We wanted to automatically make the layout more aesthetic and compact by utilizing previously unused white space. The core idea is to compute a set of attractive and repulsive forces between elements, then apply these forces with a force-directed layout simulation. The adjacency relationships of the original grid layout are modeled by selective application of attractive forces, while the content-to-content repulsion is modeled by combining the distance transforms of overlapping elements.

We attained these goals with a pipeline composed of four steps: (1) We infer adjacency relationships from the underlying topology of the grid layout, and store them in an *adjacency graph*. From this graph, we (2) derive a set of central elements,

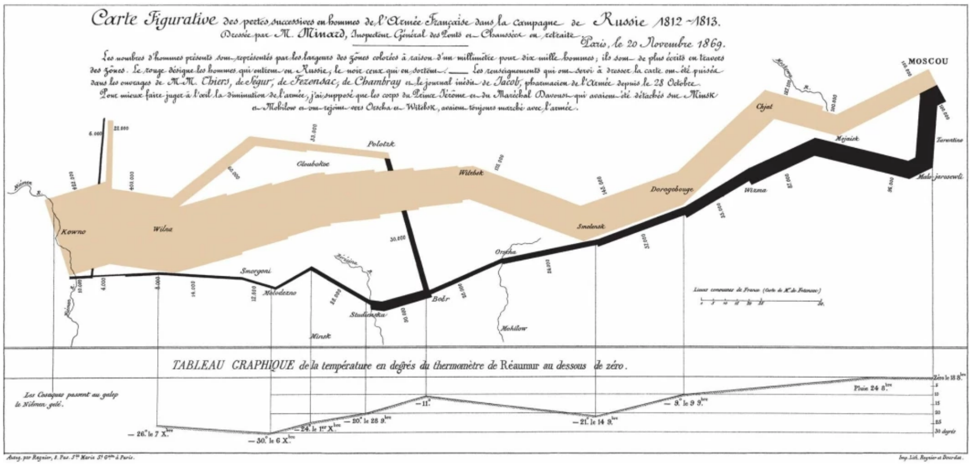


Figure 3.28: Here we see a classic visualization, namely Charles-Joseph-Minard's map of Napoleon's Russian campaign from 1812-1813 [71]. Note how the elements of the layout are positioned not according to bounding rectangles, but with regards to the content. Such positioning is preferred and achievable when hand-drawing, but highly difficult to achieve with automatic computer-generated layouts.

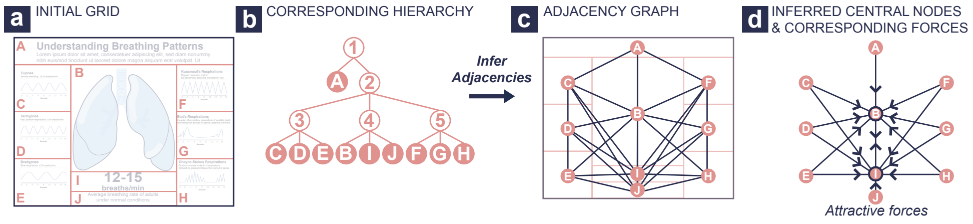

a INITIAL GRID
Understanding Breathing Patterns
12-15 breaths/min
b CORRESPONDING HIERARCHY
Infer Adjacencies
c ADJACENCY GRAPH
d INFERRED CENTRAL NODES & CORRESPONDING FORCES
Attractive forces

Figure 3.29: Here we see the three steps involved in inferring the attractive forces with our approach. First, we model the grid layout (a) as a hierarchy (b), from which we derive an adjacency graph (c), then from this adjacency graph we infer central node and set up corresponding forces (d).

such that we can (3) set attractive forces accordingly. Finally, we (4) set up repulsive forces wherein image-based content-to-content distances are computed during the force-directed simulation.

3.3.2 Arranging with Attractive Forces

Attractive forces ensure that elements gravitate towards each other. The core idea behind these forces is to emulate original arrangements so that there is only attraction between views which are adjacent in the original layout. Furthermore, in order to stabilize the layout by minimizing the total amount of energy in the layout, it is ideal to not apply any redundant attractive forces. To realize this idea, we derive the attractive forces from an original grid layout in two steps: First, we infer an adjacency graph from the underlying tree structure of the original grid layout. Then we derive a set of central elements to direct the attractive forces towards.

Adjacency Graph Computation

The topology of the original grid layout is closely approximated by our algorithm for inferring an adjacency graph. In this graph, a node represents an element in the layout, while a link represents an adjacency relationship. The algorithm aims to capture direct adjacencies between elements, as well as adjacencies between groups of elements. To illustrate this further, consider Figure 3.29, where the elements B and I are considered to be adjacent due to their adjacent parent groups labeled 3 and 4. Thus, our algorithm respects adjacency relationships between groups, as well as immediate geometric neighborhood relationships between elements. The exact algorithm can be seen in Algorithm 1 in paper C.

Central Nodes Computation

We derive the most central elements in the layout from the already inferred adjacency graph. Intuitively, some elements are visually perceived as the centers of gravity of the layout. For example, we can see that element *B* and *I* in Figure 3.29 are the most central. The main idea of this algorithm is to iteratively pick the most connected node in the adjacency graph as a central node until all non-central nodes are directly connected to at least one central node. These central elements are inferred according to Algorithm 2 in paper C, which does a traversal of the previously inferred adjacency graph.

The procedure repeats until all nodes are connected to at least one of the inferred central nodes. First, all nodes are tagged as unvisited. Then, we repeat the following process until all nodes are either central nodes, or tagged as visited. First, for every node, we compute how many unvisited nodes can be reached from it, and how many *indirectly* unvisited nodes it is connected to. Indirectly unvisited nodes are nodes that are neighbors of one node, and also neighbors of an unvisited node. We then sort the nodes according to the number of unvisited neighbors. If several nodes have the same number of unvisited neighbors, we choose the nodes with the highest number of indirectly unvisited neighbors.

For example, if we traverse the adjacency graph seen in Figure 3.29c, the first node to be detected as central is B. Afterwards, we see that J is the only unvisited neighbor of E, I, and H. But indirectly, I is connected to both E and H, which are both connected to an unvisited node (I). If we look at E and H, we see that they are only connected to one additional node – node I, that can reach J. Thus, I has a higher number of indirectly unvisited nodes, and is thus chosen as the second central element. These central elements are then used to compute a small set of attractive forces which emulate the original grid layout topology.

3.3.3 Image-Based Repulsive Forces

In order to prevent contents from overlapping, we apply a repulsive force that is computed based on the Euclidean distance transform [12] of the elements. The distance transform of an element encodes in every pixel, the Euclidean distance to the nearest content pixel. The distance transform is computed from a binary image representation of the original content, where 1 is content, and 0 is white space. In order to properly repulse elements, we need to be able to compute a content-to-content distance between them, and infer the directionality of the content-to-content repulsion.

Proximity Computation

The core idea behind computing the proximity is that we combine the distance transforms in order to find it. In a single Euclidean distance transform, each pixel encodes the Euclidean distance to the nearest content pixels. Furthermore, all distance transforms are 2D grayscale images.

If two elements are overlapping, we correspondingly sum the overlapping regions of the distance transforms, resulting in an image in which each pixel will encode the distance between the nearest *two* content pixels, i.e., the distance between the nearest pixel to the first element *plus* the distance to the nearest pixel to the other element. Thus, the computation of the Euclidean content-to-content distance between two elements is done as follows: First, we check if the elements have overlapping bounding rectangles. If they do overlap, we clip the intersecting region between these bounding rectangles from the distance transforms of both elements, then sum them together, before we finally pick the smallest value from the summed distance transforms. This smallest value is then the content-to-content distance approximation.

Directionality Computation

The distance transform proved useful to find the proximity between two elements in a content-driven fashion. Similarly, it can inform the direction of repulsion. Recall how each pixel in a distance transform encodes the distance to the nearest content pixel, consequently each “pixel” in the gradient of the distance transform encodes a 2D vector pointing away from content. Thus, when two elements are overlapping, and it is desirable to repulse one element away from another, we use this information to determine that repulsion. We infer the repulsion as follows: First we consider the gradient of the element that is pushing another element away. Then, we clip this gradient by the intersecting region between the bounding rectangles of the elements. All vectors of this clipped region of the gradient are summed together into a single vector, which is then normalized, resulting in the normalized directionality of repulsion.

3.3.4 Case Studies

The utility of our approach is demonstrated with two case studies. The main idea behind these studies is to show an existing grid layout and its corresponding content-driven layout generated by our approach.

Respiration patterns: In this case study, we show how our approach can be used to fulfill a design brief without having to manually place elements in a content-driven fashion. Here, a designer is initially tasked with creating an infographic of respiration patterns, where line charts of 6 breathing patterns are arrayed around an image of lungs and accompanying text as seen on the upper left of Figure 3.30. Normally, this would require manual positioning in Adobe Illustrator or a similar manual design tool. However, with our approach, it is possible to simply specify the topology in a gridded fashion as seen on the upper right of Figure 3.30, and then use our approach to make the

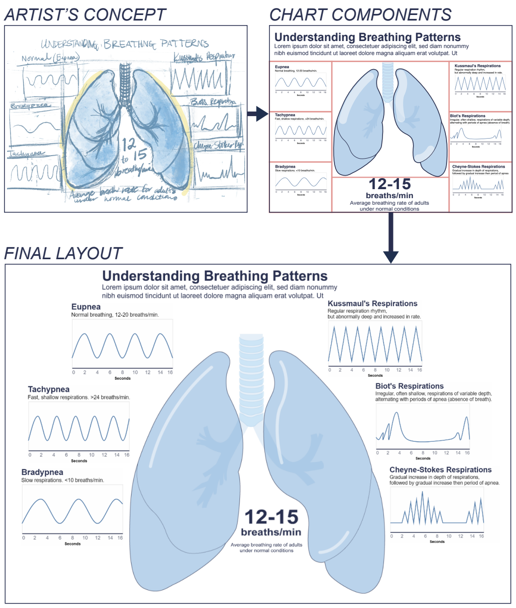

ARTIST'S CONCEPT
CHART COMPONENTS
FINAL LAYOUT
Understanding Breathing Patterns
Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut
Eupnea
Normal breathing, 12-20 breaths/min.
Kussmaul's Respirations
Regular respiration rhythm, but abnormally deep and increased in rate.
Tachypnea
Fast, shallow respirations. >24 breaths/min.
Biot's Respirations
Irregular, often shallow, respirations of variable depth, alternating with periods of apnea (absence of breath).
Bradypnea
Slow respirations. <10 breaths/min.
Cheyne-Stokes Respirations
Gradual increase in depth of respirations, followed by gradual increase then period of apnea.
Seconds
12-15
breaths/min
Average breathing rate of adults under normal conditions

Figure 3.30: With our content-driven layout approach, the artist achieves a composition where the respiratory patterns are arrayed around the lungs.

layout content-driven, resulting in the final layout seen on the bottom of Figure 3.30. Now, if an element changes slightly, the layout will simply update accordingly, rather than requiring manual tweaking in a design tool.

Wind Turbines: Here we show how our approach works with several irregular shapes. The upper left sketch in Figure 3.31 shows the artist's desired visualization. Creating this visualization manually is tedious and requires multiple adjustments. Furthermore, this visualization has several irregular elements: Four wind turbines, a map, as well as a legend with highly varying label sizes. To realize the artist's initial concept, the designer sets up an initial grid layout arrangement that approximates the desired arrangement. The designer then leverages our approach to automatically transform the grid layout into a content-driven layout shown on the bottom of Figure 3.31.

3.3.5 Discussion and Limitations

Our approach targets scenarios where there are relatively few elements, which is often the case for common infographics. This technique is therefore in its current state not well-suited for layouts of large numbers of elements. While we operated solely on static grid layouts as initial layouts, our approach could be used on different kinds of initial layouts, since the only change required would be the inferring of the initial adjacency graph. An example of such a scenario would be a "snapping" feature that assists a user in positioning elements close to each other in a manual design environment such as Adobe Illustrator. While our approach produces promising results, it was shown to be highly dependent on initial viewport dimensions, and would in some cases get stuck in a local minima. Such local minima situations could be resolved by interaction, where

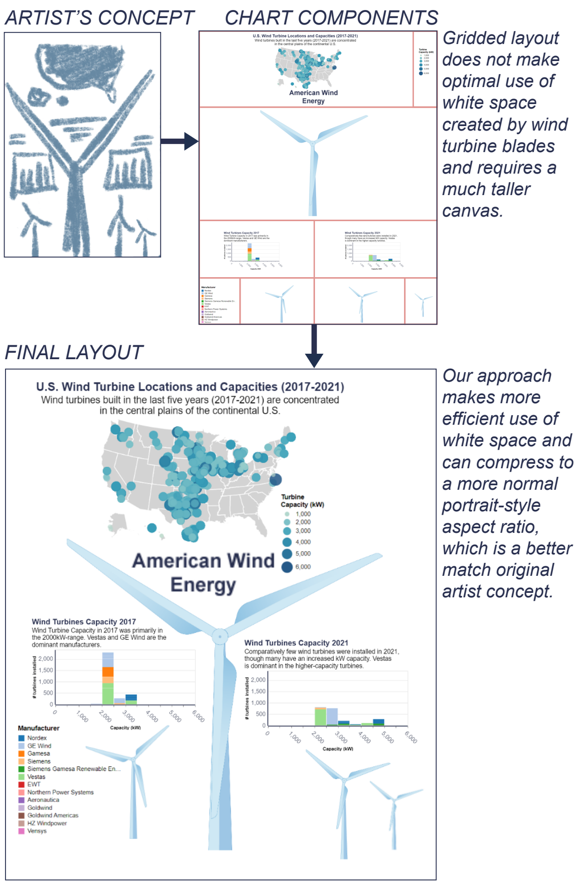

ARTIST'S CONCEPT
CHART COMPONENTS
Gridded layout does not make optimal use of white space created by wind turbine blades and requires a much taller canvas.
FINAL LAYOUT
U.S. Wind Turbine Locations and Capacities (2017-2021)
Wind turbines built in the last five years (2017-2021) are concentrated in the central plains of the continental U.S.
Turbine Capacity (kW)
1,000
2,000
3,000
4,000
5,000
6,000
American Wind Energy
Wind Turbines Capacity 2017
Wind Turbine Capacity in 2017 was primarily in the 2000kW-range. Vestas and GE Wind are the dominant manufacturers.
Wind Turbines Capacity 2021
Comparatively few wind turbines were installed in 2021, though many have an increased kW capacity. Vestas is dominant in the higher-capacity turbines.
# turbines installed
Capacity (kW)
Manufacturer
Nordex
GE Wind
Gamesa
Siemens
Siemens Gamesa Renewable En...
Vestas
EWT
Northern Power Systems
Aeronautica
Goldwind
Goldwind Americas
HZ Windpower
Vensys
Our approach makes more efficient use of white space and can compress to a more normal portrait-style aspect ratio, which is a better match original artist concept.

Figure 3.31: Here, our approach makes the plots and text elements array around the largest central wind turbine, thus optimizing the use of white space and matching the artist's intended layout more closely.

the user can manually drag elements to more ideal positions. Furthermore, we did not explore our approach in the context of dynamic content, such as with dynamic filtering and cross-linking on Vega-Lite specifications. However, we believe such dynamic behavior would be well-suited for the incremental nature of force-directed simulations, where the layout would simply adapt to individual element changes in real-time.

For most shapes, we found that using a smaller distance transform resolution gave an increase in performance with little impact on the final layout. However, for elements with finer details, this downscaling resulted in cases where they were not fully taken into account. We believe this problem could be addressed by making it possible to interactively specify the distance transform resolution, or automatically detecting an ideal resolution to capture finer details on a per-element basis. Our use of the distance transform also opens up for using other image processing techniques before the distance transform is computed. For example, it would be possible to draw bounding geometries to enforce a hard boundary around an object.

«A conclusion is the place where you got tired of thinking.»
Martin H. Fischer

Chapter 4

Conclusion and Future Work

The goal of this thesis was to explore and improve different approaches for guiding the design of multi-view visualizations. Ultimately, we wanted to introduce approaches that would make typical "expert-only" expressiveness available to non-experts. We presented solutions addressing multi-view visualization specification, reasoning, refinement, and aesthetics. The applicability and usefulness of our approaches is demonstrated with several examples and case studies.

We proposed a novel data structure for simplifying the specification of multi-view visualizations by using nesting as a first-class operator. In conjunction with flexible data mappings, this structure makes it possible to rapidly toggle between different nested visual representations. Furthermore, our flexible data mappings let the user adapt visualizations to the data format, rather than having to adapt the data format to fit the visualization template through data wrangling efforts. We then demonstrated the expressive power of these contributions with our visual builder, where non-expert users can specify high-fidelity nested visualizations as demonstrated by our generated examples.

Our focus then shifted to guided multi-view visualization design with an emphasis on automatic detection and resolution of design anti-patterns. We proposed semantic snapping, an approach that utilizes algebraic means for *reasoning* about a multi-view visualization, and then leveraged this to selectively resolve detected problems with corresponding operations. Our approach was made available to users with our proposed workflow that lets users execute understandable operations to address potential design problems. This workflow incrementally helps the user refine an existing design into a more consistent, and potentially more compact design as demonstrated by our case studies.

In our final approach, we addressed the issue of multi-view visualization layout by enabling users to transform a gridded layout into a "hand-crafted" layout where elements are positioned according to their contents. This was made possible by utilizing a force-directed layout where attractive forces model the layout topology, while repulsive forces model the contents by using distance transforms to detect and prevent overlapping. The effectiveness and utility of our approach is demonstrated by our examples, showing an existing grid layout and the corresponding layout generated by our approach.

Our presented research opens up several directions for future work. As the amount of data, and its complexity increases, so does the need for tools to make sense of this data. Without such tools, users will not be able to properly communicate about, or comprehend the data properly. Thus, we find it likely that tools for aiding the visualization design process will be needed not only by non-experts, but also by experts. It is therefore especially important to keep developing and consolidating approaches for simplifying the process of visualization design.

In the context of nested visualization design, one major challenge is the combination of nested visualizations with existing interaction, embellishment, and design techniques. For example, the drawing of links between nested elements at different hierarchical levels proved highly difficult. We believe there are two approaches to this challenge: (1) further breaking down and reformulating existing techniques to make them fit into the framework of nested visualization, or (2) making nested visualizations adapt to existing techniques. Both challenges are difficult, as nested visualizations involve several layers and elements, while existing techniques are often formulated for non-nested visualizations.

Furthermore, we believe that semantic guidance approaches such as our approach of semantic snapping will become more crucial within design tools as data and existing techniques become more diverse and complex. One major challenge here is to consolidate and formulate existing guidelines not only so that they can be interpreted programmatically, but also be applied to a wide range of different visualizations and dataset types. Existing research prototypes often only demonstrate the concept, but are yet difficult to apply “in practice”, or within existing tools. Unfortunately, most approaches assume a specific format of data which is not necessarily always the format used “in the wild”. Unified abstractions that encapsulate different dataset types and formats may be helpful in unifying approaches currently targeted towards specific kinds of data.

Furthermore, as it is becoming increasingly important to be able to communicate simple messages quickly, we expect that aesthetic, attention-grabbing artistic-looking layouts, especially for multi-view visualizations are beneficial for capturing the attention of the audience. Our final paper attempted to take one step in this direction by transforming an existing layout into a more artistic layout. However, we suspect that there is a need for more work on such layouts so that they can automatically adapt to different screen sizes, devices, and purposes. At the moment, information visualization dashboards are typically viewed on computer screens, but are also being viewed more and more on mobile devices. On such mobile devices, more compact and creative layouts are needed, and we believe this transition is already happening.

While all our approaches are separate contributions, one of the most promising areas of work in this field is to create higher-level grammars and systems that consolidate and unify existing approaches, making them more available to end-users. Experimental techniques are often accessible only to visualization experts. Thus, with all the recent development and growth in visualization grammars, layouts, and authoring tools, it is a good time to find commonalities between these techniques and make them expressible through a higher-order common language, that is also ideally accessible and understandable to non-experts.

During the development of these works, our main goal was to develop underlying mechanisms and integrate them into corresponding user interfaces. However, we did not take the step towards evaluating the user interfaces, as the development of the mechanism and interface proved to require both time and effort. An important future step for all these mechanisms is of course to evaluate them through empirical user studies and make according adjustments so that their power can be exposed to as broad an audience as possible.

Our works are also subject to another general problem with research prototypes, where the focus is on developing mechanisms as a proof-of-concept. However, the time and effort it would take to make our works bug-free enough to expose to a general audience was greater than the available time and effort available through the course of this PhD. However, we hope that the underlying concepts and mechanisms developed through this work can inspire future development within more robust, bug-free, industrial-standard systems.

In the combination of our work, we have explored several aspects of multi-view visualization design. Our work started with specification of visualizations which is largely manual, and then progressed towards increasingly automated methods. First, we provided means for reasoning about visualizations, accompanied by interactive and automatic methods derived from our means for visualization reasoning. Our approaches would likely be best when used in conjunction with each other and existing techniques, which we hope will be the case in the future. Thus, we aim to encourage visualization researchers to find means for consolidating existing design approaches by innovating higher-level solutions.

4

«Dude, suckin' at something is the first step to being sorta good at something.»
Jake the Dog

Part II

Included papers

4

Paper A

Visception: An Interactive Visual Framework for Nested Visualization Design

Yngve Sekse Kristiansen and Stefan Bruckner

University of Bergen, Norway

Abstract

Nesting is the embedding of charts into the marks of another chart. Related to principles such as Tufte's rule of utilizing micro/macro readings, nested visualizations have been employed to increase information density, providing compact representations of multi-dimensional and multi-typed data entities. Visual authoring tools are becoming increasingly prevalent, as they make visualization technology accessible to non-expert users such as data journalists, but existing frameworks provide no or only very limited functionality related to the creation of nested visualizations. In this paper, we present an interactive visual approach for the flexible generation of nested multilayer visualizations. Based on a hierarchical representation of nesting relationships coupled with a highly customizable mechanism for specifying data mappings, we contribute a flexible framework that enables defining and editing data-driven multi-level visualizations. As a demonstration of the viability of our framework, we contribute a visual builder for exploring, customizing and switching between different designs, along with example visualizations to demonstrate the range of expression. The resulting system allows for the generation of complex nested charts with a high degree of flexibility and fluidity using a drag and drop interface.

This article was published in *Computers & Graphics vol. 92*, pages 13–27, 2020. (doi: https://doi.org/10.1016/j.cag.2020.08.007)

A.1 Introduction

Nesting or embedding, i.e., the integration of additional visualizations into the marks of a chart, enables the presentation of information-dense graphical data depictions. By augmenting an outer visualization with additional details presented as information layers as part of its marks, rich depictions of complex data can be constructed from a few basic building blocks.

Nested visualizations are frequently applied in order to convey multi-faceted data and facilitate storytelling. In particular in fields such as data journalism, users would greatly benefit from being able to create such visualizations without programming.

In recent years, a new generation of visual authoring systems such as Data Illustrator [61] and Charticulator [82] have been developed to enable the creation of custom charts via intuitive visual interfaces accessible to non-experts. In particular, they aim to support the flexibility and customization options of design tools such as Adobe Illustrator, while still providing a data-driven visualization environment. While these systems feature advanced mechanisms for designing bespoke charts, they provide no or only very limited support for nesting.

In this paper, we present Visception, a visualization framework built from the ground up based on nesting as a first-class operation. For this purpose, we introduce the VC-tree as our central data structure. We detail how this approach offers flexible data mappings for transforming tabular input data into data objects, enabling the expression of a wide range of different groupings when creating a nested visualization. Individual charts are made composable with other charts through our framework's implicit handling of nesting and deformation. By providing a set of simple operations to manipulate a VC-tree, we are able to realize a large number of different embedding and layering relationships. Furthermore, we show that using a more inclusive definition of what constitutes a data-mappable channel provides additional design flexibility in the context of nesting. The full functionality of our framework is exposed in the form of a visual builder, and we demonstrate that our approach allows for the easy generation of a variety of complex nested visualizations.

A.2 Related Work

A.2.1 Formal Graphics Specifications

Foundational works like Bertin's Semiology of Graphics [10] and Wilkinson's Grammar of Graphics [108] provide constructs for concisely specifying and reasoning about graphics. Bertin describes marks as basic graphical units, and visual variables as modifications (position, shape, value, color, orientation, texture, and so on) that can be applied to marks. Munzner [73] consolidated and extended similar approaches for discussing visualizations, and also introduced the term channel as a way to control the appearance of marks. Visception features a more inclusive kind of channel, denoted as a VC-channel. Layout parameters as well as global chart properties such as for example the title or background of a chart, are exposed as VC-channels.

Early information visualization techniques utilized low level libraries. While such low level libraries enable the expression of graphics, they are not necessarily suitable for visual thinking. Thus, multiple visualization toolkits that raise the level of abstraction have been developed. Examples of such libraries include Prefuse [39], Protovis [13] and D3 [14].

At an even higher level of abstraction, **visualization grammars** such as Vega [112] enable clear expression of a wide range of visualizations declaratively. In Vega, each chart is a unit which takes in data and associated transformations, mark type, and encodings. Each encoding is a specification for how a channel is mapped. Built on Vega, Vega-Lite [87] is both a grammar of interaction and graphics. A prominent feature of the Vega-Lite grammar is its view composition algebra with four operations: *Layer* for placing one chart on top of another, *Concatenation* for placing charts side by side, *Facet* for creating one view per distinct value of a field, and *Repeat* for creating several views with the same input data. Visception uses nesting and data groupings independent of chart inputs to provide a more flexible and recursive nesting behavior. In terms of data, Visception's nesting operation corresponds to Vega's facet operation. Visually, Vega provides rows and columns as host spaces for child charts, while Visception provides a set of customizable charts as host spaces. These charts are more flexible and controllable than rectangular host spaces generated via the facet operation, and may be mapped to data. Visception's VC-channels function similarly to Vega's encodings, although each chart in Visception will have a larger set of VC-channels to modify the layout.

Tree visualization grammars are closely related to nesting. Li et al. [58] introduced a declarative grammar of tree visualizations, enabling users to rapidly specify both explicit and implicit depictions. Their visual builder allows the user to combine different tree layout algorithms, and to adjust finer aspects such as margin and padding between nodes. Visception has a similar approach in that it combines layouts. However, Visception is focused on enabling the creation of nested visualizations from tabular data, while GoTree is focused on creating tree visualizations from predefined hierarchies. Schulz et al. [90] propose a set of functional building blocks denoted as layout operators that enable building explicit node-link layouts as well as implicit space-filling layouts. They specify a highly flexible layout pipeline for rendering such trees, and expose operators that allow the user to modify the layout in a variety of ways. Similarly, Visception uses a layout pipeline in its underlying implementation, and exposes parameters that modify the layout as VC-channels. Visception only does top-down explicit layouts, and does not work bottom-up as Schulz' generative approach.

Other more **specialized grammars** focus on particular categories of visualizations. ATOM [76] is grammar for unit visualizations. With this grammar the user can subdivide a space at multiple levels and fill in units — one for each datum. Visception exposes a similar design space by providing a *unit* chart type. Wickham and Hofmann's Product Plots [107] is a framework for transforming and combining area-based visualizations. They define three 1D primitives: bars, spines and tiles. With these three primitives, they show that it is possible to express a wide range of both simple and complex visual representations of data. While both Product Plots and Visception use nesting, Visception leverages the chart type itself to provide host spaces for child charts, while Product Plots subdivide rectangles with a small set of rectangle-based 1D primitives.

Schulz and Hadlak [92] presented a way of representing visualizations by blending together existing visualizations defined as presets. In the process of describing how to interpolate between visualizations, they expose connections between different chart types, such as the polar area chart and the bar chart. Rather than presets, Visception offers a set of charts the user may choose and combine. Thus, Visception covers a discretized subset of the design space covered by preset-based visualization.

We are inspired by these approaches of combining and deforming 2D geometries, and use such concepts to handle nesting and deformation behavior for all chart types within our framework. Vuillemot and Boy [105] use nested and composite visualizations to facilitate the exploration of designs, regardless of data. They define a visual grammar with partitioning patterns and data transform operations. With our framework, we enable the specification of charts without requiring explicit specification of nesting behavior, making it easy to introduce new building blocks to the language.

A.2.2 Data Exploration and Visual Authoring

Data exploration tools focus primarily on what the user can learn about the data, rather than design and aesthetics. IVEE [2], Visage [83], and Tioga2 [3] were some of the first systems to enable visual building of queries, and visualizing the results. Polaris [97] by Stolte et al. (later commercialized as Tableau) enables rapid exploration of large multidimensional datasets, leveraging a table algebra to display a wide range of charts.

Visual authoring tools are more focused on design than data exploration, yet they serve a similar purpose and can potentially be as powerful as data exploration systems. Charticulator [82] enables the user to define a chart by articulating compound marks or glyphs, as well as links between these. Lyra [85] enables the interactive design of a wide range of visualizations using drag and drop operations. Lyra also provides visual data pipelines that allow for the expression of advanced layouts and data transformations. iVisDesigner [81] aims to cover a wide range of the visualization design space by leveraging modular visualization design. Data Illustrator [61] augments vector design tools with new concepts and operators, enabling users to bind parts of a vector-based illustration to data. Data Driven Guides [52] has a similar approach to Data Illustrator, allowing users to create data-driven shapes (also known as guides) that can be decorated by custom vector graphics. iVolver [74] provides users with the means to extract and reconstruct visualizations from both data sets and existing visualizations (including images and webpages).

While our work shares many of the general goals with the approaches mentioned above, Visception focuses on nested, aesthetic visualizations. We provide an editor and a framework to enable the design of highly customized information rich visualizations by leveraging nesting. Our definition of charts with VC-channels are made to be compatible and consistent for both nested and non-nested charts.

A.2.3 Nested Visualization and Related Techniques

Hierarchical and small-multiple layouts are closely related to nesting. Schulz et al. [91] surveyed the design space of hierarchy visualization, providing an overview of a large number of different techniques (both 2D and 3D) used to visualize hierarchies, as well as exposing unexplored parts of the design space. LeBlanc et al. [57] describe the technique of dimensional stacking, allowing the user to map high-dimensional data to a relatively small 2D space. Similar expressiveness can be achieved in Visception by nesting *rows* and *columns*. Wang et al. [106] introduced the circle packing layout, nesting circles within circles with arbitrary depths. This layout may also be expressed by nesting circles with a force-directed layout [33] within one another, which is achievable in Visception by nesting *plot* charts. Treemap layouts are often used to visualize hierarchies. Baudel et al. [8] present a generic algorithm that expresses most of the different existing treemap layouts using only a few basic operations. Visception follows a similar line of thought: to expose parametrized generic charts that may express other specific charts.

Using **nested visualizations** it is possible to express complex relationships by only using a few simple building blocks. Parker et al. [77], as early as 1998, designed NestedVision3D, allowing for the exploration of nested graphs to explore the structure of computer programs. ZAME [28] (Zoomable Adjacency Matrix Explorer) nests glyphs inside each cell of an adjacency matrix. Combined with zooming, panning, and aggregation represented as glyphs, ZAME allows for the exploration of large datasets. Javed and Elmqvist [49] detail four visual composition operators: juxtaposition, superimposition, overloading and nesting. Visception provides a flexible layering operation that, combined with movable and resizable bounds, achieves a similar level of expressiveness as using these four operators. Juxtaposition and superimposition are expressible by simply editing the bounds of a chart. HEDA (Heterogenous Data Attributes) [63] is a generic interactive visualization component that aims to enable users to explore heterogenous data as a reorderable matrix. Visception maintains reorderability, but as a side notion with an *order* VC-channel exposed for reorderable charts and focuses on providing a visual language for building nested visualizations. Slingsby et al. [95] explored the use of different layouts with editable hierarchies to incrementally answer research questions. Their approach could be described as explorative nesting of data. They define the language HiVE (Hierarchical Visualization Expression Language) which includes operations for editing, deleting, inserting and swapping different levels of a hierarchy. Visception can express similar hierarchies as HiVE, with more focus on design flexibility and support for a wider range of chart types. NodeTrix [40] enables the visualization of large networks using juxtaposition and overloading by linking adjacency matrices together. It combines the node-link diagram and the adjacency matrix into one visualization, enabling the designer to show more data and data relations using less visual space. Similarly, Domino [37] uses overloading and juxtaposition to compare and manipulate subsets across multiple datasets.

Overall, multiple specific cases of using nesting have been explored by related works. In Visception, we go beyond existing solutions by enabling the specification of nestable charts without needing to specify nesting behavior. We also enable the user to specify a wide range of different data groupings without having to modify the dataset. Finally, we include layout parameters and global properties as VC-channels, making many specific chart types expressible as configurations of a more general chart type. Visception's use of charts as building blocks follows the same line of thought as Pattison et al. [78] who proposed a "generalized layout", similar to treemaps but with more available intra-container layouts.

Term	Explanation
Chart	A chart has a type and associated tabular input data that is represented by output marks.
Chart type	A type of chart within the Visception framework, consists of its own layout that is controlled by some of its VC-channels. The layout controls the shape and position of the chart's output marks.
Output marks	Graphical marks of a single chart.
VC-channel	Controls a particular aspect of the appearance or layout of a chart.
VC-channel mapping	The assignment of data (for example a dimension) to a VC-channel (for example *fill color*).
Layout space	Normalized space in which a layout is initially computed.
Parent space	Space(s) in which a chart's output marks are embedded according to the computed layout. If the node is a root node, the parent space is simply the root viewport. Otherwise, a chart-dependent region within the output marks of the parent chart.
Data object	Represents a selection from a tabular dataset. May be one of the following: 1) A single row, 2) A list of rows, 3) A list of values.
Chart input data	A set of data objects inherited from the parent VC-node. If the node is the root VC-node, the input data is the list of rows of the entire dataset.
Chart mapping	Transforms every input data object into a new set of data objects.
Chart output data	A set of data objects, used for rendering the chart, and possibly as input data to child charts.

Table A.1: Terminology used throughout the paper.

A.3 The Visception Framework

In this section we detail our framework for nested visualization design. As the use of certain terms varies in the literature, we present the terminology used here in Table A.1. First, in Section A.3.1, we discuss how individual charts are represented and manipulated in Visception. We then introduce the VC-tree, our central data structure that enables the specification of and interaction with a nested visualization in Section A.3.2.

A.3.1 Charts and VC-channels

Charts form the basic building blocks of Visception. A chart transforms tabular input data into output data objects, which are then used to generate graphical elements referred to as output marks.

VC-channels represent the parameters that control the layout and style of a chart. Layout VC-channels affect the shape or position of a chart's output marks directly and are hence typically specific to a particular chart, while other VC-channels affect the styling of the stroke, fill, drop shadows, etc. and are generally shared among multiple charts. As a convenience, bundles of common channels are represented as three general

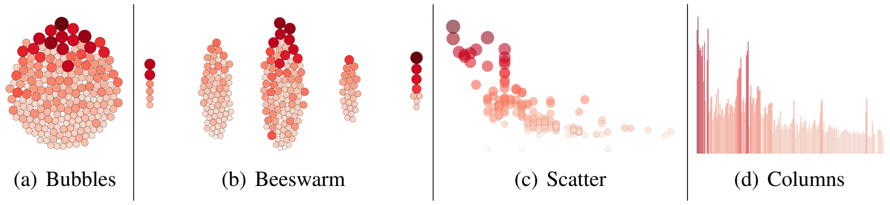


Figure A.1: Four visualizations with the same chart mapping, and different channel configurations. The first three charts (a-c) are different configurations of a *plot*, while (d) is the *columns* chart equivalent of (c).

chart prototypes, which are then used to compose more specific charts: A *base* chart with all the common VC-channels for high-level transformations applied uniformly to the entire chart, a *stroked* chart with all VC-channels relating to the stroke, and a *filled* chart with all VC-channels relating to the fill of a chart.

Layout VC-channels allow the user to control different aspects related to the layout of a chart. For instance, a bubble chart, a beeswarm plot, and a scatter plot, instead of being available as separate chart types, can be expressed as configurations of one flexible chart as illustrated in Figure A.1a–c. In A.1a the *collision radius*, which is a scaling factor of the repulsive force between the nodes of a force layout, is set to 1.0, while in A.1c it is set to 0. A.1c also has both the x and y VC-channels mapped to data. In A.1b we see that the *force x* is higher than *force y*, causing the circles to accumulate along the vertical axis. All VC-channels available in Visception can be seen in A.8, Figures A.21 and A.22.

Some attributes of a chart control global visual elements instead of the appearance of individual marks. They are usually referred to as properties in related works [82] and typically cannot be mapped to data since such an operation has limited utility. However, when nesting is introduced, the data of the parent chart can serve as input data for these properties, making them meaningfully mappable to data. For these reasons, such properties are also exposed as VC-channels, enabling an increased level of design flexibility without introducing additional complexity. For instance, we can adjust the *stroke width* of a nested chart based on the parent datum or use a categorical dimension to enable/disable effects such as drop shadows for a subset of the data.

Similar to Vega-Lite [87], we populate all VC-channels with editable default values. For example, if a *columns* chart is created, the *bar height* VC-channel is by default mapped to the (editable) *value* of 1, resulting in *N* bars with the same height. These defaults allow for rendering the chart at its intermediary stages, without requiring a complete specification of data mappings.

Furthermore, in order to be able to switch between chart types while preserving existing data mappings as much as possible, we maintain a set of semantic VC-channel equivalences between chart types as shown in Table A.2. For example, if we change a *plot* to a *columns* chart (see Figure A.1 c–d), there are two equivalence groups: one containing the *plot y* and *bar height* VC-channels, and another containing *plot x* and *bar order* VC-channels. Hence, these existing data mappings can be transferred to the new chart.

Group	Explanation	VC-channels
Position Major	Most significant VC-channel controlling the position of the marks.	plot x, line x, bar order, circular bar order (for the charts: polar area, sectors, tubes)
Size Major	Most significant VC-channel controlling the size of the marks.	tile size, unit size, stream size, plot size, bar height
Position Minor	Secondary VC-channel controlling the position of the marks.	plot y, line y
Size Minor	Secondary VC-channel controlling the size of the marks.	bar width (columns), bar height (rows), tube height (tubes chart)

Table A.2: Examples of equivalence groups within the implementation of Visception. We followed Munzner's [73] ranking of channel types by effectiveness to determine the most significant channels.

A.3.2 Visception Tree

Visception aims to be a visual and conceptual framework for nested charts by enabling operations for building and editing a hierarchy of charts, as well as implicitly handling common nesting behaviors for multiple charts. Other works such as HiVE [95] and ATOM [76] enable setting up hierarchies of charts and data. Drawing inspiration from these previous approaches, we propose the Visception Tree (VC-tree) data structure. The VC-tree provides fine-grained control over data mappings at different hierarchical levels, and implicit handling of deformation and nesting behavior. In this section we will go into detail on how the VC-tree encapsulates a tree of charts, data mappings, and spaces.

Structure and properties: The VC-tree consists of VC-nodes which have two explicit properties: A chart type and a data mapping. The data mapping represents a chosen grouping of the chart's input data, while the chart type defines the layout and thus the transformation of the output data into output marks that make up the visual representation of the data. When a VC-node is the child of another VC-node, this corresponds to nesting one chart within another. The contents of the nested chart is then displayed within the output marks of the parent chart [49]. For example, nesting a *plot* within a *columns* chart with N bars will result in N plots, one within each bar. The left-to-right order of nodes corresponds to the Z-index, with the leftmost nodes rendered on top as shown in Figure A.2. VC-nodes can be added, moved, and deleted as also illustrated in Figure A.2.

Data mappings: Each VC-node has both input and output data, consisting of a number of data objects. The output data of a node is implicitly defined by its data mapping and input data. Depending on the type of data mapping, a data object may represent a row in the dataset, a list of rows, or a list of values.

Each node's input data corresponds to the output data of its parent (at the root of the tree, the input data is the list of all rows in the tabular dataset). The data mapping of a node turns its input data into output data as shown in Figure A.3. For each chart corresponding to a VC-node, the output data is used for generating geometric shapes. A VC-node is considered nestable, i.e., it can have children, if it has one areal output mark per output data object. In order to enable nesting without requiring a very specific

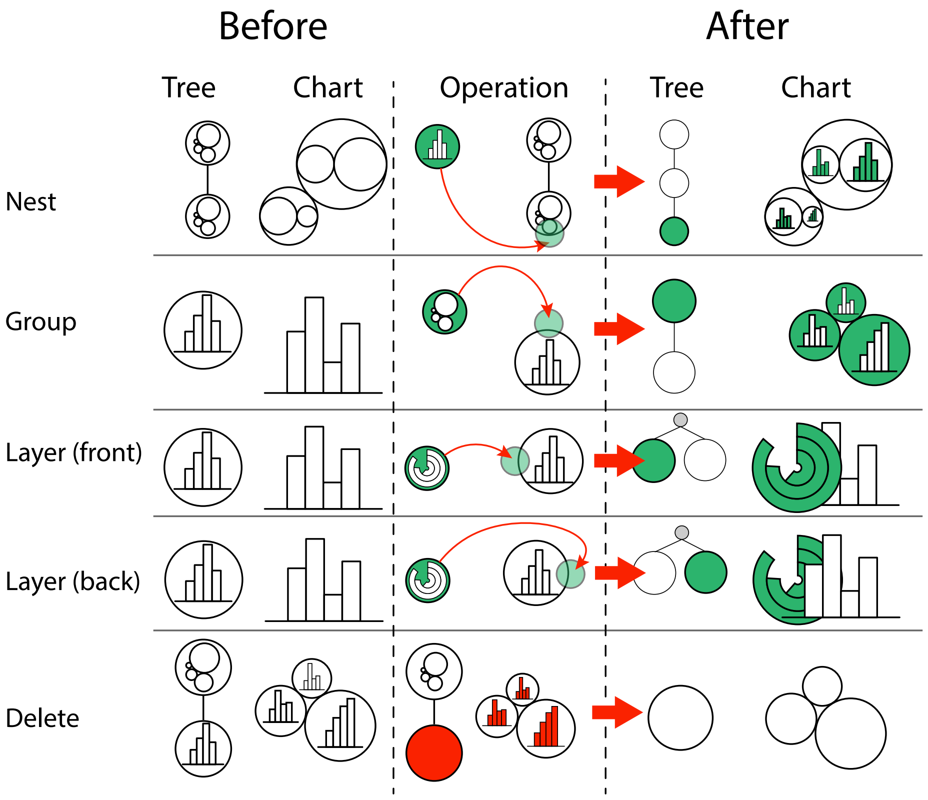


Figure A.2: This figure depicts operations that can be performed on a VC-tree, along with example charts. Note that the *move* operation is not shown here, since it corresponds to a *delete* operation followed by *nest*, *group* or *layer* operation that re-inserts the node into the hierarchy. Red represents a deleted chart, while green represents an added node

dataset format, we define four types of data mappings, *dimension*, *all*, *monolith*, and *identity*, which are summarized in Table A.3.

For *dimension* and *all*, the input data objects must always represent a list of rows to be applicable. The dimension mapping corresponds to grouping by a given dimension D. Thus, we partition each data object by distinct values of D. For example, if the input data is a single data object representing a list of all rows, and the mapping is a dimension D, the output data will be a set of data objects, each object representing a distinct value of D with a list of matching rows. If the data mapping is set to *all*, each input data object (always representing a list of rows) is “unpacked” into a list of data objects, each containing one row. For instance, if an input data object is a list of three rows, an *all* mapping would output three data objects, each containing one row. The *identity* mapping generates a list of data objects identical to those of its parent. Creating such “dummy” data objects allows for overloading charts with extra information by nesting more charts within existing marks as shown in Figure A.9. The *monolith* mapping creates one data object for every specified numeric dimension. For the list of rows

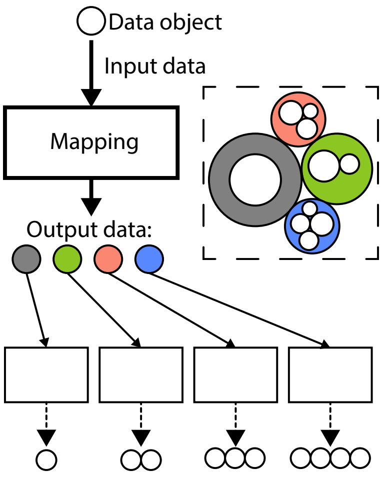


Figure A.3: A data mapping transforms every input data object into a set of new data objects. By applying this relationship recursively, each node can compute its own output data.

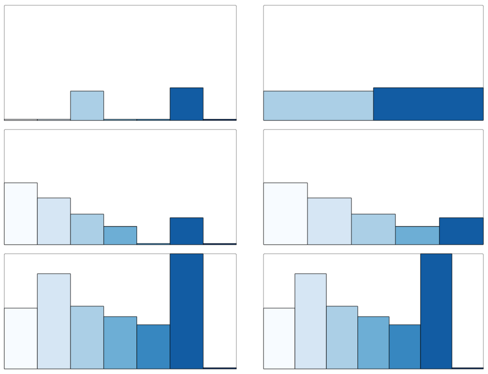


Figure A.4: On the left, we see a non-sparse mapping, which includes empty data. On the right, we have a sparse mapping showing only non-empty data objects. Observe how the non-sparse mapping horizontally arranges the innermost bars uniformly.

within every input data object, it generates one data object per dimension, containing the list of values of that dimension. Intuitively, the *monolith* mapping can be seen as "one mark per data dimension". An example of this mapping can be seen in Figure A.8.

In the case of nesting, the *all* and *dimension* mappings can be specified as *sparse* or *non-sparse*. A sparse mapping is the default, and will only create data objects that exist when nesting. For example, suppose a dataset is grouped by gender: male and female. Furthermore, the hair colors of both males and females include red, brown, grey, black and white, but only the female set has brown and grey hair. Then, if we apply a sparse mapping, the female set will include a mark for red and brown, but the male set will not. If the mapping is non-sparse, empty data objects are generated in all sets of data. This construct is useful when, for instance, creating bar charts on grids, and makes the nesting uniform as shown in Figure A.4. The examples shown in Figures A.12 and A.13 show the implications of a sparse vs. a non-sparse mapping.

Space transformations: Since the layout of an individual chart only outputs geometric shapes into a normalized space denoted as the *layout space*, the framework must handle the rest of the nesting behavior. Existing works have already addressed the problem of deforming/transforming charts. For example, Schulz and Hadlak [92], Wickham and Hofmann [107], and Charticulator [82] transform charts from Cartesian to non-Cartesian spaces. ATOM [76], Vega [87], Vuillemot and Boy [105] compose different layouts in Cartesian spaces via nesting. Using similar methods, the layout component of the Visception framework transforms the shapes from a normalized space to fit within the parent space.With nested charts, we need to consider two different spaces in order to render the chart. Given a parent-child pair of VC-nodes, each mark of the

Mapping	Description	Cardinality
Dimension (-,+)	Groups every input data object by a given data dimension *D*, with, creates one data object per existing (if not sparse-mapped) value of the domain of *D*. Always nestable.	between 0 and number of values in domain of *D*
All (-,+)	Ungroups input data object, creating **one data object per row** in the input data object. If sparse-mapped, this will create one data object per row in the root dataset. Nestable only if the child is *identity*-mapped	between 0 and number of rows in dataset
Monolith (-)	Creates *D* data objects per input data object, given *D* dimensions. Not nestable.	*D*
Identity (-)	Creates **one** identical data object per input data object. Always nestable.	1
(-) Sparse	Mapping **excludes** empty rows and data objects when nesting.	
(+) Non-sparse	Mapping **includes** empty rows and data objects when nesting.	

Table A.3: A summary of all possible data mappings in Visception. Not all mappings are nestable, some are only nestable if the child has a certain mapping. The cardinality of each mapping describes the size of each set of data items per parent data object. If a non-identity, non-monolith mapping results in only sets with a cardinality of 1, this is equivalent to an *identity* mapping.

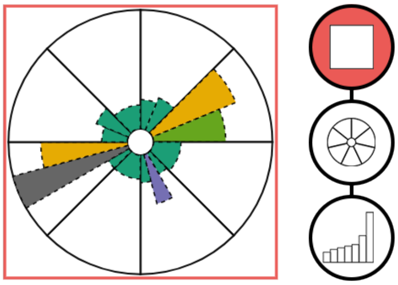
parent node holds an inner space. We refer to this space as the *parent space*. Each chart is first defined in *layout space*, a normalized space in which the shapes of each chart are calculated. Our framework implicitly handles nesting and deformation behavior by fitting layout shapes into a parent space. Figure A.5 shows how a *columns* chart can be fit into both a Cartesian and a circular parent space.

The transformation from layout space to parent space depends on the type of the parent space, the layout shapes and the specific defined behavior for the shapes of the chart. For example, nesting a *columns* chart within a *polar area* chart corresponds to transforming rectangles to fit within arcs. If the parent space is an arc, charts nested within each arc will be either deformed or fit within the arcs. For *columns* charts it makes sense to deform the rectangles as shown in Figure A.5b, whereas for scatter plots or force-directed layouts it makes sense to deform the position, but not the shape. These deformations can be reduced to a matter of either fitting (scaling and translating the whole shape to fit within the parent shape), or deforming 2D shapes by transforming every coordinate into the coordinate system of the parent. For example, to transform a rectangle to an arc, we simply transform the Cartesian (x,y) coordinates of each corner into the polar coordinates of the parent arc.

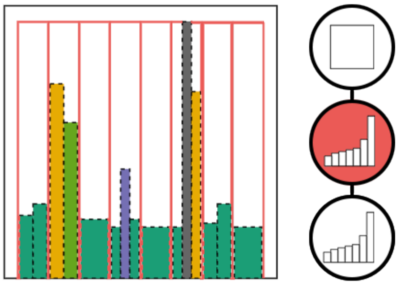
A.4 Implementation and Visual Builder

A.4.1 Implementation

Visception was implemented as a web application using VueJS for the front-end UI components, and D3 [14] for rendering the SVG. A prototype of the framework is



(a) Since the immediate parent space of the *columns* chart is an arc, the bars are deformed into arcs that fit within the parent arcs.



(b) As the parent's space is Cartesian, the bars are fit into the Cartesian coordinate system of the parent marks.

Figure A.5: Two examples of nesting with different types of parent spaces. If the parent space is deformed, each bar of a *columns* chart is also deformed as seen on the left.

available at `https://vis.uib.no/visception/`.

D3's data selections allow for creating SVG elements on a per-datum basis. This also enables creating a set of child elements for each parent element. Our implementation heavily relies on this mechanism for specifying a hierarchy of SVG groups and paths corresponding to the hierarchy of data.

The VC-tree and its VC-nodes act as a skeleton for the rest of the logic. Each VC-node has a channel manager, layout manager, guides manager, chart type, and data input. With this information, each node can compute its own layout and style. The VC-tree was realized as a simple tree data structure, with functions for moving, adding, and removing VC-nodes. Each VC-tree is tied to an SVG element, and each VC-node to a D3 selection representing the chart's output marks.

Data queries and local selections: Whenever the data mapping or chart type changes, a data query is made, and the selection of the node is updated accordingly. The data mapping and chart type of the VC-node is used to query the dataset, and thus infer the cardinality of the selection.

Layout: Whenever a VC-channel affecting the layout changes, the layout step, which itself is implemented in the form of a pipeline, is executed. The layout computes the position and shape of each mark of a node's selection. Since most charts have commonalities, we implemented a general layout pipeline where we can easily replace/insert steps for customization, but also reuse many steps across multiple chart types.

Guides: After the style or layout has changed, the guides of the chart are rendered, independent of the layout pipeline. Floating guides, such as color legends (see Figure A.8) are rendered to a group at the root of the SVG. Fixed guides such as axes (see Figure A.20) are rendered in a selection local to each node. While guides are not the main focus of this paper, they use a similar mechanism to the layout pipeline for rendering, and are deformable as seen in Figure A.20. The geometric components of the axis are transformed along with the output marks of the corresponding chart. Both axes and floating guides can be styled using VC-channels.

Component	Information displayed	Functions
Canvas	Result visualization.	Receive drops, mapping data to chart or VC-channels.
Data	Data mappings, dimensions and aggregates.	Initialize drags.
Outline	Hierarchy of charts (a VC-tree) & data and chart type of each node, selected node.	Receive drops (map data to chart), rearrange hierarchy (group, nest, layer), changing chart type of node, selecting a node.
Channels	VC-channels of selected chart.	Receive drops (map data to VC-channel), edit individual VC-channels.
Guides	Guides (legends and axes) of selected chart.	Edit guide by editing channels.

Table A.4: A summary of information each user interface component shows, and which functions it addresses. Together, these views enable the creation of nested charts, editing individual charts and accompanying VC-channels and guides.

A.4.2 Visual Builder

Design and components: Our visual builder uses drag and drop operations to expose a majority of the framework's functionality. An overview of the user interface is shown in Figure A.6 and the main functions provided by each of the components are summarized in Table A.4.

The *data view* enables the user to drag data mappings, dimensions and aggregates. Dragging an item from the data view expresses an intent to map that item to a chart or a VC-channel and potential drop targets are immediately highlighted. The data view provides all possible data mappings and individual data dimensions. A dimension can be clicked and expanded into a set of draggable aggregates (see Figure A.6). We currently provide the following aggregation functions: *sum*, *quartile*, *quantile*, *median*, *min*, *max*, *avg*, *distinct*, and *count*. Furthermore, by dragging the respective icons, the user can indicate whether the dragged mapping should be sparse (▫▫) or non-sparse (▪▫▫). Dragging the ▫ tile corresponds to dragging a *monolith* mapping of the dimension. A drag and drop operation of a data mapping or dimension aggregate can express a wide range of operations as shown in Table A.5.

All drag operations originating from the data view have three possible drop destinations: the *outline view*, *channels view*, or *canvas view*. Thus, the data view is placed in the center to all these views. The canvas view shows the rendered charts, and accepts both chart and channel mappings. When the Visception builder is initially opened, only the canvas and data views are visible. When a data mapping is dropped onto the canvas, the outline view and channels view appear. The outline view provides a high-level overview by showing the hierarchy of charts, and enables the expression of operations such as nesting, grouping, layering (see Table A.5), as well as changing chart types. When a node in the outline view is clicked, it is selected. When selected, the channels view displays all available VC-channels for a chart, and enables the user to edit and map data to individual VC-channels. For editing guides, we provide the *guides*

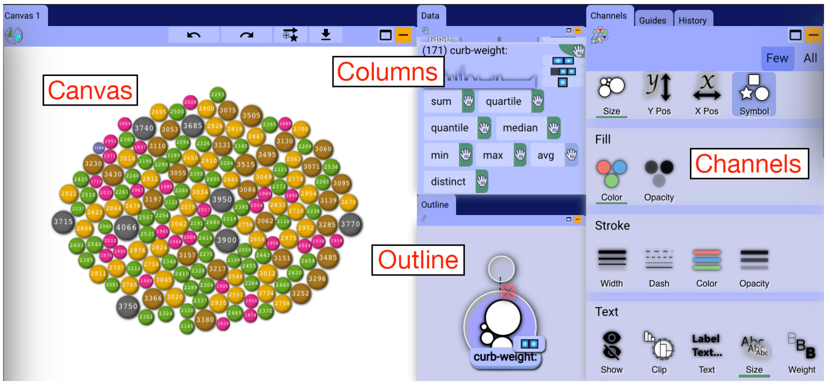


Figure A.6: A screenshot of all views exposed within Visception. Note how the Guides view is within a tab in this example. The guides view lets the user select a guide, and shows a list of VC-channels (similar to the Channels View displayed on right) that the user can edit the guide's style through.

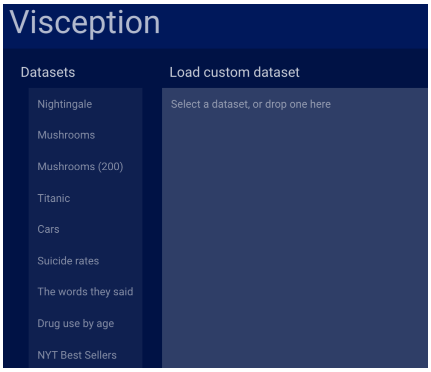
Target	Area	Operation	Result
Channel	Center	Map (VC-channel)	Map D to VC-channel
Outline Node	Center	Map (chart)	Map chart to D
	Left	Layer (front)	C' layered on top of C
	Right	Layer (back)	C' layered beneath C
	Top	Group	C nested within C'
	Bottom	Nest	C' nested within C
Canvas	Center	Map	Map chart to D
	Bottom	Map (VC-channel)	Map D to C x-axis
	Left	Map (VC-channel)	Map D to C y-axis

Table A.5: When dragging and dropping a data dimension, there is a limited set of available operations and outcomes. D denotes a dragged data dimension, or aggregate of an dimension. C is the selected chart, and C' is a new chart grouped by D.

view that allows the user to select a guide and edit its VC-channels, in the same way the VC-channels of a chart are edited. These views allow for expressing and editing a hierarchy of charts, as well as individual charts.

Example workflow: Here we demonstrate a general workflow considering the main operations of mapping data to charts, mapping data to VC-channels, editing the hierarchy of charts, and editing VC-channels.

After selecting a dataset (see Figure A.7a), the user initiates a drag operation on a data mapping. This operation highlights possible drop areas and a preview of the result will be shown (see Figure A.7c–d). If the chart is empty, the only drop area will be the visualization canvas, as shown in Figure A.7b. After the drop operation, the dropped data mapping becomes the data mapping of the chart, and the user will see a chart grouped by the given mapping (see Figure A.7d).



Visception

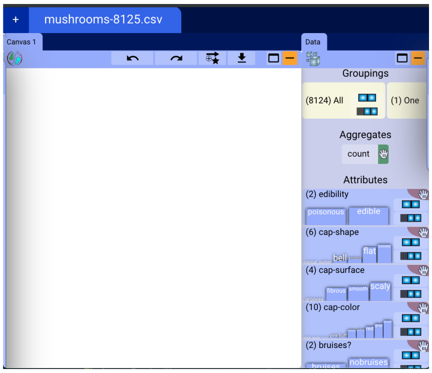
Datasets

- Nightingale
- Mushrooms
- Mushrooms (200)
- Titanic
- Cars
- Suicide rates
- The words they said
- Drug use by age
- NYT Best Sellers

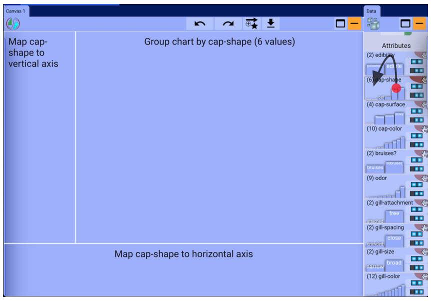
Load custom dataset

Select a dataset, or drop one here

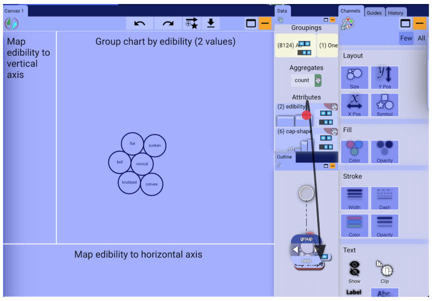
((a)) The user can select, or load a template csv dataset.



((b)) After loading a dataset, the user can see the data dimensions, and an empty canvas.

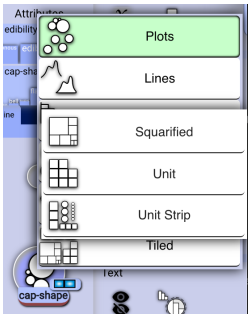


((c)) When initializing a drag, only the canvas is highlighted as a drop zone.



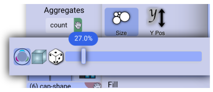
((d)) After creating a chart, the outline and channels view are made visible. When dragging a data dimension, nodes in the outline view and VC-channels in the channels view, are highlighted as drop zones.

Figure A.7: Visception, getting started step by step.



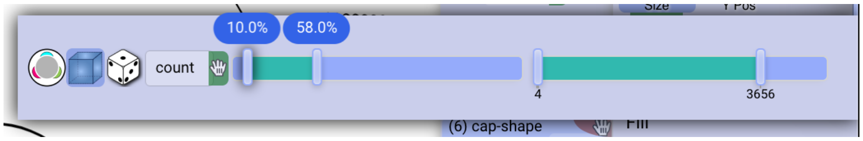
With a non-empty chart, two more views will appear: the outline view and the channels view, as seen in Figure A.7d. The outline view shows a tree corresponding to the current hierarchy of charts. The user can select a node by clicking it. If it is clicked again, a chart menu is shown (depicted to the right), enabling the user to change the chart type. When a node is selected, the channels view will display the VC-channels for that chart. For example, in Figure A.7d the channels view corresponds to a *plot*.

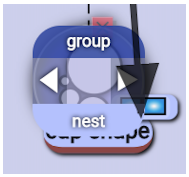
With four active views there are some operations to consider. If the user wants to edit an individual chart, the chart is selected by clicking it in the outline view. When that chart is selected, the channels view will display a set of editable VC-channels in the form of small labeled icons, grouped into categories. The category helps the user decide what to edit on a general level, while the icons and accompanying labels provide more specific hints. When the user has found and clicked a VC-channel, the widget for editing it pops up. The widget can be a slider, a color picker or another kind of control. Undo/redo functionality allows the user to try out different controls and learn from resulting changes to the chart.



By interacting with the widget, and immediately seeing the results, the user can learn what the VC-channel does. Sequentially editing VC-channels lets the user control one aspect of the chart at a time. The user must also be able to map data to a VC-channel. When initializing a drag operation of a dimension or dimension aggregation, potential target channels will be highlighted. An example of this is shown in Figure A.7d. If the user drops a dimension on a VC-channel, a corresponding mapping is created. When a VC-channel is clicked, the user may turn a mapping on or off, and the widget will change accordingly.

For example, if a VC-channel is mapped to a dimension, the user can edit the output range (for example [0%, 100%] on the x-axis, and the domain (for example [0, 20] even though the dataset only contains [8, 20]).





When a dimension is dragged, the user can drop it on one of the areas of the node. These areas appear when the drag is initiated. Table A.5 illustrates the outcomes of a drag operation.

A.5 Results

Here we demonstrate a gallery of example charts created with the Visception builder. Each example is accompanied by a screenshot of the outline view displaying the corresponding hierarchy of charts. Each chart is generated by creating such a hierarchy and applying styling/mappings to one chart at a time.

We selected a broad range of different datasets in order to demonstrate a wide variety of data mappings and chart hierarchies. Our generated examples cover a variety of designs and aim to demonstrate the generative expressiveness of our framework.

Nightingale's rose is an early and well-known data visualization, used by Florence Nightingale to illustrate avoidable deaths of soldiers during the Crimean war. A row in the dataset holds a month number, army size and death counts.

Here we demonstrate the usefulness of nesting and the *monolith* mapping. The chart is created by leveraging the *monolith* mapping to nest the dimensions *disease, wounds, other* as a *vertical stack* inside a *polar area* chart mapped to *all*, as shown in Figure A.8.

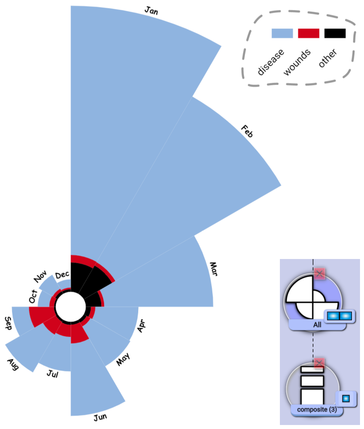


Figure A.8: Nightingale's Rose

UCI's Mushroom dataset [88] has been widely used for machine learning, and as an example dataset for visualizing categorical data. It has 22 dimensions and over 8000 rows. Each row in the dataset is one mushroom sample. Here we will see how the *identity* mapping can be used to decorate output marks at different levels of nesting.

First, we consider 200 samples of mushroom (see Figure A.9) representing the hierarchy *gill size* $\rightarrow$ *stalk-surface*. Within each container, we have a *unit* chart representing all rows in the dataset (one square per row). Within each unit, we nest an *identity*-mapped (one datum per parent datum) *plot* where the *symbol* is mapped to *cap-shape*. The *identity* mapping allows us to overload the unit squares with more information, in this case the *cap-shape*. For the second visualization, we show all 8124

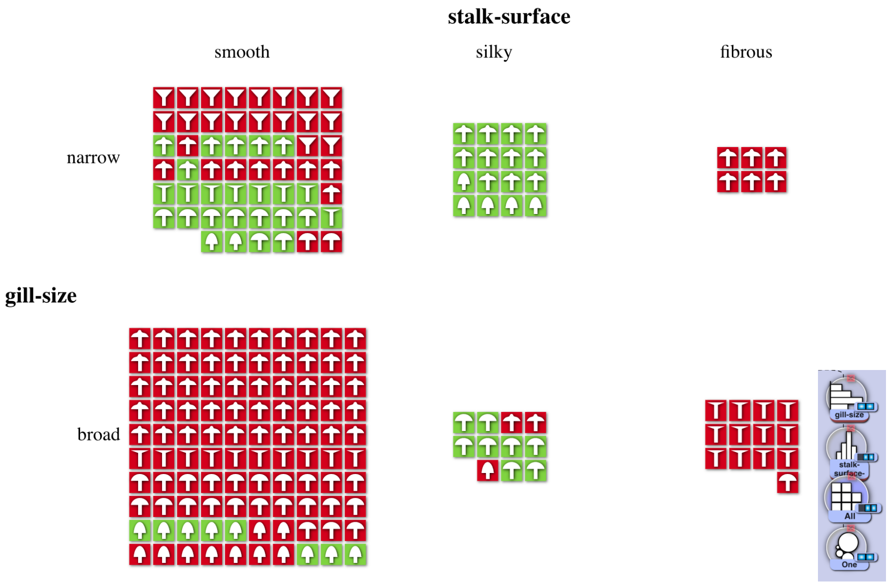


Figure A.9: 200 samples of mushroom.

rows (samples) of mushrooms, by displaying the hierarchy *cap-surface* $\rightarrow$ *cap-shape* using the *squarified* chart, with its *size* VC-channel mapped to *count* (the count of rows per aggregation). Inside each square is a *unit* chart, showing one unit per mushroom. Layered over the unit chart, an *identity*-mapped *plot* (lower left node) has its *symbol* VC-channel mapped to the cap surface, and helps show the cap shapes. With both layering and nesting available, we can overload charts with great flexibility as shown in Figure A.10.

"At the National Conventions, the Words They Used" was published by the New York Times in 2012, illustrating how much different words are used by different political parties. We create a similar visualization based on data from the 2016 presidential election. Each row in the dataset represents the following: *(word, name, mentions, Trump, Obama)*. Here we will see how the *clip* VC-channel, combined with nesting can "slice" the circles.

First, we nest a *columns* chart within the circles, and enable the *clip* VC-channel, and use the *bounds* VC-channel to stretch the columns to properly cover their parent shapes. The final result is shown in Figure A.11.

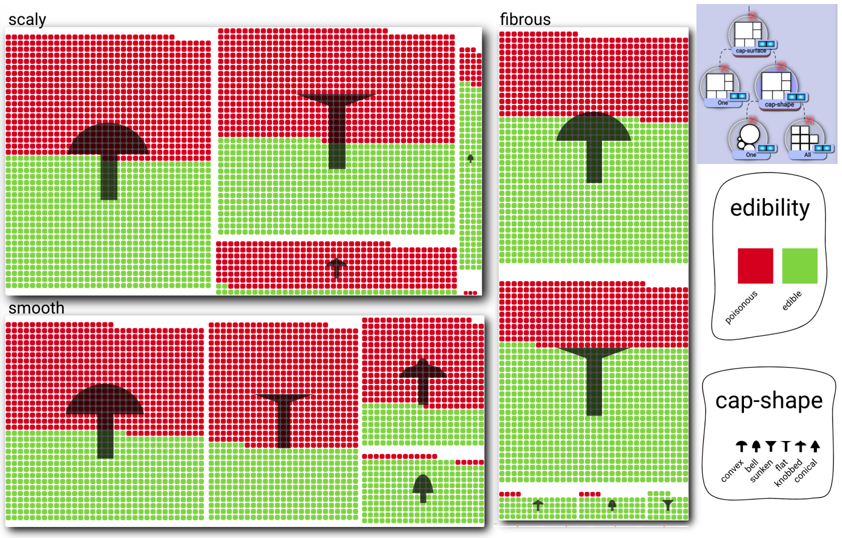


Figure A.10: 8125 samples of mushroom.

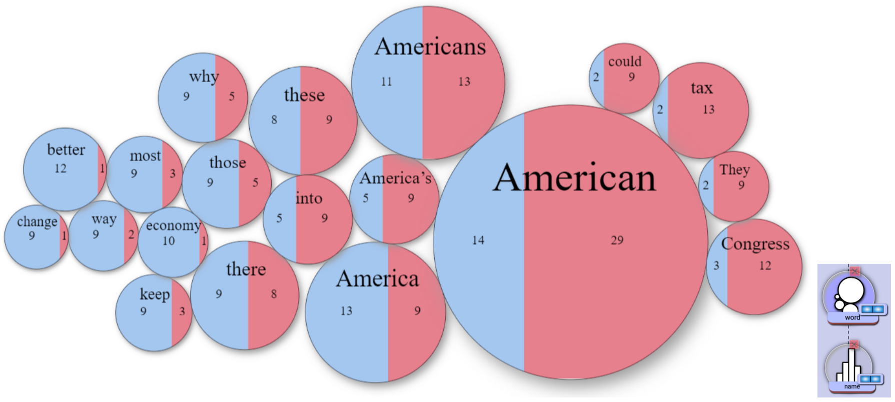


Figure A.11: Recreation of "At the National Conventions, the Words They Used".

Kaggle's **suicide rate** dataset has many dimensions, such as *(country, year, sex, suicides/100k pop, generation, ...)*. In the following two examples, we explore this dataset and demonstrate how nesting can be utilized to generate information-rich small multiples.

For the first example (Figure A.12) we investigate suicide rates per country, by gender over time. First, we create a *unit* chart mapped to *country*. We sort the chart by *avg(suicides)*. Each unit is subdivided by sex using a *rows* chart. Within each *rows* chart, an *area* chart is grouped by *year* (non-sparse), with *avg(suicides)* mapped to *y*, and *year* mapped to *x*. The data mapping of the *area* chart uses the parent datum to show *gender*. The non-sparse grouping of the *area* chart creates empty data points for years with missing data points, thus exposing this in the visualization.

Next, we look at suicide rates for the different generations per country, over time

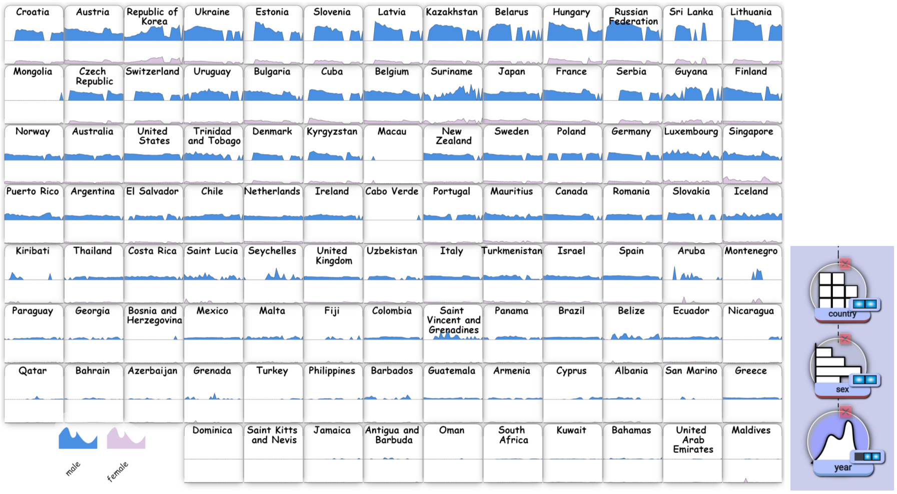


Figure A.12: Suicides per country, by gender, over time.

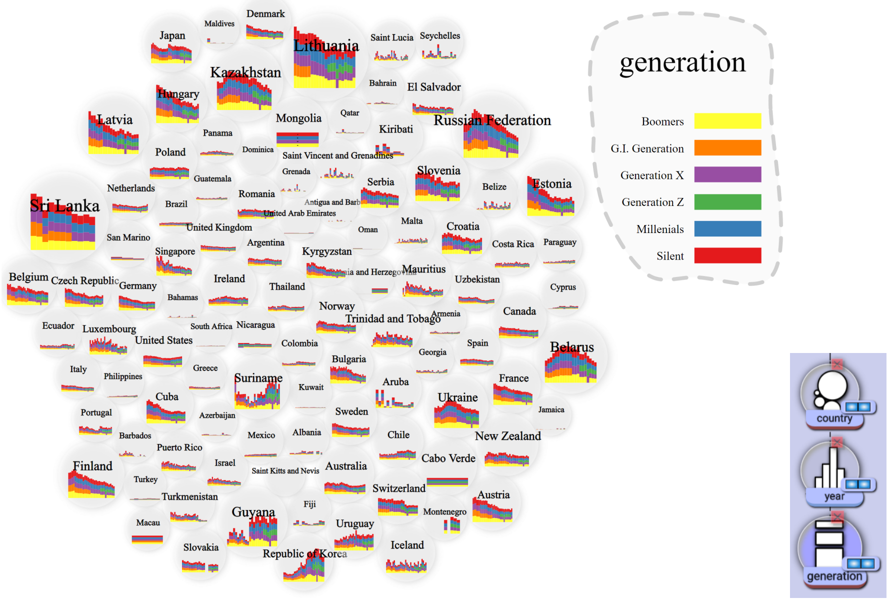


Figure A.13: Suicides per country over time, by generation, over time.

(see Figure A.13). At the root we have a force-directed layout with one node per country. Next, we nest *year* within the root chart, and set the chart type to *columns*. Finally, we subdivide the bars by nesting a *vertical stack* chart (mapped to *generation*) within it. In contrast to Figure A.12, Figure A.13 uses a sparse mapping. We show this example to demonstrate the significance of whether or not empty data items are included in a nested chart.

The Titanic dataset [1] shows how many passengers perished, and how many sur-

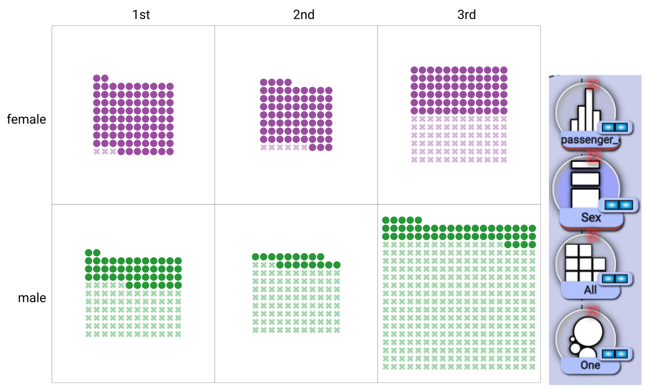


Figure A.14: A recreation of Figure 10 of the ATOM paper [76].

vived. Each row represents a passenger. In the following examples, we demonstrate how the combination of nesting and the *unit* chart enables the visualization of both the global patterns and individual details.

Figure A.14 shows a recreation of Figure 10 of the ATOM paper [76], depicting survivors of the Titanic [1]. Here we see a faceting by sex and class, with a centered *unit* layout. By using nesting, we nest a *plot* within the units, mapping the *symbol* and *opacity* to the *survival* dimension. We use the color mapping to display gender.

Now we wish to investigate the distribution of survivors, by gender and across different age groups. We do this in the form of a "unit stream" as shown in Figure A.15, by age. This chart is created with one column for every age bin, then nesting a *unit* chart within the *columns* chart. The *unit* chart is centered vertically, and ordered by *sex-survival*. Finally, we nest a *plot* within the *unit* chart, and map *fill color* to *gender*, *fill opacity* and *symbol* to *survival status*.

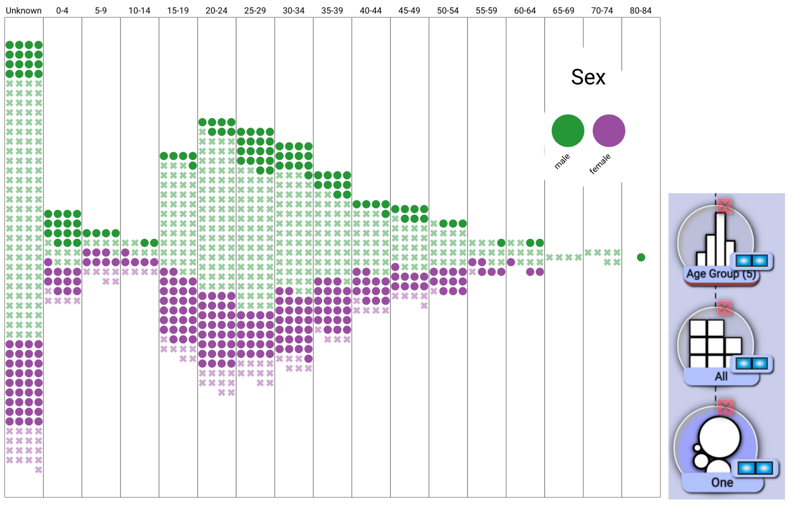


Figure A.15: A "unit stream", by age.

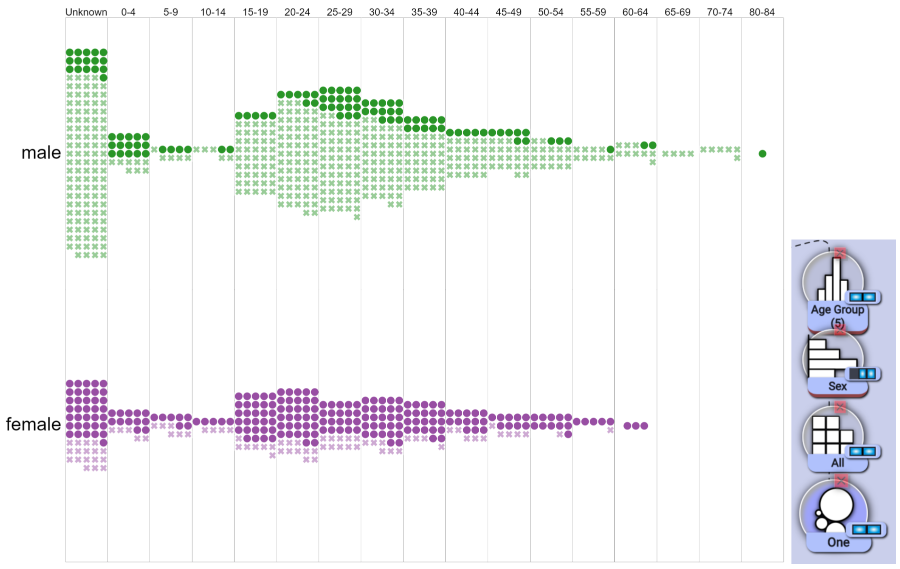


Figure A.16: Two "unit streams", by gender, then age.

Next, we split up the unit stream by gender. The root of the visualization is a *columns* chart, creating one column for every age bin. We subdivide by *sex* by creating a *vertical stack* within each of the age ranges. Then, we nest a *unit* chart within the *vertical stack*. The unit chart is sorted by *survival*. To customize the symbols, we nest a *plot* within it, and map the *symbol* and *opacity* VC-channels to *survival*. The result is shown in Figure A.16. The only difference between this figure and the previous is that there is one *rows* chart inserted into the hierarchy, above *all*, expressing the "group by gender" operation.

The **Best Bookshelf** [53] visualization displays a wide range of dimensions for the book best sellers dataset. Every row in the dataset represents a single book publication and related dimensions such as *(year, genre, title, author, author age, ...)*. We recreate this visualization as shown in Figure A.17 by utilizing nesting, layering and the *identity* mapping. Each square represents a book, with a width representing the number of pages, and the height representing the average rating of the book. Within each square, the proportion of darkened area indicates the age of the author at the time of publication. The data is faceted by creating a *rows* chart mapped to *year*. Within each year, we subdivide by genre with a *columns* chart. Finally, we create a *columns* chart mapped to all (one mark per row) where each column represents one book. The width of the bars is mapped to the *numPages* dimension. To generate the age indication, as well as the best seller star, we nest single marks within each bar using an *identity* mapping. For the age indicator, we map the *age* dimension to the *height* VC-channel. The stars are created with a *plot* with the *symbol* set to star, and *isBestSeller* mapped to *size*, setting it to 0 for False, and a non-zero value for True. We could overload the squares to show more dimensions using the nesting mechanism.

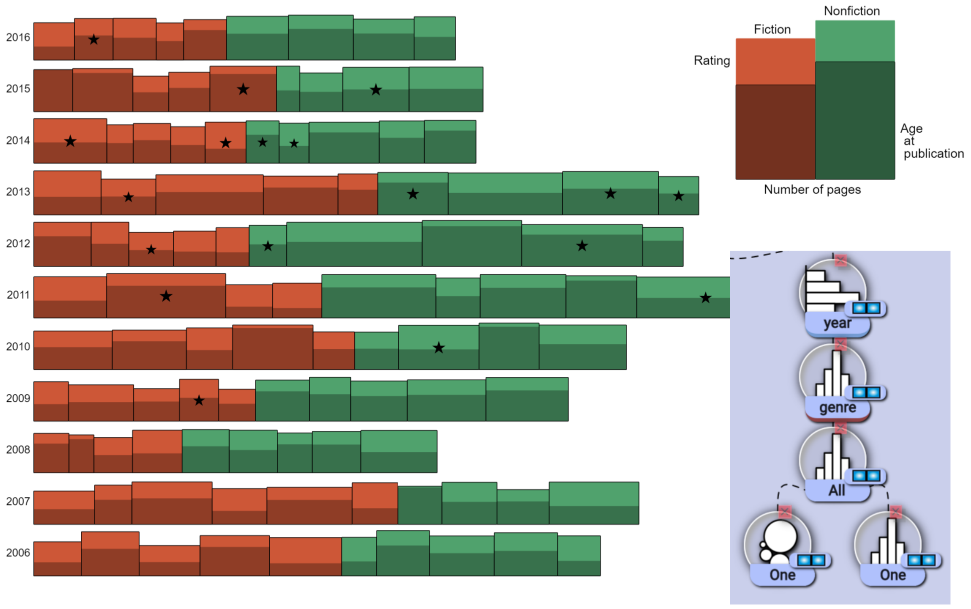


Figure A.17: An approximation of the Best Bookshelf [53] visualization.

FiveThirtyEight's **Gun Crime** dataset [19] contains over 100,000 gun crime incidents from 2012 to 2014. Each incident is represented as a row: *(year, month, intent, age, ...)*. Here we demonstrate deformation behavior, as well as layering and tweaking of a VC-channel to fit a series of labels along a single arc. At the root node we create one circular row for each *intent*. Each row is divided by *sex*, by creating a *columns* chart mapped to *sex*. We then map the aggregate *count* to the *size*. Within each gender subdivision, we subdivide again by *age*, using a *columns* chart. Finally, we nest a *vertical stack* mapped to race. To generate labels along the largest outermost arc, we create a new *identity*-mapped *tubes* chart, nest all ages within it as bars, and fit the *tubes* chart to match up with the largest arc. We do this fitting by tweaking the *start angle* and *width* VC-channels. The resulting visualization is shown in Figure A.18. The *identity*-mapped chart is used to provide a polar space in which the *columns* by *age* are laid out.

The **Cars** dataset is commonly used as a basis for example visualizations of high dimensional data. Each row represents a car and a large number of accompanying dimensions. Here, we demonstrate different chart type nestings, representing the same data hierarchy: *engine-type* $\rightarrow$ *all*. All of these charts can be toggled between by swapping the root chart type. The chart types used as root are the following: *columns, unit, squarified, sectors, tubes*. It would also be possible to change the chart type of the lower node. This allows for interactively exploring and prototyping new designs. Some example charts are shown in Figure A.19.

Axes are available when a chart has its axis VC-channel mapped to data value. Here, we aim to demonstrate that axes are available for non-nested, nested, Cartesian and non-Cartesian layouts. Figure A.20 shows a plot with *fare* mapped to *x*, and *age* mapped to *y*, both for all entries, as well as for every *class-gender* permutation. The

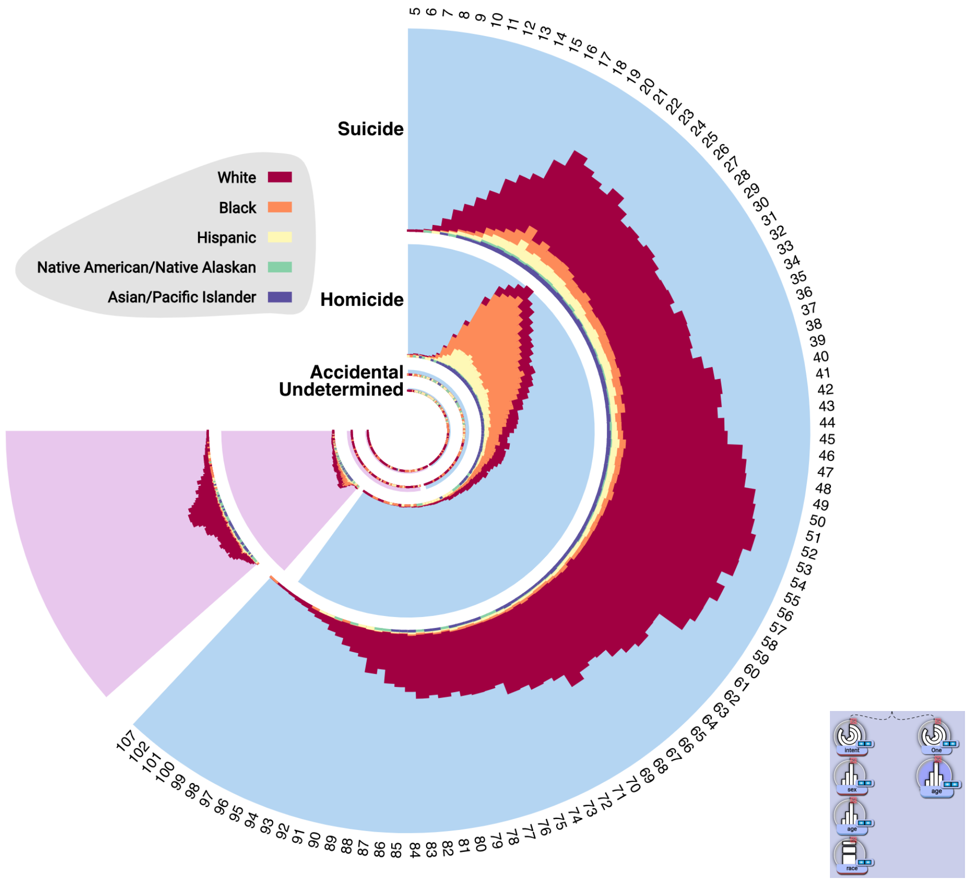


Figure A.18: Gun crime broken down by *intent, gender, race*.

axes are deformable, are thus nestable and flexible similarly to the charts themselves.

These examples show a range of different expressions that can be achieved via nesting. We have demonstrated implications and uses of different kinds of data mappings and charts used in combination. The *identity* mapping allows for overloading charts with more information and the use of charts as containers for other charts. By combining custom mappings and nesting, a great variety of visualizations can be expressed.

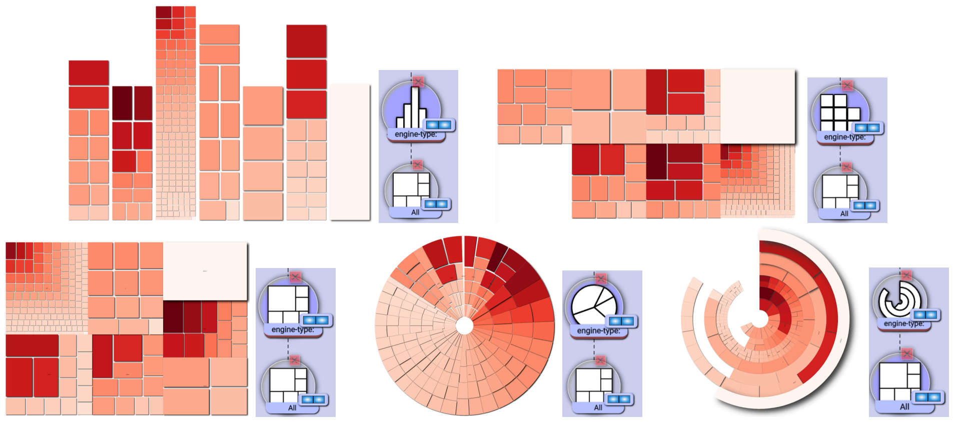


Figure A.19: Variations of the data hierarchy *engine-type* → *all*. Swapping between these variations only requires the user to change the chart type. Mapped VC-channels are transferred, thus the style is transferred.

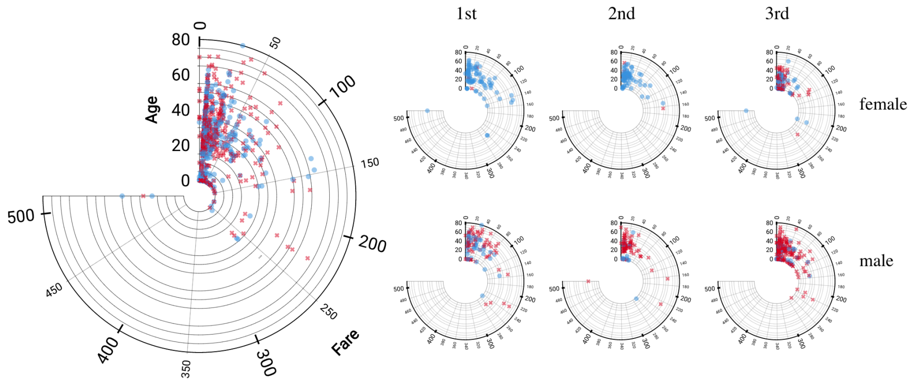


Figure A.20: Titanic survivors, faceted by class and gender, showing a polar plot with *Age* mapped to *y*, and *Fare* mapped to *x* for every category.

A.6 Discussion and Limitations

Comparison to other visual builders: Satyanarayan et al. [86] recently proposed a set of criteria to evaluate visual builders, and compare the three most feature rich, recent works: Data Illustrator [61], Charticulator [82] and Lyra [85]. Visception focuses on achieving expressiveness by nesting charts. We aim to show that features such as glyph composition, coordinate systems and data scoping can all be expressed by leveraging nesting functionality. Table A.6 summarizes the Visception visual builder in the terms proposed by Satyanarayan et al. and is meant to be compared with Table 1 in their paper. By comparing Visception to the other systems in this manner, it can be seen that Visception achieves many features via nesting and the accompanying data grouping.

Framework: While Visception provides a number of standard charts, there are several types of more complex or specialized types of visualizations that are currently not integrated. Our implementation is designed with tabular data in mind. We sup-

Component	Visception
Marks Instantiation and Customization	W: Predefined marks only, referred to as chart type H: Start with default, click chart in outline view to change. Must choose data grouping before seeing marks.
Glyph Composition	H: Combine predefined marks into glyphs as layers. Such layers can be nested within existing charts, enabling a wide range of combinations and mappings.
Path Points and Path Segments	W: Map x and y to data dimensions. H: Drag data dimension or aggregate to the x or y VC-channel of the chart, if the chart type is *line*, *area* or *stream*.
Links between Glyphs	W: Limited availability. H: For example, a line chart can be layered under a plot, with identical x/y data mappings linking the glyphs. However, for future work we aim to introduce a linking tool similar to that of Charticulator
Data Scoping for Glyphs	W: Custom dimensions and groupings: *all*, *identity*, *monolith*, sparse and non-sparse grouping modes. H: An *all* dimension to create one mark per tuple, a *identity* dimension to create a single mark representing the data selection of the parent node (if root, the entire dataset). Grouping by a dimension implicitly aggregates the data by that dimension. Groupings by a dimension are by default sparse, i.e they will not show empty marks when nesting. If non-sparse, empty data is created to populate each nested viewport. To create one glyph per dimension, use the *Monolith* grouping of a numeric dimension.
Mapping Data Values to Visual Properties	H: Drag dimension or aggregate from data view and drop on channel, or select from menu. Available mappings depend on the data scope of the selected chart.
Scales	W: Scales for categorical, temporal, and numerical data H: Implicitly created when mapping data to channels (visual properties)
Axes and Legends	H: Created when a data binding is applied. Hidden by default if chart is nested, except for color mappings. Each legend/axis is customized with channels, the same way a chart is customized.
Relative Layout	H: Use the *Bounds* channel to free-form position a chart in normalized space.
Layout in a Collection	H: Marks are always positioned according to the layout of the selected chart type. Each chart has a set of visual channels, and in most cases a set of *layout* channels, some mappable to data (for example the x and y position of a line chart).
Nested Layout	H: If the chart is nestable (appropriate data and chart type), another chart may be nested within it. Since separate aspects of layouts can be mapped to any eligible data dimension or aggregate, this implicitly changes the space in which the nesting is done. With the *bounds* channel we can edit the bounds of a chart in a normalized space. If the parent space is deformed (i.e an arc) the geometry of the child chart is deformed accordingly (for example, a square to an arc).
Coordinate Systems	W: Cartesian, Polar, extensible to others. H: Each chart is seen as a set of 2D shapes, these shapes are simply transformed to fit within the given parent shape. As such, an arc can deform a rectangle to fit within itself. Each chart type must specify *how* it is to be deformed.

Table A.6: (W=what, H=how) Summary of the Visception visual builder system components. To compare to Lyra, Data Illustrator and Charticulator we recommend the user to view this table next to the table presented in the work by Satyanarayan et al. [86]. Visception achieves many of these features through the use of nesting, while in other systems these features are more explicitly specified. Furthermore, in Visception the data scoping of a chart is implicitly defined by the data mapping of the chart, abstracting the specifics of this task away from the user.

port categorical and numerical data, but currently do not provide specific operations for specifying categorical dimensions as ordinals, as well as specialized aggregations for time-oriented data. We also do not provide explicit support for visualizations targeted at graph and network data such as node-link diagrams, and some other common visualization techniques such as parallel coordinates or parallel sets are also currently not implemented. However, we believe that they fit well within our architecture and plan to add these and other relevant chart types in the future. Links and bands between marks of different charts should also be possible to add to the framework, but it proved difficult to find ways to make bands and links work across different levels of nesting, especially given the nature of SVG group hierarchies. The challenge of increasing expressiveness is not in adding the charts themselves, but in adding general structures to support different kinds of charts so that they can leverage the existing nesting behavior.

Rendering and layout calculation: When a chart is fully reflowed, its layout is calculated before it is applied to the corresponding SVG paths. With nesting introduced, it is crucial to only apply the necessary updates to the chart and its child charts. For example, editing the *fill color* of a chart should not cause a reflow of its children. Redundant reflows break the fluidity of the interaction. We use throttling to keep the system responsive, but additional threading could further improve the situation. We rarely encountered performance problems with the SVG rendering itself, except when filter effects like drop shadows are active. This is to be expected, though it would be beneficial to disable filters effects upon zooming and interaction. Specifying which step of the pipeline a VC-channel should trigger has removed a great number of redundant full reflows. Furthermore, we noticed that complex nested visualizations expose some deficiencies in SVG support across different platforms and applications. This is mainly due to numeric instability arising from deeply nested SVG elements. The examples in this paper are screenshots taken in FireFox (version 72), which has not shown these issues.

Data querying: If the dataset is too large, there are potential performance concerns with regards to both data querying, and rendering of the chart itself. For example, the suicide dataset had about 100,500 rows, and the aggregation at the deepest level *(intent, sex, age, race)* took about 3 seconds to compute on a 2.2 GHz Quad-Core Intel Core i7 with 16GB memory. The data querying issue could be resolved by using a server for queries. However it is always preferable that the program can be used without a server. Currently, we lazily compute aggregations as well as their domains when querying the data. Whenever an aggregate is retrieved for the first time, all aggregates for that column are computed and cached. We used arrow.js to store the data in a columnar format, and a recursive data structure to generate queries for each VC-node. Taking a progressive visualization [7] approach might help in addressing this.

Visual builder user interface: The outline view tree has scalability issues if the hierarchy of trees gets too wide or too deep. To counter this, the outline view (and other windows) can be made into a floating window. However, for future work we would like to incorporate more scalable techniques for showing this.

A.7 Conclusion

In this paper we, presented our framework for nested visualization design. We introduced the VC-tree as a unified framework for the creation and manipulation of nested visualizations and demonstrated how it can be used to flexibly specify a wide variety of data groupings and visual mappings. We showed how the VC-tree provides fine-grained control over data mappings at different hierarchical levels, while providing implicit handling of deformation and nesting behavior. Based on our framework, we contributed a visual builder that exposes the full expressiveness of the framework through a user interface. To demonstrate the expressiveness of our approach, we provided a wide range of examples demonstrating various features achievable via nesting.

Acknowledgments

The research presented in this paper was supported by the MetaVis project (#250133) funded by the Research Council of Norway.

A.8 (Appendix) Overview of Charts and VC-channels

We present the full set of VC-channels in the current implementation of Visception in Figures A.21 and A.22. Figure A.21 shows all VC-channels that are common to multiple charts, while Figure A.22 shows all chart types and the VC-channels unique to each type.

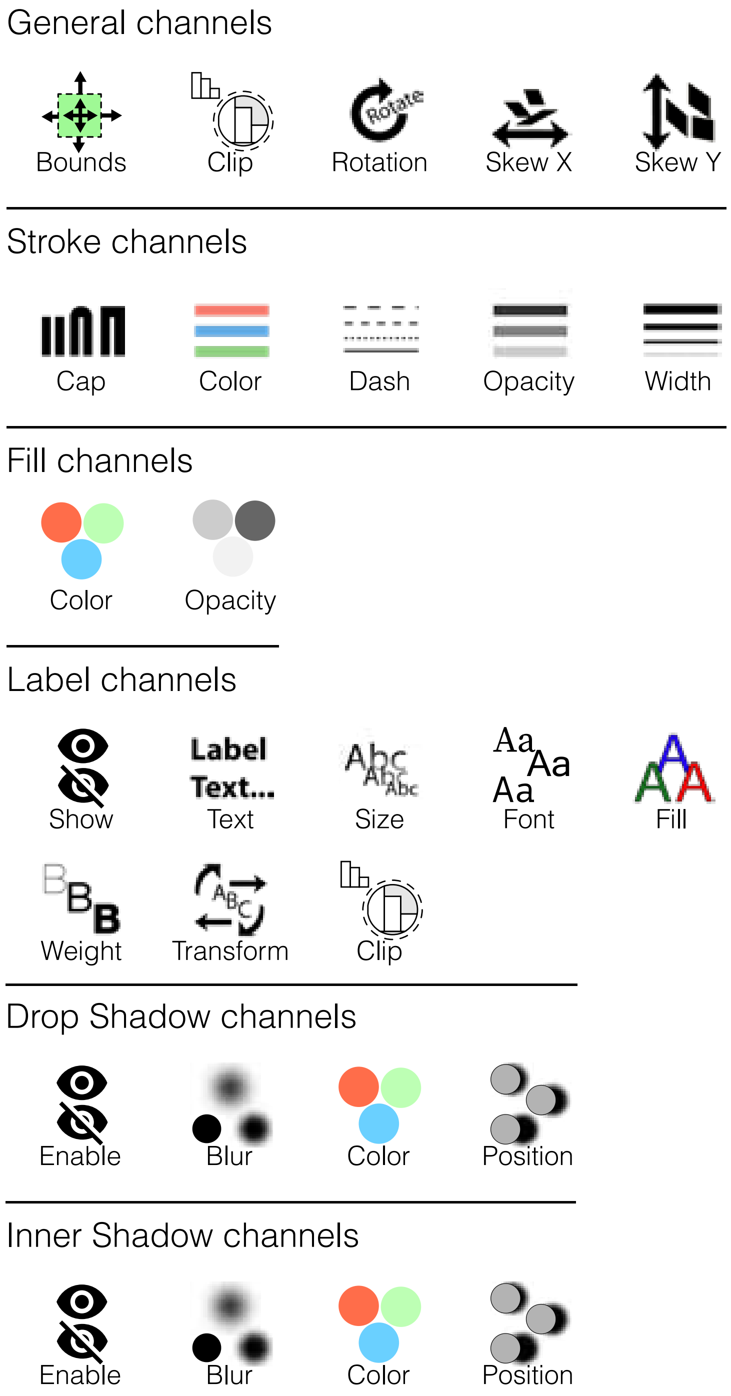


Figure A.21: All general VC-channels within Visception. These exist for all charts, with the exception of label channels not existing for *area* and *line* charts, and fill VC-channels not existing for line charts.

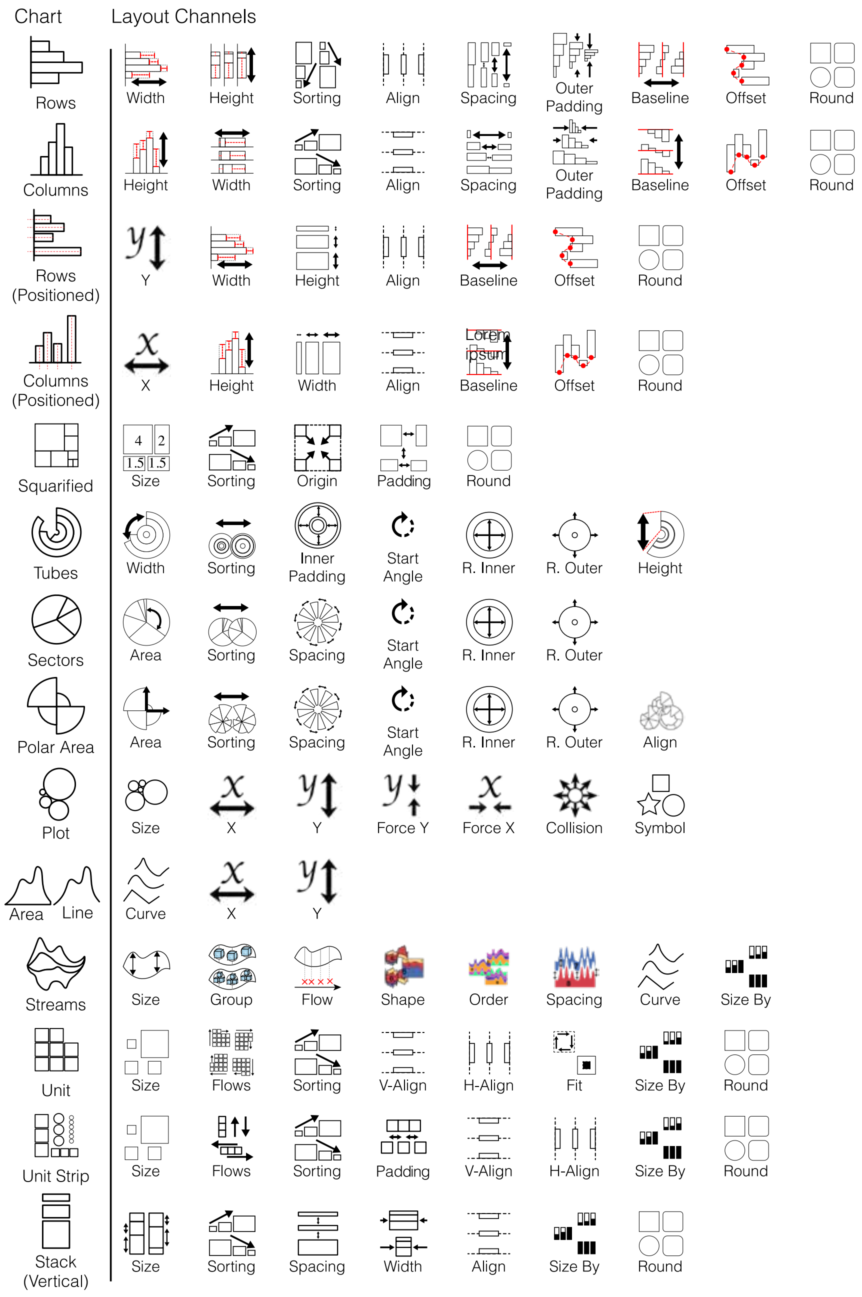


Figure A.22: All chart types, and VC-channels unique to that chart type within the current implementation of Visception. Each icon represents a channel, and each VC-channel controls the layout or some property unique to that chart, or charts with similar output marks.

A

Paper B

Semantic Snapping for Guided Multi-View Visualization Design

Yngve Sekse Kristiansen, Laura Garrison, and Stefan Bruckner

University of Bergen, Norway

Abstract

Visual information displays are typically composed of multiple visualizations that are used to facilitate an understanding of the underlying data. A common example are dashboards, which are frequently used in domains such as finance, process monitoring and business intelligence. However, users may not be aware of existing guidelines and lack expert design knowledge when composing such multi-view visualizations. In this paper, we present semantic snapping, an approach to help non-expert users design effective multi-view visualizations from sets of pre-existing views. When a particular view is placed on a canvas, it is "aligned" with the remaining views–not with respect to its geometric layout, but based on aspects of the visual encoding itself, such as how data dimensions are mapped to channels. Our method uses an on-the-fly procedure to detect and suggest resolutions for conflicting, misleading, or ambiguous designs, as well as to provide suggestions for alternative presentations. With this approach, users can be guided to avoid common pitfalls encountered when composing visualizations. Our provided examples and case studies demonstrate the usefulness and validity of our approach.

This article was published in *IEEE Transactions on Visualization and Computer Graphics vol. 92, no. 1*, pages 43–53, 2021. (doi: https://doi.org/10.1109/TVCG.2021.3114860)

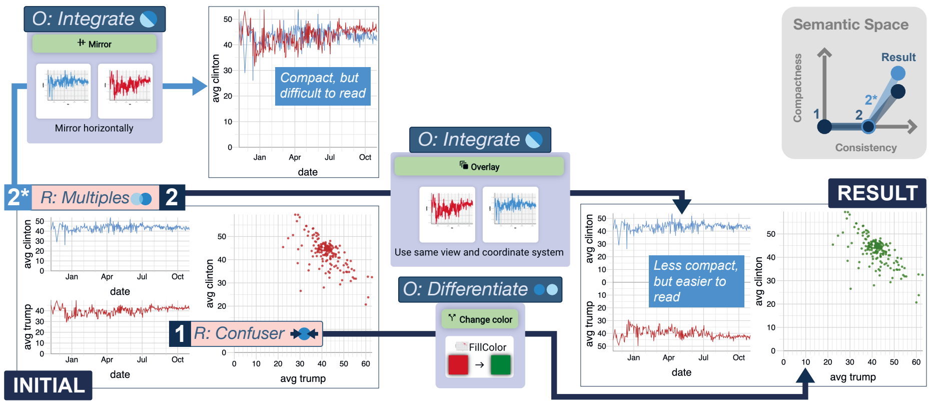


Figure B.1: Semantic snapping allows the user to perform iterative operations to improve the compactness and consistency of a multi-view visualization. Underlying algebraic rules called *relations* define the available *operations* for each iteration. In this example showing the 2016 US Election poll percentages and pollsters, from our initial composition we (1) identify a *confuser* relation between the bottom left and rightmost views showing the same color (red). We *differentiate* these views by selecting green as the fill color for the scatter plot. We next identify a *multiples* relation in the two left views. We resolve this through one of two *integration* operations. (2*) *Overlay* produces an unsatisfactory result, so we revert and (2) perform a *mirroring* operation to arrive at our resulting composition. The semantic map to the right illustrates our path through semantic space.

B.1 Introduction

Multi-view visualizations are frequently utilized to present and analyze data. Dashboards, for example, are commonly employed for monitoring and related tasks in a wide variety of fields. Popular visualization systems like Tableau [97] and PowerBI provide galleries of carefully crafted templates in order to enable the quick and easy generation of such visualizations. However, when non-expert users would like to extend, modify, or customize such a multi-view visualization, they may easily fall prey to a number of pitfalls that can result in potentially misleading or otherwise problematic results. Expert knowledge to guide such tasks is mostly available in the form of guidelines from the visualization literature, which are not readily accessible to novice users. Common examples include Qu and Hullman's constraints, C1 (the same data should be shown the same way) and C2 (different data should be shown in different ways) [79]. In this paper, we present a method that detects and helps users to resolve such potential problems in multi-view visualization design in a semi-automatic and guided fashion.

Suppose a user of a visualization system wants to create or extend a multi-view visualization from a set of pre-existing charts that are individually well-designed (e.g., based on a gallery). If the user wishes to use these visualizations in combination (e.g., in a dashboard), there are non-obvious design opportunities and pitfalls. Views may show the same data in different ways, or different data in the same way. A single view may be highly informative and take up a modest amount of screen space. However, when used in combination with other views, they may show overlapping information, or use too much screen space. These issues can be remedied by showing the same data with fewer views, i.e., making the overall design more *compact*. A multi-view visualization can be made more compact, or less conflicting, by manually redesigning and tweaking single views. However, manually detecting and resolving conflicts, and coming up with alternate representations of views, is cumbersome, error-prone and time-consuming. Our method lets the user perform this process via high-level design-altering operations.

Our approach uses a semantic space with two axes that represent the degree of consistency and compactness of a multi-view visualization. We examine *relations* between views to identify opportunities to improve the overall visualization with respect to these criteria. We then provide the user with a set of operations to realize the corresponding changes. For instance, two views may employ the same color map for different quantities. In such a case, our approach may suggest to *differentiate* the two views by modifying one of the mappings to increase the overall consistency. Likewise, when the same data are shown differently in multiple views, our method suggests different ways to *homogenize* them. In other cases it may be possible to *integrate* multiple views in order to improve the compactness of the visualization.

The contributions of our work can be summarized as follows. Based on a synthesis of existing guidelines from the literature, we present a novel approach for identifying and applying potential improvements of multi-view visualizations. We use predicate logic to represent relations between individual views and propose operations to improve the consistency and compactness of the underlying visualization based on the identified relations. Furthermore, we propose a simple workflow and user interface for presenting and selecting the suggested operations.

B.2 Related Work

Visualization design and measures. Bertin's Semiology of Graphics [10] and Wilkinson's Grammar of Graphics [108] were two of the early influential works focusing on formal aspects of reasoning about the effectiveness of visualizations. Munzner [73] later consolidated and refined existing concepts and terminology, leading to a comprehensive framework for thinking about visualization in terms of principles and design choices.

Bolte & Bruckner [11] survey measures focusing on different aspects of the visualization process: perceptual characteristics, task-oriented quality measures, structure-oriented measures, and meta-perceptual processes. Perceptual characteristics such as Cleveland & McGill's experiments on graphical perception [22] are based on human performance in elementary tasks such as comparing positions on a common scale. Other measures express desirable relationships between the data and its visual representations. For example, Tufte's data-to-ink ratio [102] describes the proportion of pixels used to represent data versus the total amount of available pixels. Furthermore, Correll et al. [23] address the issue that designs may appear to be showing the data completely, while hiding important details. They propose actions to remedy discovered vulnerabilities for different chart types. Behrisch et al. [9] categorized different quality measures from around 250 papers. Most of these measures were specific to a certain combination of underlying data, task and visualization technique.

Through literature review, Zhu [119] points out why existing definitions of visualization effectiveness are often incomplete: they usually take either a data-centric, or task-centric view on what an effective visualization is.

Data-centric effectiveness measures deal with how accurately a visualization is showing its underlying data. An example of a data-centric framework for measuring visualization effectiveness is Kindlmann and Scheidegger's algebraic framework [54]. By considering symmetries between changes in data space and resulting changes in visualization space, they describe three principles that should ideally be true for any data-to-visualization mapping: unambiguous data depiction, representation invariance, and visualization-data correspondence. We draw inspiration from this model and adopt a similar line of reasoning in the context of multi-view visualizations. Based in part on the concept of algebraic visualization design, McNutt and Kindlmann [67] present a linting mechanism for the process of designing a chart. Their linting is realized as a Python library that evaluates visualizations created with matplotlib, and returns a list of rules that are violated. While our work is based on similar fundamental considerations, we focus on multi-view visualizations and expose potential revisions through a user interface.

Many approaches take into account both data and tasks. Cantu et al. [18] outline an approach to identify relationships between visualization challenges and representation components (e.g., data transformations, filtering techniques, visual variables). They argue that these relationships can further our understanding of the mechanisms behind visualization components, which could eventually be used to build visualization recommendation tools. Silva et al. [94] survey work done on using different color scales in visualization, with a focus on desired properties and guidelines for choosing the right colors. They highlight that it is important to consider factors such as the type of data, type of visualization, type of task, and audience.

As pointed out by Zhu [119], there are multiple disjoint, sometimes conflicting sets of guidelines and measures. Efforts have been made to facilitate convergence and understanding between different viewpoints. Diehl et al. [27] initiated the VisGuides forum both to facilitate collection and discussion of visualization guidelines, and knowledge about visualization in general. Engelke et al. [30] highlight that there is a gap between the communities who propose visualization guidelines, and those who need them. They provide a conceptual model called VISupply that highlights problems and opportunities with how guidelines are currently "shipped" to non-experts.

Authoring tools and visualization recommendation. Visualization authoring tools help users creatively express a wide range of individual charts. While these systems have much design freedom, they also rely on the expertise of the user. Zhu et al. [118] survey different tools for automatically generating infographics and visualization recommendations.

Examples of systems that mostly focus on authoring and design flexibility include Charticulator [82], Lyra [85], iVisDesigner [81], Data Illustrator [61], Data Driven Guides [52]. These systems all use varying underlying frameworks for representing visualizations. We provide a set of relations and operations specified at a high enough level so that they can be expressed in terms of most individual frameworks.

Several efforts have been made to make expert knowledge available through software. Among them, visualization recommendation systems can potentially take into account expert knowledge to steer which revised designs are presented to the user.

MacKinlay's APT (A Presentation Tool) [64] was among the first of these systems. He used a composition algebra for designing visualizations, and evaluated their effectiveness in accordance with Cleveland & McGill's effectiveness metrics [22]. Wongsuphasawat et al. proposed CompassQL [111] as a general language for querying over the space of visualizations, to be used in visualization recommender systems. Voyager [110] allows for exploring data via automatically generated visualizations. With Voyager 2 [113], the user is able to partially specify what a view should show by using wildcards and also see automatically-generated charts showing data related to the existing views. Data2Vis [26] is a trainable neural translation model for automatically generating visualizations from datasets. It is powered by formulating visualization generation as a language translation problem, where data specifications are mapped to Vega-Lite specifications [87]. Grammel et al. [36] explore how novices construct visualizations. Their findings suggest the need for a tool that supports iterative refinements, and explanations that help with learning. Our method shares a similar line of thought by enabling incremental refinement of a multi-view visualization. Show Me [65] is a set of user interface commands that provide a way to display an additional data attribute within a view, as well as high-level commands for building views for multiple fields. Draco [72] makes visualization design guidelines available for a wider audience by formalizing the knowledge into precise constraints, which can then be used and accessed in an Answer Set Programming environment. They model single visualizations as sets of logical facts, and represent design guidelines as hard and soft constraints over these facts. Dziban [60] further extends Draco with anchoring mechanisms to help drive specification queries with increased user agency. These works all represent different ways of representing and reasoning about visualizations. Our method differs from these approaches in that it focuses in the incremental refinement of multi-view visualizations.

Multi-view visualization design. One of the most common use-cases of multi-view visualizations are dashboards.

Sarikaya et al. [84] construct a design space of dashboards, by analyzing multiple examples of dashboards found "in the wild."

QualDash [29] is a task-oriented dashboard generation engine that enables the mapping of specific user task sequences in healthcare quality improvement to a view composition.

For dashboards and multi-view visualizations in general, multiple views must be laid out on a single screen, or even multiple screens. PanoramicData [116] is a visual analysis tool using a canvas metaphor to explore and combine data views. We use a similar metaphor in our approach, although we focus on semantics rather than filtering and linking the views. Vistribute [42] is a framework that automatically distributes visualizations and user interface components among multiple heterogenous devices. Scout [99] is a system that helps interface designers to create layouts by using high-level constraints based on design concepts such as semantic structure, emphasis and order. While our approach currently does not address layout, we believe that our method could be combined with similar approaches to also take into account layout considerations.

Composed views such as small multiples [101] allow for comparing visualizations. Gleicher et al. [35] provide a general taxonomy of visual designs for comparing visualizations, with three categories: juxtaposition, superposition and explicit representation of relationships. Elzen and van Wijk [104] leverage small multiples so that they are not only informative, but also helpful for the data exploration process itself.

Through a series of graphical perception experiments, Ondov et al. [75] investigated which compositions of multiple charts are the most effective for different tasks. From 360 images of multi-view visualizations collected from IEEE VIS, EuroVis and PacificVis publications from 2011 to 2019, Chen et al. [20] identify common multi-view visualization practices, including typical view layouts, view types, and correlations between view types and layouts. The patterns found among these views are made available through a multi-view visualization recommendation system, allowing users to interactively browse different designs. We draw inspiration from these approaches by enabling the transformation of, for example, two bar charts into an item-wise grouped or chart-wise juxtaposed mirrored bar chart in order to increase the compactness of the overall visualization, as in Figure B.5.

Conventional snapping creates a “gravity field" around geometric objects, making it easier to place them together in certain ways. Hudson [43] introduced the notion of *semantic snapping* as an interaction technique for geometrically snapping objects together only if the objects are specified to be semantically related. Our work is a continuation of this basic concept, extending it to the scenario of multi-view visualization design and focusing on the semantic rather than geometric aspects.

Shadoan & Weaver [93] explore semantic relations in multi-view visualizations using a hypergraph querying system. While such queries are constructed similarly to *relations* in our approach, the former are driven through cross-filtering on attribute relationship graphs, while ours draws from rules heavily inspired by Kosslyn’s principles [55] and Kindlmann and Scheidegger’s algebraic framework [54]. The latter framework has been used to identify effective visualization types for certain user tasks, e.g., table cartograms [66]. Kim et al. characterize responsive visualization strategies via their targets, i.e., element(s) of a design that change, and actions, i.e., how element(s) are changed [51]. This semantics-based characterization parallels our notion of *relations* and *operations*, although the underlying models differ.

Qu and Hullman [79] discuss how to operationalize Kosslyn’s principles [55] with the two following constraints: C1 (encode the same data in the same way), and C2 (encode different data in different ways). These two constraints are further detailed by specifying lower-level constraints on encodings across two views. In a later paper [80], they found through a Wizard-of-Oz study that Tableau users unknowingly, and with some exceptions, respected their constraints C1 and C2. They found that study participants were positive to having a consistency checker tool to surface such warnings. Similarly to Qu and Hullman, we operationalize the principles C1 and C2 on an encoding-level, but we do so by using a model inspired by Kindlmann and Scheidegger’s algebraic framework [54]. Furthermore, we present a practical realization of this concept that both shows how to identify potentially problematic relations and introduces a set of concrete operations to address the relations, i.e., remove the relation itself or a problem caused by the relation.

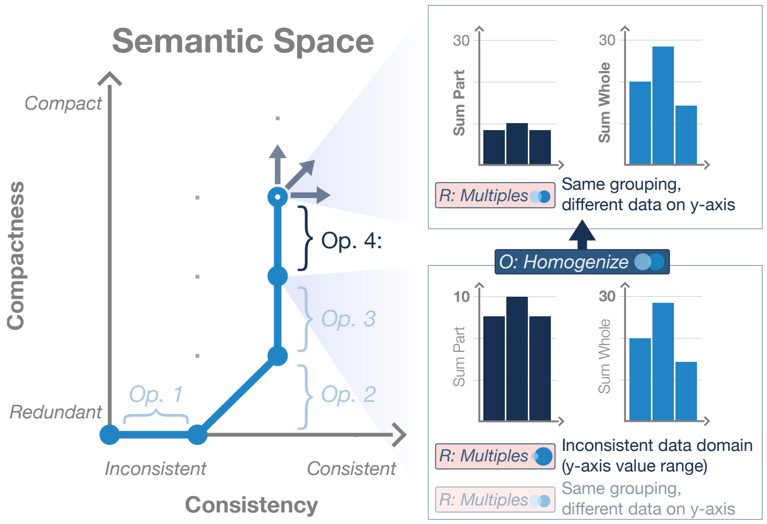


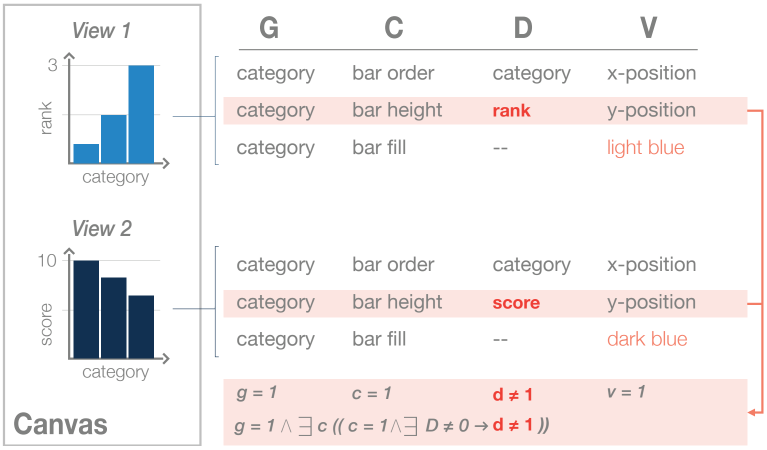
Figure B.2: Conceptual figure showing the semantic space with relations and operations for our semantic model. Operation 4 displays the *homogenize* operation which is available as a result of a *multiples* relation (the two axis scales are different, but should be the same if the underlying data represents the same quantity).

B.3 Semantic Snapping Model

Semantic snapping is the process of incrementally modifying a multi-view visualization by aligning its individual views with respect to their semantic, rather than their geometric, attributes. The main underpinning of our method is that a multi-view visualization, and its potential revisions, can be placed into a semantic space with two dimensions representing the degree of compactness and degree of consistency. We provide a conceptual overview of this semantic space in Figure B.2. In other words, a potential revision of a design is either more or less compact, or more or less consistent, than the original design. Our method identifies these potential revisions, and presents them to the user as operations. By executing these operations, the user is able to intuitively navigate the semantic space of revised designs.

Achieving high-level goals by piecing together low-level modifications can be tedious, especially for novice users. In other software, such as word processors, semi-automatic tools help with this workload by highlighting errors, and suggesting corrections to these errors. Previous approaches, such as McNutt and Kindlmann's linting mechanism [67], have already explored this direction by providing functionality akin to a spell checker for a *single* visualization. In contrast, semantic snapping can be seen as more similar to a grammar checker, since it focuses on relationships *between* visualizations, just as a grammar checker analyzes relationships among words or phrases. Errors or potential errors represent detected inconsistencies or redundancies between views, and error corrections are represented as suggested operations to revise the composition of views.

Our method identifies existing and potential semantic inconsistencies and redundancies, so-called *relations*, between single views. Each relation identifies a potential



G	C	D	V
category	bar order	category	x-position
category	bar height	**rank**	y-position
category	bar fill	--	light blue
category	bar order	category	x-position
category	bar height	**score**	y-position
category	bar fill	--	dark blue
$g = 1$	$c = 1$	$d \neq 1$	$v = 1$

$g = 1 \wedge \exists c ((c = 1 \wedge \exists D \neq 0 \rightarrow d \neq 1))$

Figure B.3: Here we see two views, and the (G, C, D, V) tuples corresponding to the three channels representing the y-axis (bar height), x-axis (bar order), and fill color. The comparison of the two highlighted tuples are what identifies these two views as *multiples (1) same grouping*.

problem which can be resolved by an *operation*. Thus, each operation is a high-level modification to the overall design. It is necessary to have the user involved in each modification to the design, since consistency and compactness are sometimes traded off for other design considerations [80]. The cycle of finding relations to infer available operations is repeated every time the design is altered.

B.3.1 Semantic Space

We begin with defining the terms that comprise our semantic space. A *canvas* is composed of multiple views of a single tabular dataset, where each individual view displays a single chart of a certain type (for example a bar chart, scatter plot, line chart, etc.). A *chart grouping* is a data dimension by which a chart is grouped, similar to SQL's **GROUP BY** command. For example, a bar chart grouped by country will have one bar for each distinct country in the dataset. Each chart has a set of *channels*, which may or may not be mapped to data. Data shown by a channel is denoted as a *data mapping*, which may also be empty (indicating an unmapped channel). For example, consider the fill color channel in a scatter plot, which can optionally be mapped to data to display an additional quantitative data attribute for each mark. When a channel has a data mapping, the data are scaled from a *data domain* to a resulting *visual output* which directly or indirectly affects the appearance of the chart. The data domain denotes the minimum and maximum value of a certain attribute, or attribute aggregate, and is used as an input for the scale from data to a resulting visual output. When groupings differ, sampled data domains may also differ. Furthermore, domains may be different due to custom configuration of individual views. While there are different ways to arrange and transform tabular data, we limit the scope of chart groupings and channel mappings to single data dimensions.

Summarized, a view has one chart grouping and multiple channels. Each channel

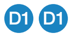
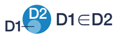
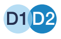
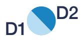
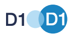
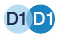
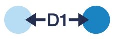
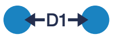
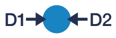
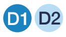
	Relation	Specification	Illustration	Possible Operations	Illustration
(a)	Full redundancy	$g = 1 \wedge \forall c(c = 1 \rightarrow d = 1)$		Delete one	
(b)	Partial redundancy	$g = 1 \wedge \forall c((c = 1 \wedge d \neq 1) \rightarrow \exists! D = 0)$		Integrate, or delete D1 view	
(c)	Multiples (1) same grouping	$g = 1 \wedge \exists c((c = 1 \wedge \exists D \neq 0) \rightarrow d \neq 1)$		Integrate or homogenize	
(d)	Multiples (2) same data	$g \neq 1 \wedge \exists c((c = 1 \wedge \exists D \neq 0) \rightarrow d = 1)$		Homogenize	
(e)	Hallucinator	$g = 1 \wedge \exists c(c = 1 \wedge d = 1 \wedge \exists D \neq 0 \wedge v \neq 1)$		Homogenize	
(f)	Confuser	$\exists c(c = 1 \wedge d \neq 1 \wedge v = 1)$		Differentiate	

Table B.1: All relations specified in terms of our model. The lower case letters: g, c, d, and v represent equalities (1) or inequalities (0) between chart groupings, channels, data mappings and visual outputs, and the specifications are predicate logic expressions operating primarily on these (lower case) equalities or inequalities. The uniqueness quantifier on $\exists! D$ indicates that there exists exactly one data mapping that satisfies a certain condition, for example being unmapped ($= 0$) for (b). For the *partial redundancy* relation, $D1 \in D2$ signifies that all data shown by one view ($D1$) is also shown by the other view ($D2$).

has a data mapping, a data domain (if the data mapping is non-empty), and a resulting visual output. A view is part of a canvas, and a canvas has a certain position in the semantic space.

The semantic space has two axes, representing (1) redundancy/compactness (shorthand: compactness axis) and (2) inconsistency/consistency (shorthand: consistency axis). The compactness axis ranges from redundant to compact, whereas the consistency axis ranges from inconsistent to consistent. For example, if a canvas is made more compact by turning two views into one, the canvas becomes less redundant, thus moving up along the compactness axis. Semantic snapping corresponds to movement along one of these semantic axes.

It is important to note that more consistency and compactness is not always desirable. For example, Qu and Hullman [80] found that in cases, homogenizing axis domains is undesirable due to the extra white space it generates. Conversely, a compact design is not always more readable, or the most ideal for telling a story. Our operations make it possible for the designer to explore this space of alternative designs more rapidly, one semantic axis at a time.

Relations between individual views identify not only where the canvas is currently located, but also which changes (denoted operations) are possible.

B.3.2 Algebraic Relations

Relations are explicit specifications of redundancies or inconsistencies between views. Although relations themselves do not indicate whether a design is good or bad, they are available to help the user identify potential problems in their overall visual design.

Our specification of relations draws both from Qu and Hullman's evaluation constraints [79] and a generalization of the principles established in Kindlmann & Scheidegger's algebraic model of visual design [54]. Originally developed in the context of only a single visualization, their model describes the relationships between three elements of the visualization process: the data, the representation of the data, and the resulting visualization. We adopt two principles from this model, which are easily framed within the two high-level constraints stated by Qu and Hullman [79]:

C1 Encode the same data in the same way. A violation of this constraint corresponds to representation invariance in Kindlmann & Scheidegger's algebraic model, which states: if the data of two visualizations are the same, the resulting visualizations should also be the same. A violation of this is called a *hallucinator* (Table B.1e).
C2 Encode different data in different ways. A violation of this constraint has a corollary again in Kindlmann & Scheidegger's model as an unambiguous data depiction, which states: if the resulting visualizations are the same, the data should also be the same. If this principle is violated, we say that there is a *confuser* (Table B.1f).

We represent aspects of a single view with the following four elements: the chart grouping (G), a channel (C), the data shown by the channel (D), and the resulting visual output (V). Typically, the term “channel" can denote the entire mapping from data to visual output. However, in our case C simply represents the name of the channel, e.g., fill color, and we use the other lower-level elements to concisely specify relations between views as predicate logic expressions as specified in Table B.1.

A single view has a grouping (G), which is the first element in our model. Each view has multiple channels, with (C) referring to a *single* channel, and correspondingly each single channel has a data mapping (D) and a resulting visual output (V). Thus, each view has one (G, C, D, V) tuple *per channel*, where G is always the same, while (C, D, V) is unique to each channel as shown by Figure B.3. Consider a bar chart grouped by category, showing average rank on the y-axis. Since G=category, C=bar height, D=average rank, and V=y-position, the tuple for the channel is then (category, bar height, average price, y-position) as seen in Figure B.3. If a channel does not have a data mapping, this is expressed as $D = 0$.

By considering a single view to be a set of (G, C, D, V) tuples (see Figure B.3), we can establish relations between two views by using predicate logic on the tuples and their equalities. When comparing the tuples of two views, we use the same lower case letter to denote equality or inequality. For instance, if two views have the same chart grouping, the relation between G_1 and G_2 is the identity: $g = 1$. Conversely, if the groupings are different, then $g \neq 1$. If a relation exists between the views A and B, and between A and C, it also exists between B and C.

A relation exists between two views if there are two tuples (one from each view) that satisfy the predicate logic formula. For example, consider the predicate logic expression of the *multiples* relation: $g = 1 \wedge \exists c((c = 1 \wedge \exists D \neq 0) \rightarrow d \neq 1)$. This relation exists between two views if there is a pair of channels (one from each view) that satisfy this expression. As illustrated in Figure B.3, the two highlighted views have the same grouping (category), but are showing different quantities on the y-axis (rank vs. score), making the *multiples* expression come true.

As discussed by Qu and Hullman [79], two encodings are showing the *same field* when the fields are *semantically* the same. We use this definition. Thus, if two fields are semantically the same, $d = 1$. To confirm semantic sameness, the user is asked to confirm if fields are the same, as seen in Figure B.5.1b if this cannot be directly determined.

We also specify that $d \neq 1$ if both data mappings are empty, but the grouping is different. For example, suppose two pie charts are respectively grouped by gender, and age group, and are both colored red. A sector of the pie chart can then represent either an age group, or a gender, yet they are colored the same. This is a potential confuser since each are showing different data, but are colored the same.

We define that the stroke color channel of charts without filled shapes (e.g., a line chart), and the fill color channel for a any chart with a fill (e.g., bar chart, scatterplot), is the same. For example, in the example shown in Figure B.1.1 we see a line chart and a scatter plot both using the color red. With our notion of channel equality, $c = 1$ since the stroke color of the line chart is the same as the fill color of the scatterplot. Furthermore, they are grouped differently ($g \neq 1$), and both of the color channels are not mapped to data. As a result of these two factors, the data mappings of the two channels are seen as different: $d \neq 1$.

The degree of redundancy and compactness in a view can be measured by the number, and severity, of detected relations. A design is more compact if it has fewer relations indicating redundancy, and more consistent if it has fewer relations indicating inconsistency. For our method, it is only necessary to know that a relation exists, and that it can be resolved. However, generating a quantitative score from these relations would be possible, and useful for many other problems. These relations are specified and visually summarized in Table B.1. We discuss each of these relations in detail in the remainder of this section.

R1: Full Redundancy. If two views are showing exactly the same data, there is a full redundancy relation between them. The full redundancy relation is present when two views have the same grouping ($g = 1$) for all channel pairs ($\forall c$). If the channels are the same ($c = 1$), then they also show the same data ($d = 1$). For example, if two bar charts are both grouped by number of cylinders ($g = 1$), and their bar height is mapped to average price, then ($c = 1 \rightarrow d = 1$) is true, i.e., there is a full redundancy relation between them.

R2: Partial Redundancy. Two views A and B are partially redundant if A is showing all data shown by B, as well as some data not shown by B. More formally, two views are considered partially redundant if they have the same grouping ($g = 1$), and for all pairs of channels showing different data ($c = 1 \land d \neq 1$), one of the channels is unmapped, and all the unmapped channels consistently belong to the same view ($\exists! D = 0$). For example, consider two bar charts, both grouped by number of cylinders ($g = 1$) and with bar height mapped to average price, but with one chart also indicating the number of cylinders via its fill color channel. When comparing the fill color channels of the charts ($c = 1$), we see that they have different data mappings ($d \neq 1$), and that one of them is not mapped to anything ($\exists! D = 0$). Thus, all the data shown by the one chart is also shown by the other chart.

R3: Multiples. There are two kinds of multiples: (1) views with same grouping but different data, or, conversely, (2) views with different groupings but same data. As an example of the former, suppose two equally grouped bar charts showing a different quantity via the bar height channel as illustrated in Figure B.4a and b. Multiples with different groupings could for example be two differently grouped bar charts showing the same aggregated dimension via the bar height channel (see Figure B.4a and d). The multiples relation is specified more precisely in Table B.1c-d.

R4: Hallucinator. Corresponding to Kindlmann and Scheidegger's model, a hallucinator is present when the same data are shown in different ways. A hallucinator exists on a canvas if two views with the same chart grouping ($g = 1$), have a common channel ($c = 1$) showing the same data ($d = 1$), but with different visual output ($v \neq 1$).

R5: Confuser. A confuser exists on a canvas if the same channel ($c = 1$) of two views has the same visual output ($v = 1$), but different data mappings ($d \neq 1$). As an example, consider charts using the same fill color (for example, reds) to show different data.

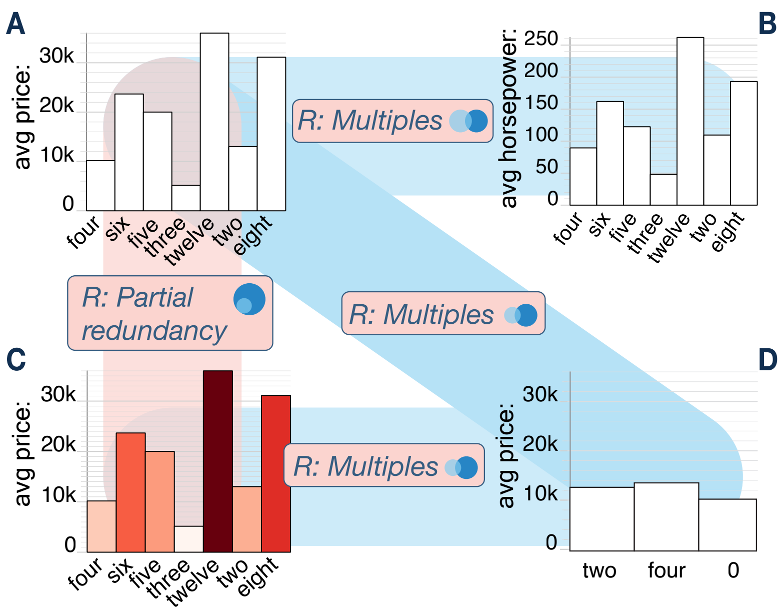


Figure B.4: These four figures illustrate all degrees of redundancy. There is a multiples relation between (a) and (b) since they both have the same grouping, but are showing different data via the bar height channel. Between a and c we see there is a partial redundancy, since (a) is showing the exact same data as (c), but (c) is also showing more data via the fill color. (a) and (d), as well as (c) and (d) are differently grouped multiples showing the same data via the bar height.

Relations identify redundancies, inconsistencies, and alternative design opportunities. They are detected by iterating over all view permutations and checking whether the permutation satisfies the predicate logic expression that corresponds to the rule. If the expression is satisfied, the relation exists between the views. Each relation has corresponding "resolutions" – operations which resolve the given relation by altering one or more of the affected views.

B.3.3 User Operations

Operations resolve potential problems between views of a canvas. The main idea behind each operation is to resolve a certain relation by changing or removing one or more views. At a low level, operations can, for example, transfer a data mapping from one view onto another, or replace two views with another view showing the same data. Our set of operations do not exhaustively express all possible combinations of low-level changes, but serves to demonstrate the wide range of possible operations to navigate the semantic space of revised designs.

Our model specifies the following classes of operations, which we summarize in Table B.1 along with associated their associated relation(s).

O1: Delete. The first operation is the most simple. If there is a *full redundancy* relation between two views, the user may delete one view. With one of the views removed, the formula for *full redundancy*, $g = 1 \wedge \forall c(c = 1 \rightarrow d = 1)$, will not evaluate to true for that pair of views, since the pair no longer exists.

O2: Homogenize. The *homogenize* operation resolves *hallucinator*, as well as *multiples* relations, where the same data are shown differently. On a high level, the *homogenize* operation makes dissimilar views more similar. For example, consider two bar charts that are showing the same data dimension with a different color scheme (hallucinator), as in Figure B.8.1a, or that are using different data domains (multiples), as shown in Figure B.6.2. The *homogenize* operation resolves these conflicts by making the visual outputs, or data domains equal for the two views. If visual outputs are made equal, the $v \neq 1$ portion of the *hallucinator* will evaluate to false and thus remove the relation. For multiples with different domains, equalizing the domains will make the views consistent. If only the data domains are different, the operation is presented to the user as *homogenize* data. When the visual outputs differ, the user will see the operation as *homogenize* style, although it also implicitly homogenizes the data domains.

O3: Differentiate. The *differentiate* operation addresses a *confuser*, where two views show different data in the same way. This is achieved by making the views to show different data in different ways (Table B.1f). For instance, if different data are shown using the same color scheme, the *differentiate* operation will assign different color schemes to the views. An example of this operation could take two views showing different data, such as age and income, where both views are mapped to the color red. This is ambiguous. Our solution is to use a different color scheme for one of the views. When the visual outputs are made different, i.e., from $v = 1$ to $v \neq 1$, the formula for a *confuser*, $c = 1 \wedge d \neq 1 \wedge v = 1$, will evaluate to false, since $v \neq 1$.

O4: Integrate. The *integrate* operation resolves redundancy to create a visually compact canvas, and can be used to resolve *partial redundancies* and certain *multiples* relations. With a *partial redundancy* relation, there are two possible solutions: delete the view showing the least data, or *integrate* the “missing” mapping into this view while deleting the other (Table B.1b). Views sharing a *multiples* relation where the data grouping is the same (Table B.1c) can be *integrated* in several ways. Since the multiples relation can only exist between two views, the act of combining these views by integration also removes the relation from the canvas. Views that are highly semantically similar are sensible to *integrate*, provided that: (1) the chart type is the same, Listing B.1: Pseudocode of the general execution flow of semantic snapping.

```
def findRelations(views):
    byView = { view: [] for view in views }
    subsets = permutations(views)
    for view1, view2 in subsets:
        for relationFn in allRelations:
            if relationFn(view1, view2):
                for view in subset:
                    byView[view].append({
                        'subset': [view1, view2],
                        'relation': relation })
    return byView

def semanticSnap(views):
    relationsByView = findRelations(views)
    view = userInput() # User selects a view
    relations = relationsByView.get(view)
    operations = [ findOperation(r) for r in relations ]
    display(operations) # User sees operations
    selectedOperation = userInput() # User selects
    newViews = selectedOperation.execute()
    return newViews
```

and (2) if $d = 1$ for the channel representing the x-axis. There are four ways to perform this *integration*: *overlay, group, stack,* and *mirror*. *Overlay* integrates multiple views into the same coordinate system. This operation can be applied to scatter plots and line charts. Figure B.1.2* shows an example of an *overlay* operation when applied to a line chart. The *mirror* operation can be applied to line charts, area charts, and bar charts. This operation first aligns the two views and then mirrors one of them, causing their marks diverge from a common origin in a manner similar to violin plots. We demonstrate an example of this in Figure B.5.2b. Similar to the group operation, the *stack* operation stacks views into a single view, turning, for instance, a set of bar charts into a stacked bar chart as shown again in Figure B.5.2c. The *group* integration bundles several views into one single view. It can, for example, turn multiple bar charts into a grouped bar chart, as illustrated in Figure B.5.2d.

Operations make high-level changes to the canvas, making it more compact or more consistent. For an operation to be applicable to a design, a certain relation must exist. When the user selects a view in the interface, our method reveals available operations to resolve a given relation. When the operation is performed, the corresponding relation is addressed.

B.3.4 Snapping Algorithm

The goal of our approach is to provide the user with a set of available design-altering operations upon selection of a single view. The outlined algorithm in Listing B.1 achieves this goal by identifying all relations and mapping them to operations for any selected view. When an operation is selected, a revised set of views is gener-

Listing B.2: Pseudocode of an example relationFn, invoked at Listing B.1 line 6, modelling a hallucinator as specified in Table B.1e.

```
def isHallucinator(view1, view2):
    if isSameGrouping(view, view2):
        pairs = findChannelPairs(view1, view2,{
            'd': 1, # same data
            'v': 0, # different visual output
            'mappedToData': 1  }) # D != 0
        return pairs.length > 0
    return 0
```

ated. The relations correspond to the descriptions in Table B.1, and can in practice be modeled as constraints or functions.

The first step of the algorithm is to identify all relations between all subsets of views. The logic of each relation is outlined in Table B.1, and is mapped to a relation function that takes in two views, and returns 1 if the relation exists between the views, or 0 if the relation does not exist between the views, as exemplified in Listing B.2. Consider line 5 in Listing B.1. Here we loop over each relationFn (relation function), and invoke it using two views as arguments. If this invocation returns 1, the relation exists between the two views. When relations are identified for all views, they are grouped by the views they affect. When the user selects a view, the view's relations are looked up and used to identify which operations are possible. Line 17 of Listing B.1 illustrates how relations are mapped to corresponding operations. When an operation is executed, a new set of views is generated and displayed to the user. With this new set of views, the algorithm is re-run, recomputing relations and potential operations.

B.4 Workflow & Implementation

Our method improves and refines canvas designs incrementally. In order to create a canvas, single visualizations must also be generated. While the creation of single visualizations is not a part of our method, we used the existing Visception visualization editor environment [56] as a basis to realize and demonstrate our method. Our semantic snapping interface enables the user to build a canvas using simple drag & drop operations from a visualization gallery and to optimize the design with semantic snapping step by step. In order to build a canvas, the user places individual views into a grid layout and is presented with a set of potential operations at every step. While browsing the operations, the user is presented with information about what they do and what potential problems in the design they resolve.

We use two primary views in our interface: the *singles view* and the *canvas view*. Both are shown in Figure B.5.1. The *singles view* is a view from where the user can drag single visualizations into the *canvas view*. Each tile in the singles view represents a data source, which, when clicked, expands to more tiles–one for each single visualization of that dataset. We highlight two of these single visualization tiles in Figure B.5, which have been dragged into the canvas view.

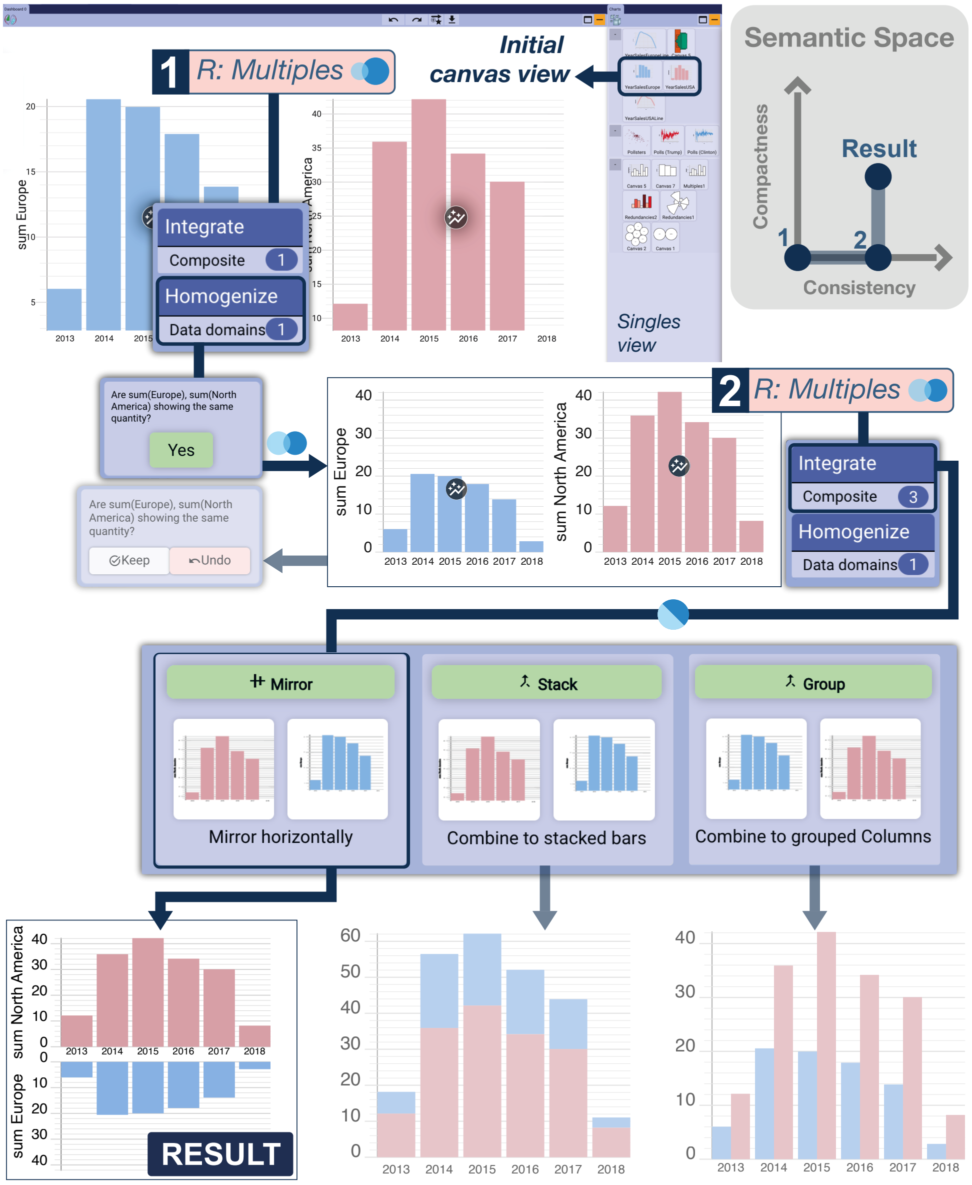


Figure B.5: Semantic snapping interface and workflow. The semantic snapping interface presents the user with a canvas to place single views within a larger layout. A clickable button on top of each view exposes the possible operations available to resolve a relation between two views, in this case a *multiples* relation (1). The user *homogenizes* the two views after confirming that the y-axes are semantically the same. This step represents a move towards increased consistency in semantic space. The user may choose to keep or undo the result of any performed operation. On execution of an operation, we recompute the set of possible operations. The user may next perform any one of three available *integration* operations to resolve a *multiples* relation in this view (2). The user selects the *mirror* operation to increase the compactness of the visualization in semantic space.

B.4.1 Workflow

The main workflow of using semantic snapping is integrated into the *canvas view*. We demonstrate this workflow in Figure B.5. As the user is constructing a canvas with multiple views, our approach detects relations and makes the corresponding operations available along every step of the way. Whenever an existing view is added to the design, or modified by an operation, relations are re-detected and the corresponding operations are updated. In order to see possible operations, the user clicks on the view of interest. When the view is clicked, a menu appears, showing how many operations are available per category (homogenize data, homogenize style, differentiate, and integrate). Only categories with available operations are displayed. The user can then click on a category and see all available operations as tiles. Each operation tile informs the user of the potential problem and its solution. Consider the canvas in our workflow example where there is a *multiples* relation between two bar charts (Figure B.5.1), showing sum of sales in Europe, and North America. To verify that the fields are semantically equal, the operation tile will ask the user "Are sum(Europe) and sum(North America) representing the same quantity?". If the user clicks "Yes", the domains are made consistent using a *homogenize operation*. When any operation is executed, the user is given the option to undo or keep it as shown in Figure B.5.2. If "keep" is clicked, the current canvas is re-evaluated and the user can proceed to explore other operations, add new views, or otherwise customize the setup.

B.4.2 Implementation

We implemented semantic snapping within the framework of Visception [56], an application written in Javascript ES6 using VueJS for user interface components and D3 for SVG rendering. The underlying framework of the authoring tool was leveraged to realize the relations and operations to support semantic snapping.

The relations are specified as functions taking in two views as parameters, returning true if they match the given relations. Operations are also defined by functions that take in a set of views, and a detected relation. From this set of views and the detected relation, we can infer what operations are possible, and also take into account the specific chart type and other edge cases. When the user modifies a design, a pipeline of four steps is run. First, all relations are detected for all sets of relations as shown in Listing B.1, and the relations are stored so that they can be looked up on a per-view basis. When the detection is done, the editor is ready for the user to specify which view to change. When the user clicks on a view, all relations and corresponding sets of views are looked up, and all possible operations are computed. When the user selects an operation, it is executed, and the existing set of views is modified, and relations are recomputed.

B.5 Case Studies

We next demonstrate our semantic snapping method workflow in three case studies. These studies include data from the 2016 US Election Results, Nightingale's historic Soldier Morbidity & Mortality, and a COVID-19 dataset. We selected these particular datasets as they are both representative of the type of data we expect to be used for our approach, as well as for their familiarity and applicability to the visualization community. Each case study represents a possible pathway through semantic space from an initial to a more compact and consistent design. We illustrate such pathways through semantic space with a semantic space map positioned in the upper right of each associated figure.

B.5.1 2016 Election Results

In this case study we demonstrate a user flow that identifies *confuser* and *multiples* relations that are resolved via *differentiate* and *integrate* operations. We also use this study case to demonstrate a flexible workflow whereby the user may perform and then revert an operation to arrive at their preferred final design.

In Figure B.1 we see an initial canvas comprised of three views depicting data from the 2016 US Election. These views show election polls over time for the two main candidates (left, top view: Democrats, bottom view: Republicans), as well as average pollster ratings for the two candidates (right scatter plot view). We localize our position in semantic space at the origin (pos. 1) in the map in the upper right of Figure B.1. We quickly identify a *confuser* relation between the bottom line chart and the right scatter plot (Figure B.1.1). This is because the color channels of the two views are using the color red as visual output. This is particularly misleading in the right scatter plot view, where each dot represents a pollster, since red may indicate that all pollsters are advocates for the Republican party. Since these charts are using the same visual output to represent different data domains, our model recommends a *differentiate* operation to change their respective visual outputs. We change the color of the scatter plot to green, as this is color is more neutral. We keep the red color in the lower left view; this makes sense to remain red, as this is the color of the US Republican Party. In our semantic map we have increased the consistency of our canvas and are now at pos. 2.

We next observe a correspondence between the two leftmost views. These views share the same x-axis, but show a different quantity on the y-axis (bottom view: avg. Trump, top view: avg. Clinton). In other words, they share a *multiples* relation. We consequently can *integrate* them to produce a more compact visualization using the *mirroring* (Figure B.1.2) or *overlaying operation* (Figure B.1.2*). Integrating these charts additionally produces a more consistent visualization, because both *overlay* and *mirror* perform an implicit axis homogenization step. To mirror or overlay, we simply select and execute either operation. In this case we first try *overlay* (Figure B.1.2*). However, while the result is very compact, it is difficult to read. We choose to do a different *integrate* operation to resolve the *multiples* relation. We revisit the available operations for this relation and select this time to *mirror* the two views (Figure B.1.2). This path in semantic space leads us to an equally consistent, while slightly less compact visualization. The resulting chart composition, however, is easier to read, which illustrates the flexibility of our approach in incorporating user goals and decision processes.

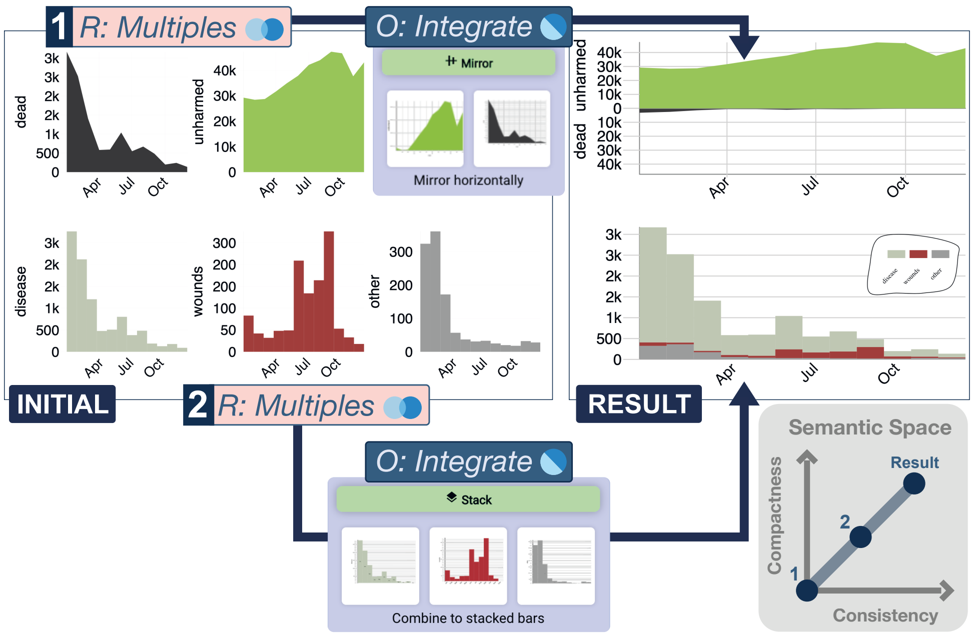


Figure B.6: Case study workflow demonstrating semantic snapping to resolve two *multiples* relations in the canvas depicting 1858 solider morbidity & mortality from the Nightingale dataset. This end result is a semantically consistent and compact visualization.

B.5.2 Nightingale Soldier Morbidity & Mortality in 1858

In this case study we use the popular Nightingale solider morbidity & mortality dataset to illustrate the use of additional operations to resolve *multiples* between canvas views.

Figure B.6 shows the fate of British soldiers in the year of 1858 in the Crimean War. In our traversal through semantic space we begin again at the origin in our semantic map at the upper right (pos. 1). The top two views of the initial canvas display area charts. The left view plots the number of soldier deaths over time while the right plots the number of unharmed soldiers over time. Because these two views comprise the same grouping (soldier morbidity & mortality) with a different data domain plotted onto the y-axis, we can say that these views share a *multiples* relation (Figure B.6.1). We can compact the views by *mirroring* (integrating) the views. When the charts are mirrored, the domains are also implicitly homogenized, which additionally increases the consistency of the resulting chart. This brings us to pos. 2 in our semantic map. It would also be possible to keep the multiples relation, and only homogenize the data domains on the y-axis. However, our choice to compact the views with mirroring illustrates that, while our model can show potential problems in a canvas, it is ultimately up to the user to decide how they wish to design their visualization.

We may also compact the three bar chart views arrayed along the bottom of the canvas that show different causes of soldier death (Figure B.6, bottom of initial canvas). Each of the three views plots by number of deaths caused by disease, wounds, or other, respectively, over time. Because these views share the same grouping (cause of solider

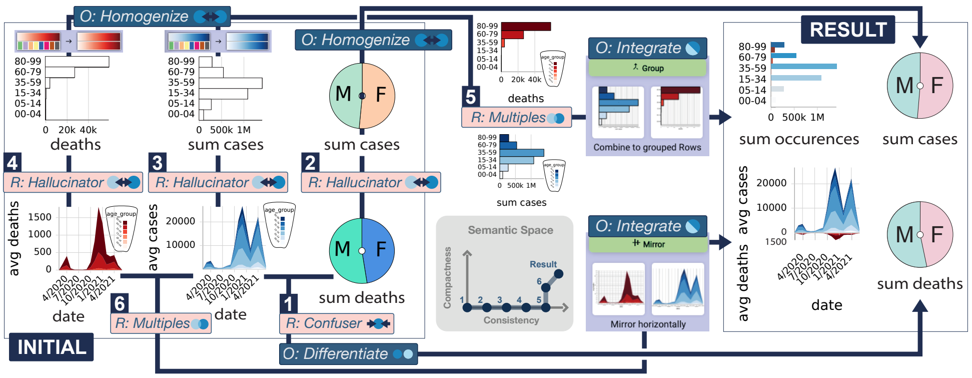


Figure B.7: Case study workflow demonstrating semantic snapping to resolve a COVID-19 dashboard with a *confuser*, several *hallucinators* and two *multiples* relations. We resolve these relations via a series of operations that include *homogenize*, *differentiate*, and *integrate*.

death) and data domain on the x-axis (time), but their data domain plotted to the y-axis is different, we identify a *multiples* relation (Figure B.6.2). However, by looking at the data, we also know that the y-axes represent the same *semantic* quantity – i.e., number of deaths. As a consequence of this, they can be integrated via a *group* or *stack* operation. Since our goal is to produce a maximally compact visualization, we choose to *stack* the views. This step additionally homogenizes the data domains for increased consisstency. This brings us to the result point in our semantic map.

B.5.3 COVID-19 in Germany

COVID-19 dashboards are now ubiquitous in society with great importance for public health. However, integration of numerous charts to demonstrate various data aspects in a dashboard may introduce numerous possible conceptual and perceptual pitfalls. For our final case study we demonstrate the ability of our approach to assist in resolving the complexities of creating a semantically consistent COVID-19 dashboard. We demonstrate an overview of this workflow in Figure B.8.

The initial layout as shown in the central part of Figure B.8 shows six charts. The first chart column shows COVID-19 deaths grouped by age, where the top bar chart represents total deaths and the bottom streamgraph indicates deaths over time. The second column displays COVID-19 cases that are again grouped by age, with the top bar chart indicating summed cases while the below streamgraph shows case load over time. The rightmost column shows two pie charts grouped by gender, where the top chart shows cases while the bottom shows deaths. Our goal is create a dashboard using multiple chart types that clearly presents the COVID-19 cases and deaths distributed by age group and gender in Germany.

As in the prior case examples we have a number of different routes through which we can traverse the semantic space, as indicated in map in the lower middle of Figure B.8. In this case study we describe the navy blue indicated route, beginning with the *confuser* that our system identifies between the pie chart showing COVID-19 deaths and the COVID-19 cases over time chart (Figure B.8.1). This is a *confuser* because the female segment in the pie chart uses the same blue as for the color mapping in the chart showing cases grouped by age over time. We perform the suggested *differentiate* operation to clarify the different groupings by changing the color mapping of genders to a light green for males and pink for females. This increases consistency in semantic space. We next resolve the *hallucinator* relation between the two pie charts by *homogenizing* the color mapping of both charts so the cases chart receives the same green and pink color mapping to males and females, respectively, for increased consistency in semantic space (Figure B.8.2).

Two additional *hallucinators* exist, one between the age-grouped COVID-19 cases charts (Figure B.8.3) and the second between the age-grouped COVID-19 deaths charts (Figure B.8.4). Each are classified as *hallucinators* because the chart data and groupings are identical but they do not share the same color mapping. We resolve the first *hallucinator* between the two case charts with a *homogenize* operation that applies the same continuous blue color mapping in the streamgraph to the bar chart. We resolve the second *hallucinator* the same way for the deaths chart, by applying the continuous red color mapping in the deaths over time streamgraph to the corresponding bar chart. Each of these operations sequentially improves consistency in semantic space.

We may compact our dashboard visualization by resolving two *multiples* relations that our system identifies. The first *multiples* relation exists between the two bar charts, which we *integrate* into a single grouped row chart (Figure B.8.5). A second compacting step in semantic space *integrates* the two streamgraphs in a mirroring operation to resolve their *multiples* relation (Figure B.8.6). In both *integration* steps the system implicitly *homogenizes* the data domains as well. The resulting COVID-19 dashboard in the right of Figure B.8 is a much more compact and semantically consistent visualization with the aid of our approach.

B.6 Discussion & Limitations

We realized our method by implementing and embedding it into the Visception visual authoring system [56]. For specifying rules, it is necessary to be able to retrieve detailed information about each channel mapping, as well as the chart type and grouping. We believe this information should be accessible in most frameworks. Specifying chart operations requires more knowledge about the underlying architecture and programming interface. For instance, our framework had support for nesting visualizations, which was highly useful for generating grouped and stacked charts. For example, two bar charts grouped the same, showing two different quantitative attributes can be grouped into a stacked bar chart, where the outer bar is grouped the same, and the inner grouping is one bar for each of the two attributes. In principle, however, we believe that our approach is applicable to a variety of different visualization systems, even if the specifics of how individual operations are implemented will differ. At present, our prototype only supports a limited number of common chart types: line charts, bar charts, pie charts, scatter plots, and streamgraphs. For addressing a wider range of charts, we believe that a framework for unifying the reasoning about these charts could allow for a more generally applicable realization of semantic snapping. We believe that frameworks that allow for expressing and modifying charts on a general level are ideal for implementing our semantic snapping concept.

At present, a canvas is limited to views of a single tabular dataset. For dealing with more advanced multi-table setups, our approach would have to be built on top of an additional abstraction over these different data topologies. Such an abstraction layer would enable a more general implementation of semantic snapping that could also facilitate the incorporation of other dataset types such as network data. Likewise, techniques such as interactive linking & brushing and crossfiltering are frequently used in multi-view visualizations but currently not explicitly supported in our framework, which also represents an interesting challenge for future research.

The layout of the views is an important factor in an overall design, which is currently not addressed in our approach but is definitely worthy of further investigation. It would be possible to specify more advanced relations by incorporating the spatial arrangement of individual views. For instance, if two views are sufficiently spatially separated, a *confuser* could be classified as less severe. Likewise, taking into account spatial arrangement could extend the space of operations as, for example, a *differentiate* operation could move views further apart or even add graphical separators or visual groupings. This is an important direction for future research. Related to this, since currently the number of possible operations is sufficiently small, we do not perform any explicit sorting. However, a larger number of possibilities would necessitate to incorporate an appropriate mechanism for prioritizing operations. We believe that such a sorting of potential revisions using for example Qu & Hullman's effectiveness preservation score [79] would be useful when there are many potential solutions.

While the operations of our method alter the design and resolve potential inconsistencies, it would provide more flexibility and design freedom if they were customizable. For example, the mirror operation could be parameterized by letting the user decide the spacing between the views and the placement of the labels. A general assumption of our method is that the existing views are already well-designed individually. However, when a view is placed into a multi-view design, the aspect ratio and size will change. Keeping font sizes and other styles consistent across a design becomes tedious. While our operations do combine views and optimize design, they do not at present allow for a final fine-tuning of, for instance, font sizes. Such global controls are not a part of our method, but would be highly helpful in any multi-view visualization design process.

Finally, our method is based on general principles in the sense of Kindlmann and Scheidegger [54] and thus does not take into account an explicit task specification. While this focus was deliberate, since meaningfully characterizing user tasks is a significant challenge of its own that would also explode the design space, we still believe that exploring how different types of general user tasks could guide the evaluation of relationships and the presentation of operations is an important topic for future research.

B.7 Conclusion

We presented semantic snapping, a semi-automatic guided method that allows for incrementally refining multi-view visualizations. While previous work on multi-view visualizations has given us guidelines and constraints for reasoning about and improving visualizations, we further operationalized these concepts by (1) specifying relations between views precisely, and (2) proposing how each relation can be resolved by an operation. Each operation is a step in the semantic space with two axes representing the consistency and compactness. Furthermore, we presented a prototype implementation of our method, where users can perform operations to gradually refine a multi-view visualization design. In the future, believe that our approach to specifying relations and corresponding operations can be applied to more elements of multi-view visualizations such as their layout. Furthermore, many additional rules and guidelines for single visualizations could be adapted to or extended for multi-view visualizations.

The research presented in this paper was supported by the MetaVis project (#250133) funded by the Research Council of Norway as well as the VIDI project (#813558) funded by the the Trond Mohn Foundation in Bergen, Norway.

Semantic Snapping for Guided Multi-View Visualization Design: Supplementary Material

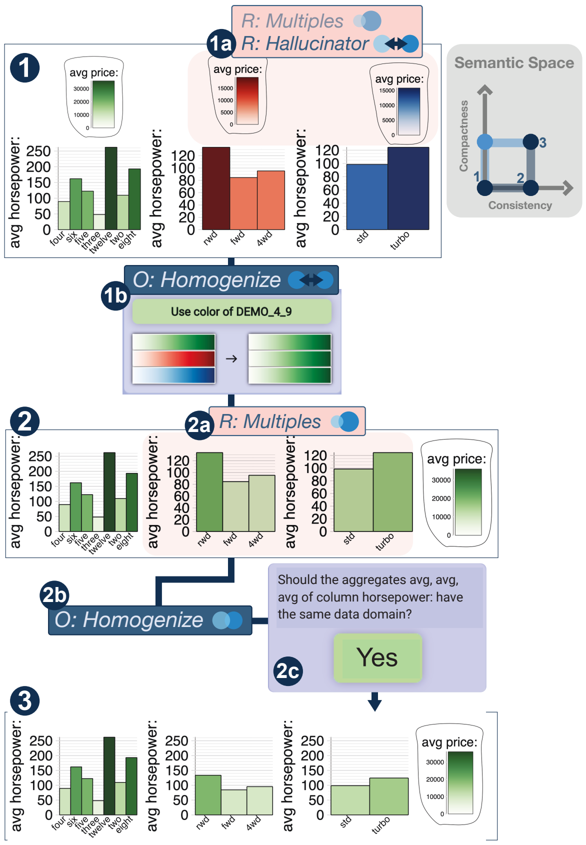


Figure B.8: Case study workflow demonstrating semantic snapping to resolve a *hallucinator* and a *multiples* relation in a canvas plotting three variables from the cars dataset against average horsepower. We resolve these relations via two *homogenization* operations, one on the color channel and the other on the data domain placed on the y-axis for the two rightmost views.

This is an additional case study that could not be included in the paper due to space restrictions. In this example, we illustrate the resolution of a *hallucinator* and *multiples* relation using the cars dataset. We begin with the three bar chart views arrayed on the canvas in Figure B.8.1. These views plot average horsepower against: number of cylinders (leftmost view), wheel drive (middle view) and aspiration (rightmost view). Each of these views constitute different pairwise groupings that share the same data domain plotted to the y-axis (average horsepower). This describes a *multiples* relation. We furthermore see that all three views use the color channel to visualize the average price, but with different data domains. This constitutes a *hallucinator* (Figure B.8.1a). We resolve this *hallucinator* with a

homogenize operation that changes the visual output to green (Figure B.8.1b), in addition to homogenizing the data domains. This moves us from pos. 1 at the origin of our semantic space a more consistent position in the space (pos. 2).
With consistent color outputs across all views (Figure B.8.2), as a consequence of the *multiples* relation between the three views, we are able to *homogenize* the data domains of the three views (Figure B.8.2a-b). view (Figure B.8.2a-b). Our model offers a sanity check for any explicit operation to *homogenize* the data domain (Figure B.8.2c). This is to help avoid a potential backwards operation in semantic space, i.e., create a less compact and/or less consistent visualization), that unreasonably distorts the data. However, the user ultimately validates whether the views they wish to combine are semantically the same. The option to undo their move is always available. With successful *homogenization* of the y-axis data domains in all views we have arrived to pos. 3 in semantic space. Our resulting visualization is both more semantically consistent and compact than it began (Figure B.8.3).

B

Paper C

Content-Driven Layout for Visualization Design

Yngve Sekse Kristiansen, Laura Garrison, Stefan Bruckner

University of Bergen, Norway

Abstract

Multi-view visualizations are typically presented in a grid layout with elements positioned according to their bounding rectangles. These rectangles often contain unused white space. In cases where Tufte's Shrink Principle can be applied to reduce non-data-ink without impairing the communication of information, unused white space can be utilized for the placement of other elements. This is often done in manually "hand-crafted" layouts by designers. However, upon changes to individual elements, this design process has to be repeated. To reduce non-data-ink and repetitive manual design, we contribute a method for automatically turning a grid layout into a *content-driven* layout, where elements are positioned with respect to their contents. Existing approaches have explored the use of a force simulation in conjunction with proxy geometries to simplify collision handling for irregular shapes. Such customized force directed layouts are usually unstable, and often require additional constraints to run properly. In addition, proxy geometries become less accurate and effective with more irregular shapes. To solve these shortcomings, we contribute an approach for identifying central elements in an original grid layout in order to set up corresponding attractive forces. Furthermore, we utilize an image-based approach for collision detection and avoidance that works accurately for highly irregular shapes. We demonstrate the utility of our approach with three case studies.

C.1 Introduction

Data-driven infographics and dashboards often have a small set of elements positioned by an underlying grid layout, which considers only bounding rectangles of individual elements. In this paper, we propose means for compacting the layout of designs that have (1) unused white space between contents of grid cells and (2) one or a few identifiable central elements. By Tufte's Shrink Principle [102], many data graphics can have their data-density increased and be reduced in area without loss of information or readability. One way to achieve this is to reduce the amount of non-data-ink, which occurs frequently in designs with underlying grid

In *Proceedings of the International Symposium on Visual Information Communication and Interaction* layouts. This process is frequently performed manually. However, manual designs become tedious to re-design upon changes to individual elements due to manual revisions or changes in the underlying data.

Consider a user placing a set of visual elements onto a canvas in order to create a layout. A concrete example of this would be a user designing a Tableau [97] dashboard by arranging elements into a grid layout, as shown in Figure C.1a. In this layout, each element is contained by a bounding rectangle. Within these bounding rectangles, there may be much unused white space. However, a grid layout does not allow the user to utilize this space efficiently.

One alternative to a grid layout is a force-directed layout [33], which allows for flexible control of distances between elements. However, force-directed layouts are often designed to work with only simple shapes such as circles, and rarely support irregular shapes where other representations are necessary. Ali et al. [5] utilized a force layout wherein irregular shapes are represented by convex hulls. Such proxy geometries are approximations to complex, fine-grained details, and become less accurate with more irregular and complex shapes. For example, the marks of the scatter plot on the right in Figure C.4 are difficult to describe geometrically without losing some fidelity. As another example, consider the lower concave region of the lungs in Figure C.2. A convex hull would fail to properly capture this region. In this paper, we aim to enable the use of this white space by employing a force-directed layout that reflects the original grid layout topology by attracting peripheral elements towards central elements, avoiding content-to-content collisions with high precision, even for highly irregular shapes. We present a scheme for better preserving the original grid layout topology, which applies attractive forces only towards a small set of inferred central elements. We use a novel image-based approach for content-to-content repulsion that enables fine-grained control of distances between elements. With three case studies, we demonstrate how our approach successfully turns grid layouts with a high degree of unused white space into content-driven layouts. These layouts effectively utilize white space around irregular shapes, and are able to better capture the aesthetic qualities of a manual design.

C.2 Related Work

The primary goal of a multi-view visualization layout is to position elements so that they convey information to the user as intended. Such layouts are typically designed manually, often with the help of a grid [31]. Grid-based layouts are frequently used in visualization software programs such as Tableau [97] and for other multi-view visualizations. Since then, many more sophisticated techniques for multi-view visualizations have been introduced. Our work is a continuation of this, as we provide a more content-driven alternative to the typically used grid layout.

Multi-view visualization layout techniques have been explored by several researchers. For example, Javed et al. [49] defined the space of composite visualization in terms of four operators: juxtaposition, superimposition, overloading, and nesting. Chen et al. [20] explored 360 visualizations and identified a set of composition and configuration patterns in multi-view visualizations. Also on the topic of dashboard design, Sarikaya et al. [84] contributed a design space across several dimensions, including functional design, purpose, audience, and data semantics. They further pointed out that dashboards are currently venturing into the realm of infographics. In this work, we aim to enable for such a transition from a traditional dashboard towards a more “infographics”-like dashboard by transforming its layout into a more compact “hand-crafted” version.

Automated layout techniques typically fall into one of two categories: machine learning techniques, and constraint based techniques [62]. These techniques often extend such quality measures to encompass higher-level concepts. Jahanian et al. [48] quantified concepts from art and aesthetics into a system for automatically designing magazine covers. This knowledge was further leveraged to create a recommendation system [47] for generating magazine covers by adhering to high-level intuitive cues such as "formal" or "sporty". In their layout process, they identify non-salient image segments of the main image, and use them as potential regions to place secondary content. Yang et al. [115] proposed a system to automatically generate layouts by leveraging expert-designed, topic-dependent templates and a computational framework for integrating and harmonizing high-level aesthetics and low-level image features. Moritz et al. [72] proposed Draco, a system for formalizing such design knowledge and guidelines, making them accessible to non-experts through a constraint-based Answer Set Programming language.

Techniques for **synthesis and optimization of grid layouts** are useful since they operate on an already widely used layout technique. Jacobs et al. [45] introduced an approach for adapting grid-based magazine layouts to different screen sizes. Xu et al. [114] proposed an interface for beautifying layouts by visualizing and editing relationships with sketching gestures. Sketchplorer [100] integrates sketch-based design with a real-time layout optimizer. It automatically infers the designer's task and searches for local and global improvements. Dayama et al. [24] presented a method for interactively transferring a layout of a single user interface design to another user interface. Li et al. [59] proposed the use of LayoutGAN, a generative adversarial network, to synthesize, model, and edit geometric relations between different 2D elements. Schrier et al. [89] presented a system for assembling documents from different sources into a single grid-based design, which automatically adapts to different viewing conditions and content selections.

Space usage optimization has been explored in the context of multi-view visualization as well as windowing management. A foundational idea behind such optimization is to reduce the loss of white space. Analogous to this idea, Albano and Sapuppo [4] explored heuristic methods for allocating irregular 2D shapes with a minimal amount of white space for reducing loss of physical fabric while cutting. Similarly, Bouganis and Shanahan [15] used computer vision and AI techniques to minimize white space in layouts with varying shapes on both regular and irregular surfaces. Ishak and Feiner [44] devised a technique dubbed content-aware layout to position several windows by taking their contents into account. Steinberger et al. [96] presented a dynamic window management technique which identifies coherent information that is then used as a basis for moving and scaling windows. Haraty et al. [38] proposed a genetic algorithm for optimizing multi-window layouts for specific tasks. Ishak and Feiner [44] devised a technique dubbed content-aware layout to position several windows by taking their contents into account. Zheng et al. [117] introduced an approach to generate high quality, content-aware magazine graphic designs by using a deep learning generative model trained on a large magazine-layout dataset. Effective screen space usage is becoming even more relevant with a myriad of different devices and screen sizes being used to dissect data. Kim et al. [51] characterized different responsive visualization strategies by analyzing 378 pairs of large screen and small screen visualizations. They identify implications for existing works as well as future work. Andrews and Smrdel [6] applied the principles of responsive web design, and leveraged these principles to make commonly used visualizations responsive. Motivated by interviewing journalists, and analyzing 231 responsive news visualizations, Hoffswell et al. [41] proposed a prototype system for previewing and editing multiple visualization versions simultaneously. While such works focus on optimization of space usage, they typically consider elements only by their bounding rectangles, rather than contents. Our work shares the goal of efficient use of space, but achieves it for cases where it is more desirable to have a

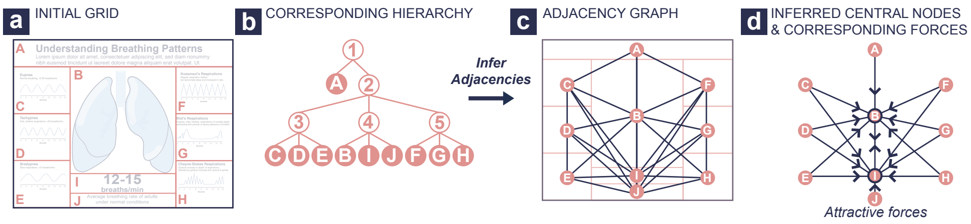

a INITIAL GRID
b CORRESPONDING HIERARCHY
c ADJACENCY GRAPH
d INFERRED CENTRAL NODES & CORRESPONDING FORCES
Understanding Breathing Patterns
12-15 breaths/min
Infer Adjacencies
Attractive forces

Figure C.1: Here we see the steps for generating a set of attractive forces from an initial grid layout (a), which corresponds to a grid layout hierarchy (b). From this hierarchy (b) we infer all adjacent elements in the grid layout (a) using Algorithm 1, and store them as a neighborhood graph (c). This neighborhood graph is again used to determine which elements are central in the original layout (a). In this case, elements **B** and **I** are inferred to be central nodes (d). This results in attractive forces directed only *towards* **B** and **I**, indicated by the direction of the arrows along the links (d).

compact layout that adheres precisely to the contents of its elements.

The common **force-directed layout approach** [33] is often used alone or in conjunction with other constraints to achieve highly flexible graph layouts. Force-directed layouts have a wider range of applications, including visualizing biological pathways [34], improving Euler diagrams [69], rendering Lombardi-style graphs [21], targeting network spatialization [46], and rapidly visualizing large networks [16]. Dengler et al. [25] used a generalization of a simple force-layout to generate diagram layouts satisfying geometric and aesthetic/perceptual constraints. We are inspired by the work of Ali et al. [5] who proposed a tool for creating blueprints to integrate interactive illustrations into one layout. However, their approach represented elements by their convex hulls, limiting its ability to tightly arrange irregular shapes (for example, the concave region of the lungs in Figure C.2). Our method uses an image-based, rather than geometric, approach to deriving Euclidean content-to-content distances between elements, which does not suffer from these drawbacks.

C.3 Content-Driven Layout

Layouts used in the context of multi-view visualizations typically position elements by their bounding rectangles, rather than contents. In a *content-driven* layout, elements are positioned by their contents, i.e., pixels that are not white space. A content-driven layout can make better use of previously unused white space, resulting in a potentially more visually pleasant, compact and “hand-crafted” appearance as seen in Figure C.3. The main goal of our method is to enable the automatic transformation of a grid layout to a content-driven layout. This is achieved by generating a force-directed layout, with forces derived from the original grid layout.

C.3.1 Terminology

A **grid layout** (Figure C.1a) is a layout where a rectangular space is recursively subdivided horizontally or vertically. A single **element** contains visual content and white space, while **content** excludes white space. Each grid layout corresponds to a hierarchy as seen in Figure C.1b. If several elements in the grid layout share the same strip of space, they have the same parent node in the corresponding hierarchy, where the left-to-right order of nodes correspond to the order in which elements appear in the grid layout. In this hierarchy, non-leaf nodes have a **flow** that controls the order and direction in which their children are placed, which is either vertical left-to-right, or horizontal top-to-bottom. Consider how elements C, D, E in Figure C.1a correspond to nodes in the hierarchy in Figure C.1b. These three nodes also share the same parent (node 3 in Figure C.1b), which flows vertically top-to-bottom, while A is a child of the root node (node 1 in Figure C.1b), which also flows vertically top-to-bottom. Adjacency relationships between elements in the grid layout are referred to as the **topology** of the layout. For example, in Figure C.1a we see that elements C, D, and E are to the left, and appear in order from top to bottom, while B, I, and J are in the middle, and element A is on the top. This topology can be concisely expressed by an extracted **adjacency graph** (Figure C.1c), where each element in the grid layout is represented as a node positioned at its original center of gravity, and each adjacency between two elements in the grid layout is represented as a link between two nodes.

C.3.2 Overview

On a high level, our method follows a pipeline composed of the following steps. First, we derive adjacency relationships from the underlying tree structure of the grid layout (Figure C.1b) and store them as an adjacency graph (Figure C.1c). This adjacency graph is then used to identify a set of nodes (Figure C.1d), which correspond to the visually most central elements in the original layout (Figure C.1a, elements B and I). Attractive forces are set up such that in a force simulation, elements are pulled only *towards* these central elements. Image-based repulsive forces between each distinct pair of elements are used for content-to-content repulsion. With these attractive and repulsive forces, a force simulation is started, where each element is initially placed according to its original grid layout position. The execution of the force simulation transforms the grid layout into a content-driven layout.

C.3.3 Attractive Forces

Content-to-content attraction ensures that elements gravitate towards each other. Visually, it is easy to see that a grid layout has a small set of central elements. To represent the topology of the original grid layout, we model attractive forces so that they are all directed *towards* these central elements. To achieve this, we derive a graph from the original grid layout where each link represents the adjacency relationship between two elements in the grid layout. From this adjacency graph, we infer a minimal set of central nodes. Finally, we apply attractive forces that pull elements towards central elements.

We define the adjacency graph by a conditional that determines adjacency between two elements a and b. Throughout this paragraph, we refer to lines in Algorithm 1, and consider the nodes B and F in Figure C.1 as examples of a and b, respectively. Adjacency between non-sibling nodes is determined by first finding their LCA (lowest common ancestor) in the grid hierarchy (line 10). Next, we extract the paths $path_a$ and $path_b$ between the nodes and their LCA (line 11), which would correspond to the paths consisting of node 4 between B and 2, and 5 between F and 2. Now, the goal is to find out if there are no other leaf nodes *between* the two possibly adjacent nodes. Thus, we check how $path_a$ and $path_b$ are positioned in relation to each other by using the subroutine *pos* (line 1). This subroutine checks the order of appearance of nodes with respect to the flow of the LCA. For example, since the flow of the LCA (node 2) is horizontal, and there are no other horizontally flowing nodes *between* B and

Algorithm 1 Inferring adjacency graph

```
1:  procedure POS($path$)
2:      $ancestorFlow$ = flow of first node in path
3:      $path'$ = $path$ without its two first nodes
4:      $nlp$ = nodes in $path'$ with $parent.flow = ancestorFlow$
5:      return '$only$' if $nlp.length = 0$
6:      return '$first$' if $\forall n \in nlp : index(n) = 0$
7:      return '$last$' if $\forall n \in nlp : index(n) = |\text{siblings}(n)|$
8:      return '$middle$'
9:  procedure ISADJACENT($a$, $b$)
10:     if $a$ and $b$ are siblings: return $true$
11:     $l \leftarrow \text{LCA}(a,b)$                         ▷ Lowest common ancestor
12:     $a'$, $b'$ = ancestors of $a$ and $b$ that are children of $l$
13:     if index($a'$) > index($b'$): return isAdjacent($b$, $a$)
14:     if index($b'$) - index($a'$) > 1: return $false$
15:     $path_a, path_b$ = path from $l$ to $a$, and $l$ to $b$
16:     return pos$(path_a) \in \{'last','only'\} \wedge$ pos$(path_b) \in \{'first','only'\}$
```

2, or F and 2, *pos* will in both cases return '*only*'. If there was a horizontally flowing node on the path from F to 2, F would have to be the horizontally first or only leaf node to be in contact with B. Conversely, B would have to be the horizontally last or only leaf node in the case of a horizontally flowing node between B and 2. Thus, two nodes are adjacent if *pos* evaluates to '*last*' or '*only*' for $path_a$, and '*first*' or '*only*' for $path_b$ as seen on line 16.

Algorithm 2 Identifying central elements

```
1:  procedure FINDCENTRALNODES(N)
2:      $C \leftarrow []$                                   ▷ central nodes
3:      $V \leftarrow []$                                   ▷ visited nodes
4:      until all nodes in $N$ are in $C$ or $V$, do:
5:        for each node $n : n \in N$, do:
6:          $n.a \leftarrow$ number of unvisited neighbors of $n$
7:          $n.b \leftarrow$ number of neighbors connected to unvisited node
8:        $N' \leftarrow V$ if $|V| \neq \emptyset$, otherwise $N$
9:        $a_{max} \leftarrow max(\{n.a | n \in N'\})$                   ▷ Greatest $a$
10:       $A \leftarrow \{n | n \in V \wedge n.a = a_{max}\}$            ▷ Nodes with greatest $a$
11:       $b_{max} \leftarrow max(\{n.b | n \in A\})$                    ▷ Greatest $b$
12:       for each node $n \in A$ where $n.b = b_{max}$, do:
13:         add $n$ to $C$                                  ▷ $n$ is central
14:         add all neighbors of $n$ to $V$                 ▷ neighbors of n are visited
```

From the adjacency graph, we can now determine the most central elements. For every node n in the adjacency graph, we count how many of its immediate neighbors are unvisited, and store this number as $n.a$ as seen on line 6 of Algorithm 2. Furthermore, we count how many of its neighboring nodes are connected to an unvisited node, and store this number as $n.b$ (Algorithm 2, line 7). We then identify the highest a-value among all non-central nodes (Algorithm 2, line 9), and find all nodes with this a-value (Algorithm 2, line 10). From these nodes, we find the highest b-value, as shown on line 11 of Algorithm 2. Finally, we select only the nodes with the highest identified a-value, and associated highest b-value (Algorithm 2, line 12), which are tagged as central (Algorithm 2, line 13). Furthermore, the neighbors of each central node are tagged as visited on line 14 of Algorithm 2.

Based on the computed central nodes, we can now apply attractive forces drawing corresponding peripheral elements towards central elements, and each central element towards all other central elements. In other words, no elements are attracted *towards* peripheral elements, and bidirectional attractive forces are set up between all central elements. The **directionality** of a force attracting element B towards element A is defined as the normalized vector going from the center of gravity of B, to the center of gravity of A. We use standard spring forces based on Hooke's law with a constant **strength**.

C.3.4 Repulsive Forces

Content-to-content repulsion prevents elements from overlapping. The first step towards achieving this is to consider each element by its content rather than its bounding rectangle. In existing approaches, irregular shapes are often simplified to geometric shapes, which are then used to compute content-to-content distances and overlaps. For example, Ali et al. [5] used convex hulls to compute content-to-content distances and collisions. Such geometry-based approaches become less reliable and more difficult to handle with irregular shapes. For example, consider the image of a lung in Figure C.2. Using a convex hull around this fails to represent the gap between the two lungs. Hence, we use an image-based approach to detect and avoid overlapping contents. We achieve accurate content-to-content distances by utilizing the Euclidean distance transform [12], which we compute for every element by using Meijster's algorithm [68]. The distance transform of an element A is denoted as dt_A, and encodes in every pixel the Euclidean distance to the nearest content pixel. We compute the distance transform from a binary image of the original content, where 1 is content and 0 is white space.

Repulsive forces are applied between all elements. These forces are active only if elements have overlapping bounding rectangles. If two elements A and B have intersecting bounding rectangles at the region $A \cap B$, they may also have overlapping contents. To compute the distance between the contents of the two views, we first consider the distance transforms of A and B, dt_A and dt_B. Then, we clip out the region $A \cap B$ from the distance transforms of A and B, giving us the images dt'_A and dt'_B. These clipped distance transforms are then summed together pixel by pixel, giving the content-to-content distance at the smallest pixel as follows:

$$d_{AB} = \min(dt'_A + dt'_B) \tag{C.1}$$

The repulsive force pushes the content of element B away from the content of element A. The **directionality** of repulsion is determined by the gradient of the distance transform. To compute the directionality, we consider the distance transform gradient of A, at the region of intersection $A \cap B$, denoted $\nabla dt'A$. The directionality of repulsion is the normalized sum of vectors from $\nabla dt'_A$.

In practice, it is often desirable to have a certain margin M around the content of an element. This requires correspondingly enlarging the bounding rectangle of an element to avoid issues when the content is close to its borders. The **repulsive strength** between two elements A and B is then computed as:

$$\rho_{AB} = \frac{1}{\max(d_{AB} - M, \theta)} \tag{C.2}$$

where θ acts as a lower threshold for the denominator of the expression, avoiding near infinite strengths as $d_{AB} - M$ approaches zero.

C.4 Implementation

We implemented our approach by using Vue.js and TypeScript for rendering the layout elements, and D3.js [14] for running the force simulation. The attractive and repulsive forces are implemented as custom forces in D3's force-directed layout implementation. Layout elements are reactively linked to underlying layout objects, which are updated in each step of the layout simulation.

The underlying grid layout is specified as a recursive JSON object corresponding to the underlying tree structure outlined in Section C.3. Non-leaf nodes have the following properties: (1) **flow**, which is either vertical or horizontal, and **children**, a list of children. Leaf nodes have a **source** property which may be an image link, or a Vega-Lite [87] specification string which enables the easy integration of a rich set of different types of visualizations.

The most expensive part of our algorithm is finding the content-to-content repulsion between nodes. More specifically, adding together the distance transforms at the region of intersection, and finding the smallest pixel is most time-consuming. We currently use a straightforward CPU implementation using Javascript's Float64Array to represent the underlying distance transforms, so a GPU-based approach could lead to significant time reductions. The performance of this operation is also highly dependent on the resolution of the distance transform. To illustrate this impact, we present the performance of our algorithm with two different distance transform resolutions: (1) $\max(width, height) = 50$, and (2) $\max(width, height) = 200$. We generated the lungs example in Figure C.2 using the Mozilla Firefox browser, on a machine with 32GB RAM and a Intel Core i7-7700K CPU @ 4.20GHz. With both resolutions, convergence was reached after 35 iterations of the force layout, which took 905 and 1207 milliseconds for resolutions 50 and 200, respectively. On average, each iteration took 25.8 milliseconds with the resolution of 200, and 18 milliseconds with a resolution of 50 (in both cases, this includes rendering time). In our experiments, we found that even when using resolutions significantly lower than rendering resolutions, impacts on the final results were negligible with large-scale structure shapes such as the lungs in Figure C.2. However, a higher resolution was required to capture finer aspects such as very small bubbles in the bubble plot in Figure C.4.

C.5 Case Studies

In this section we illustrate the value of our approach in three case studies in collaboration with a designer who is a coauthor of this paper. Each case study begins with the project brief that a designer or data journalist would typically receive at the beginning of a project. We then discuss the ideation phase, where the designer develops layout concepts, and the subsequent challenges faced in typical production tools such as Tableau or Adobe Illustrator. We then compare the functional capabilities of these typical production tools against our content-driven layout algorithm. For all case studies we use styling from the Vega cook book repository. A demonstration video illustrating our approach and these case studies is available at: `https://tinyurl.com/y652kt6n`.

C.5.1 Respiration Patterns

We begin with a case study illustrating a relatively simple white space optimization challenge that is not possible to achieve in standard visualization tools, such as in Tableau. The design

https://github.com/aezarebski/vegacookbook

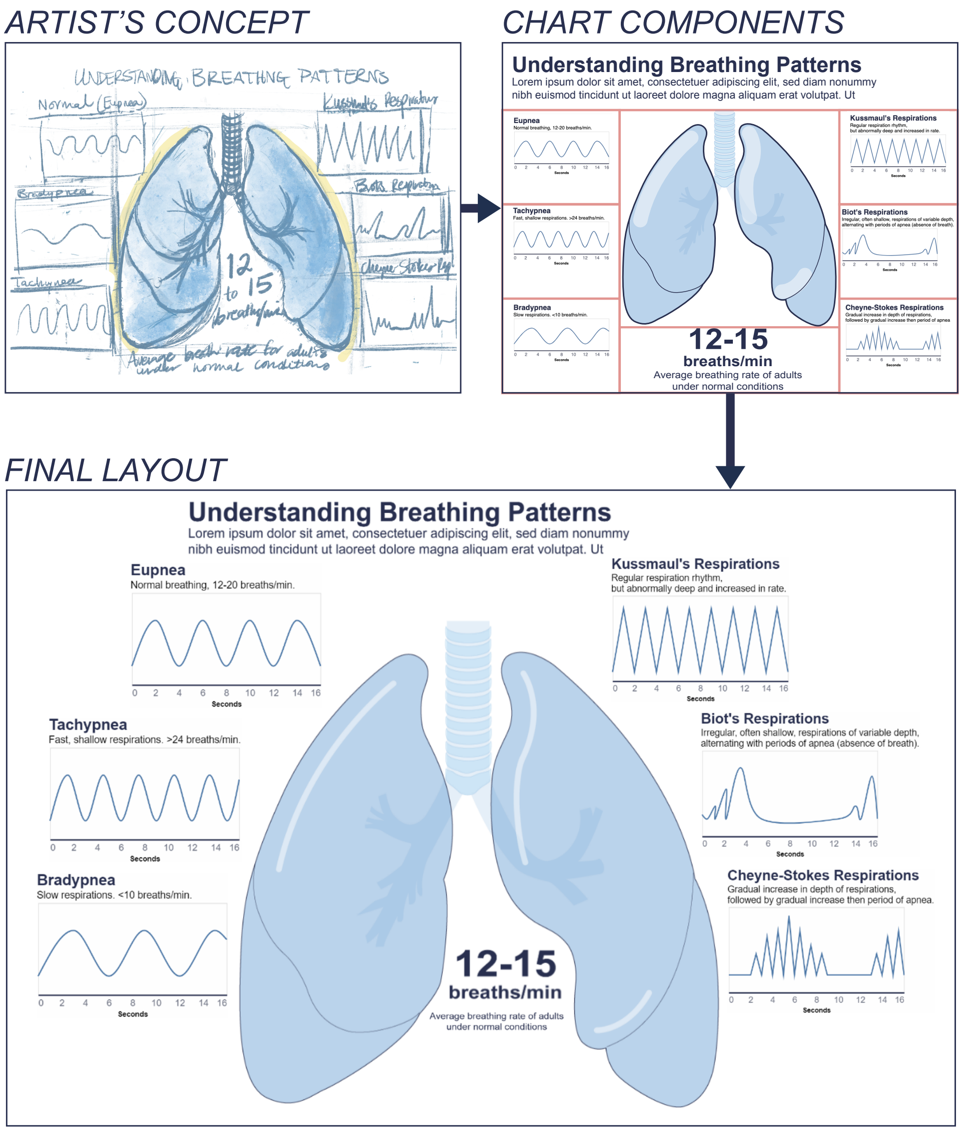

ARTIST'S CONCEPT
UNDERSTANDING BREATHING PATTERNS
Normal (Eupnea)
Kussmaul's Respirations
Bradypnea
Biot's Respirations
Tachypnea
Cheyne-Stokes Resp
12 to 15 breaths/min
Average breath rate for adults under normal conditions
CHART COMPONENTS
FINAL LAYOUT
Understanding Breathing Patterns
Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut
Eupnea
Normal breathing, 12-20 breaths/min.
Tachypnea
Fast, shallow respirations. >24 breaths/min.
Bradypnea
Slow respirations. <10 breaths/min.
Kussmaul's Respirations
Regular respiration rhythm, but abnormally deep and increased in rate.
Biot's Respirations
Irregular, often shallow, respirations of variable depth, alternating with periods of apnea (absence of breath).
Cheyne-Stokes Respirations
Gradual increase in depth of respirations, followed by gradual increase then period of apnea.
0 2 4 6 8 10 12 14 16
Seconds
12-15
breaths/min
Average breathing rate of adults under normal conditions

Figure C.2: Using our content-driven layout approach, the artist is able to achieve a composition where the six respiratory patterns are arrayed around the margins of the lungs.

brief for this scenario is to design an infographic describing six common breathing patterns as seen in the output of a ventilator. This is intended for medical staff to use as a study aid, with an engaging vector graphic of the lungs and additional text describing the normal rate of respiration. The lungs are intended as the center piece, with the six respiration patterns arrayed around this central graphic.

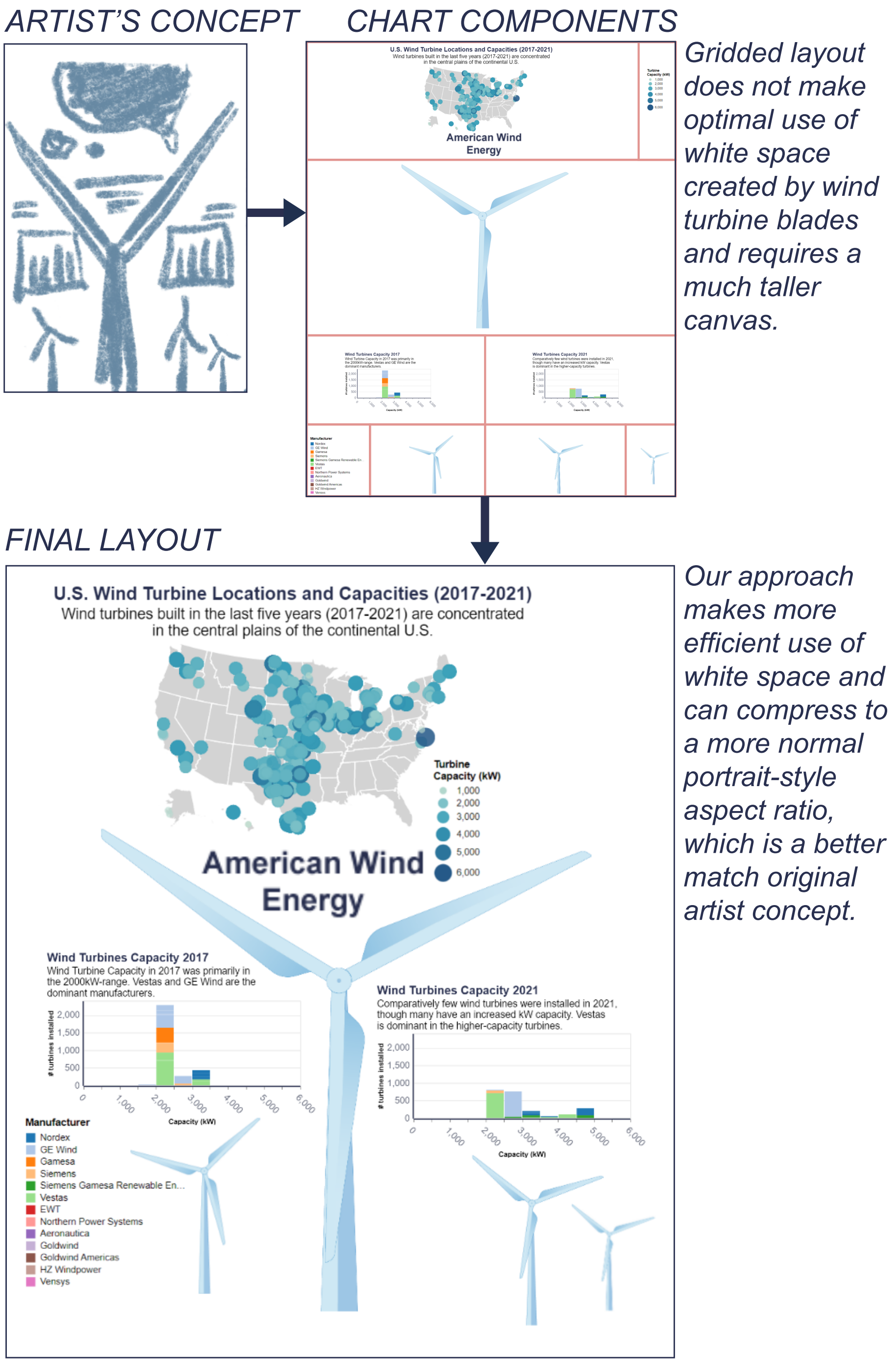
Per the design brief, the designer places the lung anatomy centrally and as the largest graphical element in the figure. To add visual interest to the infographic, they array the six respiration patterns in a slightly out-of-grid layout to fan around the outer borders of the lungs, as per the artist's concept sketch in Figure C.2. Additionally, the text stating the average normal breathing rate (12–15 breaths per minute) per the brief also requires visual emphasis, and the gap where the heart would normally rest in the white space between the lungs is an ideal position for its optimization of white space and visual interest. However, such a placement is impossible to implement in Tableau, which uses a grid-based design ignoring white space around the graphical assets, per Figure C.2, top right. These adjustments would typically need to be implemented in Adobe Illustrator or similar software. While creating such a layout is reasonably low-effort once, placement and alignment becomes tedious with multiple design iterations, particularly if design elements change. For example, an interactive version of this infographic may require the average respiration rate text to change with selection of particular respiration patterns. This requires minor layout tweaks in each iteration that rapidly become tedious for the designer. Our method enables the graphs of respiration patterns to align smoothly along the margins of the lungs in the central graphic. It furthermore is able to move the average respiration text into the white space between the lungs to add visual interest and emphasis to this information.

C.5.2 Wind Turbine Distribution in the US

This case study illustrates the value of our method when creating layouts that use the white space surrounding highly irregular-shaped assets. In this instance, the designer has been tasked with creating a visually-engaging poster infographic explaining the distribution of wind turbines across the US. The brief requires hand-drawn vector graphics of wind turbines alongside charts describing wind turbine geographic distribution and power capacity, as well as charts describing changes in power capacity and dominant manufacturers from 2017 to 2021.

Wind turbines are large, irregularly-shaped objects, with large amounts of white space between the turbine blades of the central, large wind turbine that anchors the other design elements of the infographic. To add visual interest and economize white space, the designer wishes to nest the data visualizations in these white spaces, as in the artist's concept shown in Figure C.3. Such use of white space normally requires manual layout creation in Adobe Illustrator or similar artistic programs. The closest achievable layout in a grid-based system is illustrated in the top right image of Figure C.3. There are several instances of inefficient use of white space in this layout. First, the title of the infographic cannot efficiently nest between the blades of the central turbine. Second, the width of the bounding box around the central turbine is dictated by the span of the turbine blades, where there is insufficient space on either side to draw the bar charts and companion smaller wind turbine graphical elements. These four elements are not able to nest below the turbine blades and thereby make more efficient use of the white space in the lower half of the infographic. The canvas must be taller to accommodate the two smaller windmills and bar charts, which are pushed below the central wind turbine element.

Our content-driven approach generates more compact layouts for positioning irregularly-



Gridded layout does not make optimal use of white space created by wind turbine blades and requires a much taller canvas.

Our approach makes more efficient use of white space and can compress to a more normal portrait-style aspect ratio, which is a better match original artist concept.

Figure C.3: Our content-driven layout enables plots and text elements to array around the large, central wind turbine to optimize use of white space and more closely match the artist's intended layout.

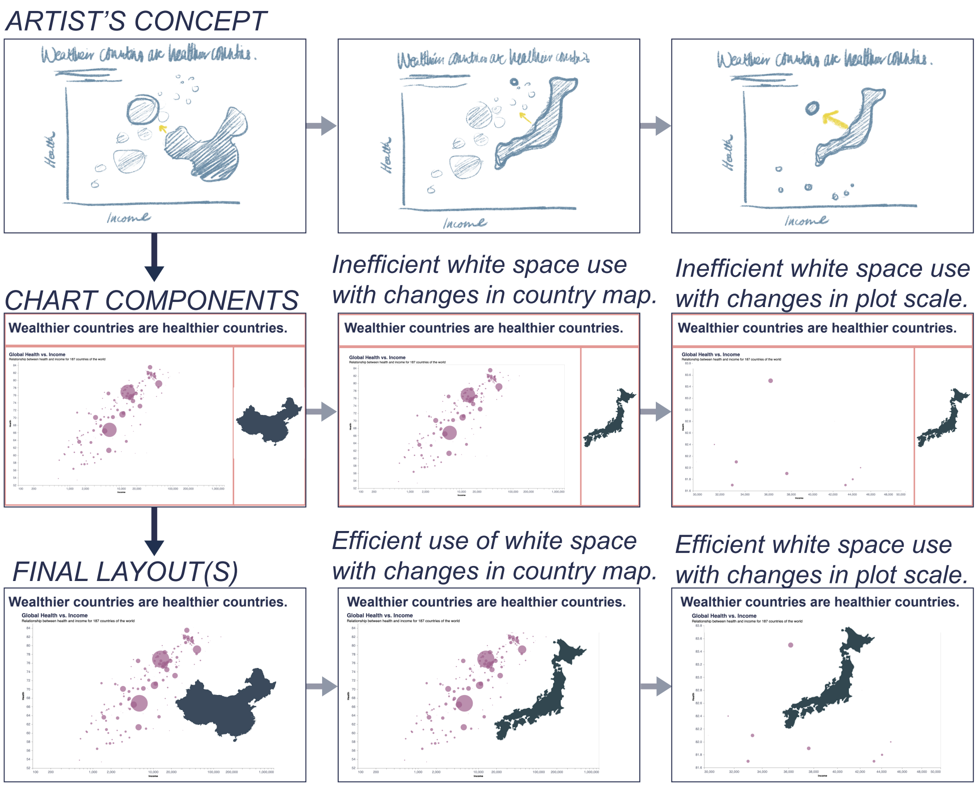

ARTIST'S CONCEPT
Wealthier countries are healthier countries.
Health
Income
CHART COMPONENTS
Inefficient white space use with changes in country map.
Inefficient white space use with changes in plot scale.
Wealthier countries are healthier countries.
Global Health vs. Income
Relationship between health and income for 187 countries of the world
FINAL LAYOUT(S)
Efficient use of white space with changes in country map.
Efficient white space use with changes in plot scale.

Figure C.4: Concept for visualization of health vs. wealth of countries of the world. While a grid layout is unable to place a country map element near the data points, our content-driven approach places the map element in an optimal position. This concept shows efficient use of white space in (1) changing elements and (2) changing white space.

shaped graphics with a graphically-interesting use of white space underneath the large wind turbine blades. The infographic title, “American Wind Energy” tucks into the space immediately above the central wind turbine, which enables enlargement of the wind turbine distribution map. The two bar charts describing wind turbine capacity and manufacturer for the years 2017 and 2018, with their companion graphical wind turbine elements, can move close to either side of the turbine pole, which enables the entire canvas to take on the standard portrait aspect ratio dimensions that the designer originally intended.

C.5.3 Health vs. Wealth in the Countries of the World

Our final case study considers the optimization of white space under two changing conditions in a visualization: (1) changing the accompanying image element in a visualization and (2) changing the scales of the axes of a plot that leads to a different distribution of white space. This design brief is for a visualization plotting the health against wealth of 187 countries of the world, from the GapMinder dataset. To add visual interest, the brief requests each data point to include a map of the country.

https://www.gapminder.org/data/

This design is centered around a bubble plot that describes the correlation of countries' health and wealth. Each bubble represents a country, and its size is encoded to the population of that country. Hovering over a bubble reveals a tooltip of information for that country, as well as a map of the country. The designer furthermore wishes to have this map element appear as close to the associated country bubble as possible, without obscuring surrounding data points in the bubble plot. A complication to this brief is that the plot scales can be adjusted to zoom in to different regions of the plot, which requires a different layout for the map element depending on the changing white space.

The designer's concept is shown in the top of Figure C.4 for inspection of China and Japan, and a subsequent alteration of the plot scale to look closer at only the healthiest and wealthiest countries, including Japan. The design in these three instances, first with the change of image element and second with the change of plot scale and subsequent bubble positioning, takes advantage of the extensive white space around the bubble plot to nest maps closely around the data elements. Unfortunately, this layout is not achievable in a standard grid-based layout system, which is illustrated in the middle row of Figure C.4. Here, the maps must be much smaller and further away from their associated data point. This is an inefficient use of white space, and forces the viewer to look back and forth between associated elements.

Our content-driven approach enables the country map element to slide in near the data point of interest to reduce white space in the visualization, and to reduce the number of places the user must focus their attention on in the visualization. In the first scenario at the bottom left two images of Figure C.4, the differently-shaped country maps of China and Japan each are able to position efficiently in the white space in the lower right of the bubble plot. When adjusting the plot scales to show only the healthiest and wealthiest countries (bottom rightmost), our algorithm positions the Japan map element more optimally in the upper right of the bubble plot. This case study thus demonstrates our algorithm's capabilities both in conditions of (1) changing elements and (2) changing white space.

C.6 Discussion and Limitations

As demonstrated in the presented case studies, our approach is capable of transform a grid layout that makes inefficient use of white space into a content-driven layout that can closely mirror the artist's original design concept and more efficiently use the white space in the visualization. Our approach is targeted at scenarios with relatively few elements (i.e., tens), which is typical for common infographics, and hence is not well-suited for the layout of large data collections [32].

While the method presented in this paper primarily focuses on turning grid layouts into content-driven layouts, it could greatly benefit from being combined with existing works and approaches. Our force-directed layout setup could be used on other initial layouts with irregular shapes, as long as it is possible to infer the layout topology as a graph. Our prototype starts from a fully-specified grid layout and specific screen size, which introduces some instability to the layout. However, using interaction to incrementally place or move shapes could help tweak and overcome such instabilities.

Although we found that our approach delivers good results even with distance transforms computed at lower resolutions, there may be instances where this fails to take into account details that may be relevant for the final layout. In such cases, an additional pre-processing step could be used (e.g., a low-pass filter) to make sure that all important features are captured. Alternatively, or additionally, the input to the distance transform could be adapted according to the semantics of individual components of the visualization, e.g., by giving higher priority to data-encoding pixels compared to legends or other decorations. Such extensions could be easily integrated into our approach by automatically applying different styles for layout and display purposes to the Vega-Lite specification. Similarly, object detection techniques could be leveraged to identify salient regions of images or charts and separate it from the background, before generating the distance transform. Furthermore, image-space edits could be done to the image before the distance transform is computed. For example, drawing a line on the right side of an image before computing the distance field would create a hard boundary on the right side of that image in the layout simulation, and such manipulations could be used as additional alignment guides. Other types of interactions could also be integrated in order to further increase the degree of design freedom. While currently our approach is based on an automatic identification of central elements, this initial selection could be modified by the user in order to provide a more tailored user experience as discussed by Tyagi et al. [103].

While our current prototype implementation already allows for basic interactions when supplied with corresponding Vega-Lite specifications, we do not yet support fully dynamic content (e.g., interactive filtering or cross-linking between multiple views), partly due to the fact that it has proven difficult to implement event translation consistently across different browsers. However, we plan to extend this functionality in the future. We plan to evaluate our approach through a user study or a larger set of existing grid layouts and element types. Here, in particular, it will be interesting to study differences related to the design expertise of individual users.

C.7 Conclusion

In this paper we presented an approach for turning an existing grid layout into a content-driven layout, wherein elements are positioned by their contents rather than their proxy bounding geometries. Our method parses and transforms the original grid layout topology into a smaller graph which informs the selective application of attractive forces and leverages distance transforms to repulse elements based on their content to enable better space utilization. Furthermore, we presented three case studies demonstrating the effectiveness and versatility of our algorithm. In the future, our approach to repulsing irregular shapes can be applied to other use cases, in combination with other image-based techniques.

Bibliography

[1] https://www.kaggle.com/hesh97/titanicdataset-traincsv, 2018. Accessed: 2019-11-07.

[2] AHLBERG, C., AND WISTRAND, E. IVEE: An environment for automatic creation of dynamic queries applications. In *Proc. ACM CHI* (1995), pp. 15–16, doi: 10.1145/223355.223381.

[3] AIKEN, A., CHEN, J., STONEBRAKER, M., AND WOODRUFF, A. Tioga-2: a direct manipulation database visualization environment. In *Proc. International Conference on Data Engineering* (1996), pp. 208–217, doi: 10.1109/ICDE.1996.492109.

[4] ALBANO, A., AND SAPUPPO, G. Optimal allocation of two-dimensional irregular shapes using heuristic search methods. *IEEE Trans. Systems, Man, and Cybernetics 10*, 5 (1980), 242–248, doi: 10.1109/TSMC.1980.4308483.

[5] ALI, K., HARTMANN, K., FUCHS, G., AND SCHUMANN, H. Adaptive layout for interactive documents. In *Proc. International Symposium on Smart Graphics* (2008), pp. 247–254, doi: 10.1007/978-3-540-85412-$8_2$4.

[6] ANDREWS, K., AND SMRDEL, A. Responsive data visualisation. In *Proc. EuroVis (Posters)* (2017), pp. 113–115, doi: 10.2312/eurp.20171182.

[7] ANGELINI, M., SANTUCCI, G., SCHUMANN, H., AND SCHULZ, H.-J. A review and characterization of progressive visual analytics. *Informatics 5* (2018), doi: 10.3390/informatics5030031.

[8] BAUDEL, T., AND BROEKSEMA, B. Capturing the design space of sequential space-filling layouts. *IEEE Trans. Visualization and Computer Graphics 18*, 12 (2012), 2593–2602, doi: 10.1109/TVCG.2012.205.

[9] BEHRISCH, M., BLUMENSCHEIN, M., KIM, N. W., SHAO, L., EL-ASSADY, M., FUCHS, J., SEEBACHER, D., DIEHL, A., BRANDES, U., PFISTER, H., ET AL. Quality metrics for information visualization. *Computer Graphics Forum 37*, 3 (2018), 625–662, doi: doi.org/10.1111/cgf.13446.

[10] BERTIN, J. *Semiology of Graphics*. University of Wisconsin Press, 1983.

[11] BOLTE, F., AND BRUCKNER, S. Measures in visualization space. In *Foundations of Data Visualization*. Springer, 2020, pp. 39–59, doi: 10.1007/978-3-030-34444-3_3.

[12] BORGEFORS, G. Distance transformations in digital images. *Computer Vision, Graphics, and Image Processing 34*, 3 (1986), 344–371, doi: 10.1016/S0734-189X(86)80047-0.

[13] BOSTOCK, M., AND HEER, J. Protovis: A graphical toolkit for visualization. *IEEE Trans. Visualization and Computer Graphics 15*, 6 (2009), 1121–1128, doi: 10.1109/TVCG.2009.174.

[14] BOSTOCK, M., OGIEVETSKY, V., AND HEER, J. D^3: data-driven documents. *IEEE Trans. Computer Graphics and Visualization 17*, 12 (2011), 2301–2309, doi: 10.1109/TVCG.2011.185.

[15] BOUGANIS, A., AND SHANAHAN, M. A vision-based intelligent system for packing 2-d irregular shapes. *IEEE Trans. Automation Science and Engineering 4*, 3 (2007), 382–394, doi: 10.1109/TASE.2006.887158.

[16] BRINKMANN, G. G., RIETVELD, K. F., AND TAKES, F. W. Exploiting gpus for fast force-directed visualization of large-scale networks. In *Proc. International Conference on Parallel Processing* (2017), pp. 382–391, doi: 10.1109/ICPP.2017.47.

[17] BRULS, M., HUIZING, K., AND WIJK, J. J. V. Squarified treemaps. In *Data visualization 2000*. Springer, 2000, pp. 33–42, doi: 10.1007/978-3-7091-6783-0$_4$.

[18] CANTU, A., GRISVARD, O., DUVAL, T., AND COPPIN, G. Identifying the relationships between the visualization context and representation components to enable recommendations for designing new visualizations. In *Proc. International Conference on Information Visualisation* (2017), pp. 20–28, doi: 10.1109/iV.2017.55.

[19] CASSELMAN, B. `https://github.com/fivethirtyeight/guns-data`, 2016. Accessed: 2019-11-07.

[20] CHEN, X., ZENG, W., LIN, Y., AL-MANEEA, H. M., ROBERTS, J., AND CHANG, R. Composition and configuration patterns in multiple-view visualizations. *IEEE Trans. Visualization and Computer Graphics 27*, 2 (2021), 1514–1524, doi: 10.1109/TVCG.2020.3030338.

[21] CHERNOBELSKIY, R., CUNNINGHAM, K. I., GOODRICH, M. T., KOBOUROV, S. G., AND TROTT, L. Force-directed lombardi-style graph drawing. In *Proc. International Symposium on Graph Drawing* (2011), pp. 320–331, doi: 10.1007/978-3-642-25878-7$_3$1.

[22] CLEVELAND, W. S., AND MCGILL, R. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American Statistical Association 79*, 387 (1984), 531–554, doi: 10.1080/01621459.1984.10478080.

[23] CORRELL, M., LI, M., KINDLMANN, G., AND SCHEIDEGGER, C. Looks good to me: Visualizations as sanity checks. *IEEE Trans. Visualization and Computer Graphics 25*, 1 (2018), 830–839, doi: 10.1109/TVCG.2018.2864907.

[24] DAYAMA, N. R., SANTALA, S., BRÜCKNER, L., TODI, K., DU, J., AND OULASVIRTA, A. Interactive layout transfer. In *Proc. International Conference on Intelligent User Interfaces* (2021), pp. 70–80, doi: 10.1145/3397481.3450652.

[25] DENGLER, E., FRIEDELL, M., AND MARKS, J. Constraint-driven diagram layout. In *Proc. IEEE Symposium on Visual Languages* (1993), pp. 330–335, doi: 10.1109/VL.1993.269619.

[26] DIBIA, V., AND DEMIRALP, Ç. Data2vis: Automatic generation of data visualizations using sequence-to-sequence recurrent neural networks. *IEEE Computer Graphics and Applications 39*, 5 (2019), 33–46, doi: 10.1109/MCG.2019.2924636.

[27] DIEHL, A., ABDUL-RAHMAN, A., EL-ASSADY, M., BACH, B., KEIM, D. A., AND CHEN, M. Visguides: A forum for discussing visualization guidelines. In *Proc. EuroVis (Short Papers)* (2018), pp. 61–65, doi: 10.2312/eurovisshort.20181079.

[28] ELMQVIST, N., DO, T.-N., GOODELL, H., HENRY, N., AND FEKETE, J.-D. ZAME: Interactive large-scale graph visualization. In *Proc. IEEE PacificVis* (2008), pp. 215–222, doi: 10.1109/PACIFICVIS.2008.4475479.

[29] ELSHEHALY, M., RANDELL, R., BREHMER, M., MCVEY, L., ALVARADO, N., GALE, C. P., AND RUDDLE, R. A. Qualdash: Adaptable generation of visualisation dashboards for healthcare quality improvement. *IEEE Trans. Visualization and Computer Graphics 27*, 2 (2020), 689–699, doi: 10.1109/TVCG.2020.3030424.

[30] ENGELKE, U., ABDUL-RAHMAN, A., AND CHEN, M. Visupply: A supply-chain process model for visualization guidelines. In *Proc. International Symposium on Big Data Visual and Immersive Analytics* (2018), pp. 1–9, doi: 10.1109/BDVA.2018.8534029.

[31] FEINER, S. A grid-based approach to automating display layout. In *Proc. Graphics Interface* (1988), pp. 192–197.

[32] FREY, S. Optimizing grid layouts for level-of-detail exploration of large data collections. *Computer Graphics Forum 41* (2022), doi: 10.1111/cgf.14537.

[33] FRUCHTERMAN, T. M., AND REINGOLD, E. M. Graph drawing by force-directed placement. *Software: Practice and Experience 21*, 11 (1991), 1129–1164, doi: 10.1002/spe.4380211102.

[34] GENC, B., AND DOGRUSOZ, U. A constrained, force-directed layout algorithm for biological pathways. In *Proc. International Symposium on Graph Drawing* (2003), pp. 314–319, doi: 10.1007/978-3-540-24595-$7_2$9.

[35] GLEICHER, M., ALBERS, D., WALKER, R., JUSUFI, I., HANSEN, C. D., AND ROBERTS, J. C. Visual comparison for information visualization. *Information Visualization 10*, 4 (2011), 289–309, doi: 10.1177/1473871611416549.

[36] GRAMMEL, L., TORY, M., AND STOREY, M.-A. How information visualization novices construct visualizations. *IEEE Trans. Visualization and Computer Graphics 16*, 6 (2010), 943–952, doi: 10.1109/TVCG.2010.164.

[37] GRATZL, S., GEHLENBORG, N., LEX, A., PFISTER, H., AND STREIT, M. Domino: Extracting, comparing, and manipulating subsets across multiple tabular datasets. *IEEE Trans. Visualization and Computer Graphics 20*, 12 (2014), 2023–2032, doi: 10.1109/TVCG.2014.2346260.

[38] HARATY, M., NOBARANY, S., DIPAOLA, S., AND FISHER, B. D. Adwil: adaptive windows layout manager. In *Proc. ACM CHI*. 2009, pp. 4177–4182, doi: 10.1145/1520340.1520636.

[39] HEER, J., CARD, S. K., AND LANDAY, J. A. Prefuse: A toolkit for interactive information visualization. In *Proc. ACM CHI* (2005), pp. 421–430, doi: 10.1145/1054972.1055031.

[40] HENRY, N., AND FEKETE, J.-D. NodeTrix: a hybrid visualization of social networks. *IEEE Trans. Visualization and Computer Graphics 13*, 6 (2007), 1302–1309, doi: 10.1109/TVCG.2007.70582.

[41] HOFFSWELL, J., LI, W., AND LIU, Z. Techniques for flexible responsive visualization design. In *Proc. ACM CHI* (2020), pp. 1–13, doi: 10.1145/3313831.3376777.

[42] HORAK, T., MATHISEN, A., KLOKMOSE, C. N., DACHSELT, R., AND ELMQVIST, N. Vistribute: Distributing interactive visualizations in dynamic multi-device setups. In *Proc. ACM CHI* (2019), pp. 1–13, doi: 10.1145/3290605.3300846.

[43] HUDSON, S. E. Adaptive semantic snaping—a technique for semantic feedback at the lexical level. In *Proc. ACM CHI* (1990), pp. 65–70, doi: 10.1145/97243.97253.

[44] ISHAK, E. W., AND FEINER, S. Content-aware layout. In *Proc. ACM CHI* (2007), pp. 2459—-2464, doi: 10.1145/1240866.1241024.

[45] JACOBS, C., LI, W., SCHRIER, E., BARGERON, D., AND SALESIN, D. Adaptive grid-based document layout. *ACM Trans. Graphics 22*, 3 (2003), 838–847, doi: 10.1145/882262.882353.

[46] JACOMY, M., VENTURINI, T., HEYMANN, S., AND BASTIAN, M. Forceatlas2, a continuous graph layout algorithm for handy network visualization designed for the gephi software. *PloS one 9*, 6 (2014), e98679, doi: 10.1371/journal.pone.0098679.

[47] JAHANIAN, A., LIU, J., LIN, Q., TRETTER, D., O'BRIEN-STRAIN, E., LEE, S. C., LYONS, N., AND ALLEBACH, J. Recommendation system for automatic design of magazine covers. In *Proc. International Conference on Intelligent User Interfaces* (2013), pp. 95–106, doi: 10.1145/2449396.2449411.

[48] JAHANIAN, A., LIU, J., TRETTER, D. R., LIN, Q., DAMERA-VENKATA, N., O'BRIEN-STRAIN, E., LEE, S., FAN, J., AND ALLEBACH, J. P. Automatic design of magazine covers. In *Proc. Imaging and Printing in a Web 2.0 World III* (2012), pp. 114–121, doi: 10.1117/12.914596.

[49] JAVED, W., AND ELMQVIST, N. Exploring the design space of composite visualization. In *Proc. IEEE PacificVis* (2012), pp. 1–8, doi: 10.1109/PacificVis.2012.6183556.

[50] KANDEL, S., HEER, J., PLAISANT, C., KENNEDY, J., VAN HAM, F., RICHE, N. H., WEAVER, C., LEE, B., BRODBECK, D., AND BUONO, P. Research directions in data wrangling: Visualizations and transformations for usable and credible data. *Information Visualization 10*, 4 (2011), 271–288, doi: 10.1177/1473871611415994.

[51] KIM, H., MORITZ, D., AND HULLMAN, J. Design patterns and trade-offs in responsive visualization for communication. *Computer Graphics Forum 40*, 3 (2021), 459–470, doi: 10.1111/cgf.14321.

[52] KIM, N. W., SCHWEICKART, E., LIU, Z., DONTCHEVA, M., LI, W., POPOVIC, J., AND PFISTER, H. Data-driven guides: Supporting expressive design for information graphics. *IEEE Trans. Visualization and Computer Graphics 23*, 1 (2017), 491–500, doi: 10.1109/TVCG.2016.2598620.

[53] KIM, T. `http://tany.kim/best-bookshelf`, 2019. Accessed: 2019-11-07.

[54] KINDLMANN, G., AND SCHEIDEGGER, C. An algebraic process for visualization design. *IEEE Trans. Visualization and Computer Graphics 20*, 12 (2014), 2181–2190, doi: 10.1109/TVCG.2014.2346325.

[55] KOSSLYN, S. M. Understanding charts and graphs. *Applied cognitive psychology 3*, 3 (1989), 185–225, doi: 10.1002/acp.2350030302.

[56] KRISTIANSEN, Y. S., AND BRUCKNER, S. Visception: An interactive visual framework for nested visualization design. *Computers & Graphics 92* (2020), 13–27, doi: 10.1016/j.cag.2020.08.007.

[57] LEBLANC, J., WARD, M. O., AND WITTELS, N. Exploring n-dimensional databases. In *Proc. IEEE Visualization* (1990), pp. 230–237, doi: 10.1109/VISUAL.1990.146386.

[58] LI, G., TIAN, M., XU, Q., MCGUFFIN, M. J., AND YUAN, X. Gotree: A grammar of tree visualizations. In *Proc. ACM CHI* (2020), p. 1–13, doi: 10.1145/3313831.3376297.

[59] LI, J., YANG, J., HERTZMANN, A., ZHANG, J., AND XU, T. Layoutgan: Generating graphic layouts with wireframe discriminators. *arXiv preprint arXiv:1901.06767 abs/1901.06767* (2019), doi: 10.48550/arXiv.1901.06767.

[60] LIN, H., MORITZ, D., AND HEER, J. Dziban: Balancing agency & automation in visualization design via anchored recommendations. In *Proc. ACM CHI* (2020), pp. 1–12, doi: 10.1145/3313831.3376880.

[61] LIU, Z., THOMPSON, J., WILSON, A., DONTCHEVA, M., DELOREY, J., GRIGG, S., KERR, B., AND STASKO, J. Data illustrator: Augmenting vector design tools with lazy data binding for expressive visualization authoring. In *Proc. ACM CHI* (2018), pp. 123:1–123:13, doi: 10.1145/3173574.3173697.

[62] LOK, S., AND FEINER, S. A survey of automated layout techniques for information presentations. *Proc. International Symposium on Smart Graphics 2001* (2001), 61–68.

[63] LOORAK, M. H., PERIN, C., COLLINS, C., AND CARPENDALE, S. Exploring the possibilities of embedding heterogeneous data attributes in familiar visualizations. *IEEE Trans. Visualization and Computer Graphics 23*, 1 (2017), 581–590, doi: 10.1109/TVCG.2016.2598586.

[64] MACKINLAY, J. Automating the design of graphical presentations of relational information. *ACM Trans. Graphics 5*, 2 (1986), 110–141, doi: 10.1145/22949.22950.

[65] MACKINLAY, J., HANRAHAN, P., AND STOLTE, C. Show me: Automatic presentation for visual analysis. *IEEE Trans. Visualization and Computer Graphics 13*, 6 (2007), 1137–1144, doi: 10.1109/TVCG.2007.70594.

[66] McNutt, A. What are table cartograms good for anyway? an algebraic analysis. *Computer Graphics Forum 40*, 3 (2021), 61–73, doi: 10.1111/cgf.14289.

[67] McNutt, A., and Kindlmann, G. Linting for visualization: Towards a practical automated visualization guidance system. In *Proc. IEEE VIS Workshop on the Creation, Curation, Critique and Conditioning of Principles and Guidelines in Visualization* (2018).

[68] Meijster, A., Roerdink, J. B., and Hesselink, W. H. A general algorithm for computing distance transforms in linear time. In *Proc. Mathematical Morphology and its Applications to Image and Signal Processing*. 2002, pp. 331–340, doi: 10.1007/0-306-47025-X36.

[69] Micallef, L., and Rodgers, P. eulerforce: Force-directed layout for euler diagrams. *Journal of Visual Languages & Computing 25*, 6 (2014), 924–934, doi: 10.1016/j.jvlc.2014.09.002.

[70] Mike Bostock, Shan Carter, M. E. At the national conventions, the words they used, 2012. Accessed: 2019-11-07, `https://archive.nytimes.com/www.nytimes.com/interactive/2012/09/06/us/politics/convention-word-counts.html`.

[71] Minard, C.-J. Carte figurative des pertes successives en hommes de l'armée francais dans la campagne de russe.

[72] Moritz, D., Wang, C., Nelson, G., Lin, H., Smith, A. M., Howe, B., and Heer, J. Formalizing visualization design knowledge as constraints: Actionable and extensible models in draco. *IEEE Trans. Visualization and Computer Graphics 25*, 1 (2019), 438–448, doi: 10.1109/TVCG.2018.2865240.

[73] Munzner, T., and Maguire, E. *Visualization Analysis and Design*. CRC Press, 2015, doi: 10.1201/b17511.

[74] Nacenta, M. A., and Méndez, G. G. iVoLVER: A visual language for constructing visualizations from in-the-wild data. In *Proc. ACM International Conference on Interactive Surfaces and Spaces* (2017), pp. 438–441, doi: 10.1145/3132272.3132299.

[75] Ondov, B., Jardine, N., Elmqvist, N., and Franconeri, S. Face to face: Evaluating visual comparison. *IEEE Trans. Visualization and Computer Graphics 25*, 1 (2019), 861–871, doi: 10.1109/TVCG.2018.2864884.

[76] Park, D., Drucker, S. M., Fernandez, R., and Elmqvist, N. Atom: A grammar for unit visualizations. *IEEE Trans. Visualization and Computer Graphics 24*, 12 (2018), 3032–3043, doi: 10.1109/TVCG.2017.2785807.

[77] Parker, G., Franck, G., and Ware, C. Visualization of large nested graphs in 3D: Navigation and interaction. *Journal of Visual Languages and Computing 9*, 3 (1998), 299–317, doi: 10.1006/jvlc.1998.0086.

[78] Pattison, T., Vernik, R., and Phillips, M. Information visualisation using composable layouts and visual sets. In *Proc. Asia-Pacific Symposium on Information Visualisation* (2001), pp. 1—10, `https://dl.acm.org/doi/10.5555/564040.564041`.

[79] QU, Z., AND HULLMAN, J. Evaluating visualization sets: Trade-offs between local effectiveness and global consistency. In *Proc. BELIV* (2016), pp. 44–52, doi: 10.1145/2993901.2993910.

[80] QU, Z., AND HULLMAN, J. Keeping multiple views consistent: Constraints, validations, and exceptions in visualization authoring. *IEEE Trans. Visualization and Computer Graphics 24*, 1 (2018), 468–477, doi: 10.1109/TVCG.2017.2744198.

[81] REN, D., HÖLLERER, T., AND YUAN, X. iVisDesigner: Expressive interactive design of information visualizations. *IEEE Trans. Visualization and Computer Graphics 20*, 12 (2014), 2092–2101, doi: 10.1109/TVCG.2014.2346291.

[82] REN, D., LEE, B., AND BREHMER, M. Charticulator: Interactive construction of bespoke chart layouts. *IEEE Trans. Visualization and Computer Graphics 25*, 1 (2019), 789–799, doi: 10.1109/TVCG.2018.2865158.

[83] ROTH, S. F., LUCAS, P., SENN, J. A., GOMBERG, C. C., BURKS, M. B., STROFFOLINO, P. J., KOLOJECHICK, A. J., AND DUNMIRE, C. Visage: a user interface environment for exploring information. In *Proc. IEEE InfoVis* (1996), pp. 3–12, doi: 10.1109/INFVIS.1996.559210.

[84] SARIKAYA, A., CORRELL, M., BARTRAM, L., TORY, M., AND FISHER, D. What do we talk about when we talk about dashboards? *IEEE Trans. Visualization and Computer Graphics 25*, 1 (2018), 682–692, doi: 10.1109/TVCG.2018.2864903.

[85] SATYANARAYAN, A., AND HEER, J. Lyra: An interactive visualization design environment. *Computer Graphics Forum 33*, 3 (2014), 351–360, doi: 10.1111/cgf.12391.

[86] SATYANARAYAN, A., LEE, B., REN, D., HEER, J., STASKO, J., THOMPSON, J., BREHMER, M., AND LIU, Z. Critical reflections on visualization authoring systems. *IEEE Trans. Visualization and Computer Graphics 26*, 1 (2020), 461–471, doi: 10.1109/TVCG.2019.2934281.

[87] SATYANARAYAN, A., MORITZ, D., WONGSUPHASAWAT, K., AND HEER, J. Vega-lite: A grammar of interactive graphics. *IEEE Trans. Visualization and Computer Graphics 23*, 1 (2017), 341–350, doi: 10.1109/TVCG.2016.2599030.

[88] SCHLIMMER, J. `https://archive.ics.uci.edu/ml/datasets/mushroom`, 1987. Accessed: 2019-11-07.

[89] SCHRIER, E., DONTCHEVA, M., JACOBS, C., WADE, G., AND SALESIN, D. Adaptive layout for dynamically aggregated documents. In *Proc. International Conference on Intelligent User Interfaces* (2008), pp. 99–108, doi: 10.1145/1378773.1378787.

[90] SCHULZ, H., AKBAR, Z., AND MAURER, F. A generative layout approach for rooted tree drawings. In *Proc. IEEE PacificVis* (2013), pp. 225–232, doi: 10.1109/PacificVis.2013.6596149.

[91] SCHULZ, H., HADLAK, S., AND SCHUMANN, H. The design space of implicit hierarchy visualization: A survey. *IEEE Trans. Visualization and Computer Graphics 17*, 4 (2011), 393–411, doi: 10.1109/TVCG.2010.79.

[92] SCHULZ, H.-J., AND HADLAK, S. Preset-based generation and exploration of visualization designs. *Journal of Visual Languages And Computing 31* (2015), 9–29, doi: 10.1016/j.jvlc.2015.09.004.

[93] SHADOAN, R., AND WEAVER, C. Visual analysis of higher-order conjunctive relationships in multidimensional data using a hypergraph query system. *IEEE Trans. Visualization and Computer Graphics 19*, 12 (2013), 2070–2079, doi: 10.1109/TVCG.2013.220.

[94] SILVA, S., MADEIRA, J., AND SANTOS, B. S. There is more to color scales than meets the eye: a review on the use of color in visualization. In *Proc. International Conference on Information Visualization* (2007), pp. 943–950, doi: 10.1109/IV.2007.113.

[95] SLINGSBY, A., DYKES, J., AND WOOD, J. Configuring hierarchical layouts to address research questions. *IEEE Trans. Visualization and Computer Graphics 15*, 6 (2009), 977–984, doi: 10.1109/TVCG.2009.128.

[96] STEINBERGER, M., WALDNER, M., AND SCHMALSTIEG, D. Interactive self-organizing windows. *Computer Graphics Forum 31*, 2pt3 (2012), 621–630, doi: 10.1111/j.1467-8659.2012.03041.x.

[97] STOLTE, C., TANG, D., AND HANRAHAN, P. Polaris: a system for query, analysis, and visualization of multidimensional databases. *Communications of the ACM 51*, 11 (2008), 75–84, doi: 10.1109/2945.981851.

[98] STÖRRLE, H. On the impact of layout quality to understanding uml diagrams: Diagram type and expertise. In *Proc. IEEE Symposium on Visual Languages and Human-Centric Computing* (2012), pp. 49–56, doi: 10.1109/VLHCC.2012.6344480.

[99] SWEARNGIN, A., WANG, C., OLESON, A., FOGARTY, J., AND KO, A. J. Scout: Rapid exploration of interface layout alternatives through high-level design constraints. In *Proc. ACM CHI* (2020), pp. 1–13, doi: 10.1145/3313831.3376593.

[100] TODI, K., WEIR, D., AND OULASVIRTA, A. Sketchplore: Sketch and explore with a layout optimiser. In *Proc. ACM Conference on Designing Interactive Systems* (2016), pp. 543–555, doi: 10.1145/2901790.2901817.

[101] TUFTE, E. *Envisioning Information*. Graphics Press, 1990.

[102] TUFTE, E. R. *The Visual Display of Quantitative Information*. Graphics Press, 1986.

[103] TYAGI, A., ZHAO, J., PATEL, P., KHURANA, S., AND MUELLER, K. Infographics wizard: Flexible infographics authoring and design exploration. *Computer Graphics Forum 41*, 3 (2022), 121–132, doi: https://doi.org/10.1111/cgf.14527.

[104] VAN DEN ELZEN, S., AND VAN WIJK, J. J. Small multiples, large singles: A new approach for visual data exploration. *Computer Graphics Forum 32*, 3 (2013), 191–200, doi: 10.1111/cgf.12106.

[105] VUILLEMOT, R., AND BOY, J. Structuring visualization mock-ups at the graphical level by dividing the display space. *IEEE Trans. Visualization and Computer Graphics 24*, 1 (2018), 424–434, doi: 10.1109/TVCG.2017.2743998.

[106] WANG, W., WANG, H., DAI, G., AND WANG, H. Visualization of large hierarchical data by circle packing. In *Proc. ACM CHI* (2006), pp. 517–520, doi: 10.1145/1124772.1124851.

[107] WICKHAM, H., AND HOFMANN, H. Product plots. *IEEE Trans. Visualization and Computer Graphics 17*, 12 (2011), 2223–2230, doi: 10.1109/TVCG.2011.227.

[108] WILKINSON, L. *The Grammar of Graphics*. Springer-Verlag New York, 2005, doi: 10.1007/0-387-28695-0.

[109] WILKINSON, L., ANAND, A., AND GROSSMAN, R. Graph-theoretic scagnostics. In *Proc. IEEE Symposium on Information Visualization* (2005), pp. 157–164, doi: 10.1109/INFOVIS.2005.14.

[110] WONGSUPHASAWAT, K., MORITZ, D., ANAND, A., MACKINLAY, J., HOWE, B., AND HEER, J. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *IEEE Trans. Visualization and Computer Graphics 22*, 1 (2015), 649–658, doi: 10.1109/TVCG.2015.2467191.

[111] WONGSUPHASAWAT, K., MORITZ, D., ANAND, A., MACKINLAY, J., HOWE, B., AND HEER, J. Towards a general-purpose query language for visualization recommendation. In *Proc. Workshop on Human-In-the-Loop Data Analytics* (2016), pp. 4:1–4:6, doi: 10.1145/2939502.2939506.

[112] WONGSUPHASAWAT, K., MORITZ, D., SATYANARAYAN, A., AND HEER, J. Vega: A visualization grammar. `https://vega.github.io/`, 2013.

[113] WONGSUPHASAWAT, K., QU, Z., MORITZ, D., CHANG, R., OUK, F., ANAND, A., MACKINLAY, J., HOWE, B., AND HEER, J. Voyager 2: Augmenting visual analysis with partial view specifications. In *Proc. ACM CHI* (2017), pp. 2648–2659, doi: 10.1145/3025453.3025768.

[114] XU, P., FU, H., IGARASHI, T., AND TAI, C.-L. Global beautification of layouts with interactive ambiguity resolution. In *Proc. ACM Symposium on User Interface Software and Technology* (2014), pp. 243–252, doi: 10.1145/2642918.2647398.

[115] YANG, X., MEI, T., XU, Y.-Q., RUI, Y., AND LI, S. Automatic generation of visual-textual presentation layout. *ACM Trans. Multimedia Computing, Communications, and Applications 12*, 2 (2016), 1–22, doi: 10.1145/2818709.

[116] ZGRAGGEN, E., ZELEZNIK, R., AND DRUCKER, S. M. PanoramicData: Data analysis through pen touch. *IEEE Trans. Visualization and Computer Graphics 20*, 12 (2014), 2112–2121, doi: 10.1109/TVCG.2014.2346293.

[117] ZHENG, X., QIAO, X., CAO, Y., AND LAU, R. W. Content-aware generative modeling of graphic design layouts. *ACM Trans. Graphics 38*, 4 (2019), 1–15, doi: 10.1145/3306346.3322971.

[118] ZHU, S., SUN, G., JIANG, Q., ZHA, M., AND LIANG, R. A survey on automatic infographics and visualization recommendations. *Visual Informatics 4*, 3 (2020), 24–40, doi: 10.1016/j.visinf.2020.07.002.

[119] ZHU, Y. Measuring effective data visualization. In *Proc. International Symposium on Visual Computing* (2007), pp. 652–661, doi: /10.1007/978-3-540-76856-2$_6$4.

Graphic design: Communication Division, UIB / Print: Skipnes Kommunikasjon AS

uib.no

ISBN: 9788230850749 (print)
9788230842553 (PDF)