

Bangor University

DOCTOR OF PHILOSOPHY

Analysis, Design and Implementation of Multiple View Visualisations

Al-Maneea, Hayder Mahdi Abdullah

Award date:
2021

Awarding institution:
Bangor University

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 20. Apr. 2024



PRIFYSGOL
BANGOR
UNIVERSITY

School of Computer Science and Electronic Engineering
College of Environmental Sciences and Engineering

Analysis, Design and Implementation of Multiple View Visualisations

Hayder M. Al-maneea

Submitted in partial satisfaction of the requirements for the
Degree of Doctor of Philosophy
in Computer Science

Supervisor Prof. Jonathan C. Roberts

October 2021

Statement of Originality

The work presented in this thesis/dissertation is entirely from the studies of the individual student, except where otherwise stated. Where derivations are presented and the origin of the work is either wholly or in part from other sources, then full reference is given to the original author. This work has not been presented previously for any degree, nor is it at present under consideration by any other degree awarding body.

Student:

Hayder M. Al-maneea

Statement of Availability

I hereby acknowledge the availability of any part of this thesis/dissertation for viewing, photocopying or incorporation into future studies, providing that full reference is given to the origins of any information contained herein. I further give permission for a copy of this work to be deposited with the Bangor University Institutional Digital Repository, the British Library ETHOS system, and/or in any other repository authorised for use by Bangor University and where necessary have gained the required permissions for the use of third party material. I acknowledge that Bangor University may make the title and a summary of this thesis/dissertation freely available.

Student:

Hayder M. Al-maneea

Acknowledgements

First and foremost, I would like to express my deep and sincere gratitude to my research supervisor **Prof. Jonathan Roberts**, for giving me the opportunity to do this research and providing invaluable guidance throughout this study. His dynamism, vision, sincerity and motivation have deeply inspired me. He has taught me the methodology to carry out the research and to present the research work as clearly as possible. It was a great privilege and honour to work and study under his guidance. I am extremely grateful for what he has offered me and for his corrections of this thesis. I would also like to thank him for his friendship and empathy.

I would also like to acknowledge the Ministry of Higher Education and Scientific Research (MOHESR) of Iraq for generosity in funding my PhD study. Great thanks to Basra University for supporting and helping me during my undergraduate, master, and PhD study.

It would not have been possible to write this thesis without the help and support of the kind people around me: my lovely wife **Shatha Al-maliki** for his personal support and great patience at all times. Also, my sweet kids **Fatimah** and **Mohammad** for their kindness and they make my life full with hope without despair. As well, I would like to thank all my family in Iraq: wonderful mom, dad, sisters and brothers for this support and prayers to me.

Also, I want to thank my lovely aunt, **Sahar Al-maneea**, and her kind husband, **Hayder Kubba**, for their support during my PhD study.

Last, but not least, I would like to thank all the staff of the School of Computer Science and Electronic Engineering at Bangor University for their help during these years.

Abstract

Multiple view systems are often used by visualisation developers. They are useful for displaying or interpreting data in several, multiple or parallel ways. One of the reasons developers use such duplication is to help clarify the information. Perhaps a user understands one style of visualisation better than another, or perhaps one type of visualisation form makes it easier to perform a particular task, whereas another form makes it easier to perform a different task. For these purposes, there are many visualisation tools and programming libraries that help users create multiple view visualisations. However, it is not easy for a new developer to know how to lay out and position the views in their systems, how many views they should use, what is the best visualisation type for each view, or what design attributes work best. Therefore, developers and learners should have guidelines and frameworks to assist them in making the right design decisions.

The long-term vision of this research is to develop theories for data visualisation, and develop specific guidelines on best practices for multiple view systems; this will assist researchers and developers in understanding and developing multiple view systems. To achieve these goals, guidelines need to be based on current practice. Our methodology is to perform an in-depth quantitative investigation to understand best practices for what researchers currently do with multiple view systems. From this investigation guidelines can be developed. Furthermore, such guidelines could be programmed into a grammar, and into a multiple view design tool, which would help developers create multiple view visualisations.

Therefore, this work focuses on investigating multiple view systems, in order to develop a set of guidelines and a system to help new developers to create multiple view visualisations and make the correct design decisions. For this purpose, we designed a visualisation tool based on a quantitative evaluation of multiple view systems.

This grammar-based tool allows learners to create, control and save multiple view visualisations in a simple way by using a multiple view grammar.

The research focuses on eight research questions that are used to structure this thesis, ranging from counting views, quantifying views, developing a multiple view grammar, and creating a multiple view tool: (1) What are the strategies for selecting which multiple view images to evaluate? (2) What are the strategies for defining and determining a “view” in multiple view visualisation? (3) What are the strategies for coding multiple view topologies and visualisation types? (4) How many views should developers use in multiple view systems? (5) What layout arrangements are popular in multiple view systems? (6) What visualisation types are used in each view and what types of visualisation come together? (7) What salient guidelines can be learnt from the analysis, to assist users in developing multiple view visualisations? (8) What is a multiple view grammar and how is it used to create a multiple view layout?

By tackling and answering these research questions, the thesis makes six novel research contributions. First, it introduces a strategy for selecting which images of multiple view visualisations to evaluate (Chapter Three). Second, this research creates a strategy to help researchers ascertain “what constitutes a view” in a multiple view visualisation (Chapter Three). Third, through statistical analyses of multiple view visualisations, this research produces results of a comprehensive quantitative analysis of multiple view visualisations, which can help researchers to conduct further investigation on multiple view systems (Chapter Four). Fourth, from this analysis, the thesis develops a set of guidelines to help novices in the data visualisation field, as well as developers, to create robust multiple view visualisations (Chapter Four). Fifth, this research introduces a new grammar to create multiple view layouts by using the concept of cutting a view vertically or horizontally to create two views (Chapter Five). Sixth, this work develops the LayMV tool to create, control and save multiple view visualisation, based on the analysis of the multiple view systems and the set of guidelines for creating multiple view systems. The LayMV tool uses the multiple view grammar to create and manage multiple view visualisations (Chapter Six).

In conclusion, this dissertation provides learners and practitioners with an in-depth analysis of the multiple view field, which can help them create multiple view

visualisations and carry out further investigations on multiple view systems. In addition, the LayMV tool and the multiple view grammar can help users to create, control, save and reload multiple view systems.

Contents


1	Introduction	1
1.1	Introduction	1
1.2	Motivation	3
1.3	Vision and aims	3
1.4	Scope of research	5
1.5	Research questions and objectives	5
1.6	Research methodology and the structure of the thesis	8
1.7	Contributions	10
2	Related work	13
2.1	Introduction	13
2.1.1	Scope and methodology of the related work	14
2.2	History, background and concepts	16
2.2.1	What is a “view” (so that we can have multiple views)?	20
2.2.2	How frequently are <i>multiple</i> and <i>view</i> used?	21
2.2.3	What parts of speech are used for <i>multiple</i> and <i>view</i> ?	22
2.2.4	What are the meanings of a <i>view</i> ?	25
2.2.5	What words have people used instead of “view”?	26
2.2.6	Creation of views	28
2.3	Design concepts and general principles of multiple views	30
2.4	Multiple view tools and their use	33
2.5	Theories and design guidelines for multiple view visualisations	37
2.6	Summary	39
3	Data gathering and quantification preparation	40
3.1	Introduction to data gathering, quantification and preparation	40
3.2	Image selection and storing data	43
3.3	Developing general guidelines for view identification	49
3.4	Coding the layout arrangements	54
3.5	Coding the visualisation types	62
3.6	Discussion	64
3.7	Summary	66
4	Quantification (data collection), analysis, and design guidelines for multiple view systems	67

4.1	Introduction	68
4.2	The tabletop strategy to quantify multiple view layouts	71
4.3	The Quantification and the analysis of views number in multiple view layouts	73
4.4	The Quantification and the analysis of the symmetrical multiple view layouts	75
4.5	The quantification and the analysis of the layouts arrangements	77
4.6	Understanding tasks and domain	80
4.7	Quantification and analysis of visualisation types	88
4.8	Quantification and analysis of collocational pairs of the visualisation types	95
4.9	Developing of design guidelines for multiple view visualisations . . .	97
4.10	Discussion	100
4.11	Limitations	101
4.12	Summary	103
5	Design and development of a grammar for the MV Layouts tool	105
5.1	Introduction to design of the grammar	105
5.2	Developing the rules for the MVG grammar	108
5.3	Design of the MVG grammar	111
5.4	MVG grammar for multiple view layouts	113
5.5	Examples for the MVG grammar	115
5.6	Discussion	118
5.7	Summary	120
6	Design and implementation of the MV layouts tool	122
6.1	Introduction	122
6.2	Design of the LayMV tool	125
6.3	Implementation of the LayMV tool	130
6.4	Results	138
6.5	Discussion	143
6.6	Summary	145
7	Case studies and discussion	147
7.1	Introduction	147
7.2	Preparation process and scenario to create multiple view visualisation	149
7.3	First case study	150
7.4	Second case study	153
7.5	Third case study	161
7.6	Discussion	167
7.7	Summary	168
8	Discussion and conclusions	170
8.1	Discussion	170

8.2	Consideration of the work	173
8.3	Reflections on the research questions	175
8.4	Limitations and future work	177
8.5	Conclusion	179

List of Figures

1.1	(Research Procedure) (1) In (§3) Preparation , we explain how we extracted 491 images from IEEE VIS 2012-2018 Conference publications, and how we defined a view in multiple view visualisations. (2) In (§3) Coding the multiple view visualisations , we illustrate how we coded the images by their topology (making sketches of the layout, totalling 22 sheets of paper); discussing cases to confirm their layouts, we cut the 22 sheets of sketches into individual tiles, and organised them on a tabletop, to analyse and tally the quantities. Also, in this section we explain how we coded the visualisation types in order to calculate their quantity. (3) In (§4) Quantification, analysis and discussion , we analyse and discuss the results from quantifying multiple view visualisations. Finally, (4) in (§4) Discussion, Conclusions and Guidelines , we discuss and summarise the overall results, and we recommend a set of design guidelines to help beginners in the data visualisation field to create robust multiple view visualisations.	9
2.1	The methodology of the related work chapter based on exploring the topics related to the multiple view visualisation area.	14
2.2	Picture from Roberts’ paper “On encouraging multiple views for visualisation”, showing how multiple views can be formed by changing parameters in the filtering of the data, or the mapping of the data, or the presentation and display of the information. Image ©IEEE [105] presented in International Conference on Information Visualisation (IV), 1998.	18
2.3	Wordcloud of our thesaurus of collocated words that are found in similar contexts to the given lemma <i>view</i> , in the v6Y corpus.	26
2.4	Multiple views can be created and displayed alongside each other in different ways: from different projections (a), changing the visual form (b), or by adapting the data (c). Views can also be created to be superpositioned (d) or merged through an explicit encoding (e).	28
3.1	Methodology of data gathering and quantification preparation chapter.	42

3.2	Four visualisation tools (T1-4). Visualisations T1 and T2 clearly show three views, while T3 has two windows and a menu that is ignored (three explanatory visualisation tools created by our students [107]. T4 shows the Vinca estuary visualisation tool [47] demonstrating five views. . . .	49
3.3	We sketched each topology in a small picture; we sketched all possible alternatives, which were later discussed.	56
3.4	Nomenclature and icons for the most frequent layouts.	57
3.5	Image (from Brown et al. [22]), an example of coding 4-views layout. . .	58
3.6	Image (from Chu et al. [34]), an example of coding 4-views layout. . .	58
3.7	Image (from Barba et al. [10]), an example of coding 6-views layout. . .	59
3.8	Image (from Crnovrsanin et al. [37]), an example of coding 3-views layout.	59
3.9	Image (from Hall et al. [53]), an example of coding 2-views layout. . . .	60
3.10	Figure showing 21 image thumbnails and their respective layout classification. Sample images from our database of 491 images.	61
4.1	The methodology of the quantification and analysis process chapter. . .	69
4.2	Using a tabletop strategy, I cut these sheets into individual tiles such that I could discuss them and move them around on a tabletop. I layout each of the multiple-view topologies as separate sketched tiles. The simple abstract drawings gave us a easy way to compare the structures without being distracted by the actual view design or content. This provides a visual summary of the range of layouts.	72
4.3	Histogram showing the frequency distribution of the views. From 491 multiple view systems, in our study we find that a 3-view system is most frequent.	73
4.4	Histogram of symmetrical versus non-symmetrical views.	76
4.5	Correlations matrix of visualisation types, highlighted the collocational and non-collocational pairs for visualisation types in layouts with six views or less.	96
5.1	The picture demonstrates the Multiple View Grammar. It presents a state-diagram to explain the cut algorithm. The user can cut the main view horizontally (so it becomes two views) or vertically to create two-view vertical layout.	107
5.2	The MVG grammar expression “ V(50H(50,50),50) ” describes the layout  in a hierarchy structure.	113
5.3	The shortcut MVG grammar expression “ grid 3x3 ” created a grid layout with nine views.	114

5.4	The shortcut MVG grammar expression “GoldenRatioV6Q4S70%” created a golden ratio layout, where “V6” is the number of views in the layout (in this example, we have 6 views), “Q4” indicates the position of the golden ratio in the layout (in this example, the golden ratio is placed in the fourth quarter of the layout), and “S70” represents the size of the layout which is 70 percent of the multiple view tool panel, and we will give more detail about it in the next chapter.	115
5.5	On the left, we show the grammar code v(50h(50,50),50) . Where, the rendering of the code is shown on the right.	115
5.6	On the left we show the grammar code v(75h(25,75v(50,50)),25) . Where, the rendering of the code is shown on the right. And, as shown in the picture, we can define the size of the view by changing the view’s ratio in the grammar expression	116
5.7	This example shows how the user of the MVG grammar can define variables to describe more complex layout.	116
5.8	The grammar code, on the left, shows how variables can be used. Where, the rendering of the code is shown on the right.	117
5.9	This example show how to use a complex grammar expression to describe a complex multiple view layout.	117
5.10	On the left we show the grammar code. The rendering of the code is shown on the right. Or, we can use the shortcut grammar “grid 4x4” to create the same layout.	118
6.1	This diagram illustrates the stages to build the LayMV tool, started with the design of the tool, the implementation, Describe the LayMV Tool, and finally a discussion about the LayMV tool.	124
6.2	The wireframe layout of LayMV tool, showing four main views: the grammar editor, visual panel to control the layout visually, and tree-navigator (to select different parts of the layout code) and the visualisation editor (to allow users to add in their own visualisation code).	126
6.3	This diagram describes the LayMV tool and illustrates the main components and functions, these are the main components of the system, where “A” represented the visual components and “B” underneath the line represents the hidden (algorithms) components.	130
6.4	This diagram describes the implementation of LayMV tool, we asked experts in multiple view tools to use the LayMV prototypes and give feedback for each version of the tool.	131

6.5	A snapshot for an early version of the LayMV tool, the “Add View” button adds a view to the first left view of the LayMV tool each time we clicked on it. In addition, we can use the mouse cursor to drag and drop the views to move them around and build a multiple view layout; Then, by clicking on the “Render Layout” button, we can render the same layout in the third view of the tool.	133
6.6	A snapshot for an updated version of the LayMV tool, the “Create layout by MVG Grammar” button created the multiple view layout by using the MVG grammar “a:50; b:(a,a); Expr:v(40h(25,75vb),20hb,20hb,20hb);” .134	134
6.7	A snapshot for the LayMV tool after we added the tree navigator view and the shortcut grammar (Layout technique editor), the expression “goldenratio6q4s100%” in the Layout technique editor created the above golden ratio layout.	135
6.8	Template viewer. Users can choose a starting layout, search for a specific view quantity, various multiple view Layouts can be created with different views number, and the user can later edit the template.	139
6.9	LayMV tool, with three linked views. (A) grammar panel to edit the grammar (either shorthand e.g., v(50,50) or full JSON, shown), (B) Visualisation panel of either the wireframe editor or layout editor (shown) and (C) property panel.	140
6.10	Using MVG grammar to create multiple view Layouts.	142
6.11	Using LayMV Layout Technique to create multiple view Layouts.	143
7.1	This diagram describes the steps that the user of the LayMV tool should follow to create a multiple view visualisation.	148
7.2	Template viewer. Users can choose a starting layout, search for a specific view quantity, various multiple view layouts can be created with different views number, and it can edit later in the second screen. Label “A” shows the multiple view layout that was chosen through a left mouse click , then we clicked on the “Create New Layout” button as shown by label “B” . . .	151
7.3	Main view. After choosing the template it is shown in the main view. There are three vertical panels. The left most panel controls the grammar, and has three tabs. The middle part is the visual editor, allowing users to ‘draw’ the views. The right most panel shows two views, which can control the appearance and allow users to add in data and visualisation code. Along the top, users can choose to save the project and load it. . .	152
7.4	Main screen showing two bar chart visualisations. Users can choose a starting layout, search for a specific view quantity, various multiple view Layouts can be created with different views number, and the user can later edit the template.	152

7.5	Final visualisation. This screenshot demonstrates the final visualisation for the first case study. It shows a simple D3.js side-by-side view of two bar charts.	153
7.6	Template viewer. We start again, for case study 2, on the template screen. In this case the user selects the 2 by one view layout.	154
7.7	Template viewer. Users can choose a starting layout, search for a specific view quantity, various multiple view Layouts can be created with different views number, and the user can later edit the template.	155
7.8	Shorthand grammar viewer. Users can write the shorthand grammar in this window, which will display (when updated) in other views. Using the shorthand users can create complex views, using some simple commands.	155
7.9	Layuot Appearance views. The user can adapt different aspects of the appearance. They are able to choose a specific Component of the layout. The tree is automatically updated from the information in the MVG grammar. Subsequently, users can then select the Viz, add the Data and the Visualisation code. Here the Layout appearance is shown in the lower view. Users can change the parameters to adapt how the views appear. .	156
7.10	This screenshot for the LayMV tool shows the result of the creation process of the multiple view visualisation. Where label “A” points to the layout editor (Figure 7.11 gives a clear picture with more details about this part of the LayMV tool). Moreover, label “B” points to the properties window of the visualisation technique that located in the third view at the bottom of the multiple view visualisation (Figure 7.12 gives a clear picture with more details about this part of the LayMV tool). Finally, label “C” points to the Data section in the tree navigator that belong to the third view (Figure 7.13 shows a screenshot for the Date window that should appear instead of the properties window of the visualisation technique in the user click on the Data part that belongs to the third view).	157
7.11	This figure shows a clear picture for the layout editor window that was labelled in Figure 7.10 as “A”.	158
7.12	This figure shows a clear picture for the properties window of the visualisation technique that was labelled in Figure 7.10 as “B”.	158
7.13	This figure shows a clear picture for the Data window that was labelled in Figure 7.10 as “C”.	159
7.14	Screenshot for the multiple view visualisation that created by the LayMV tool.	160
7.15	This figure shows how the user can change the layout of the multiple view visualisation, that shown in Figure 7.14, using the MVG grammar . . .	161
7.16	The MVG grammar to create a layout with eight views.	162
7.17	Template viewer. Starting again from scratch for case study 2. First use the template screen. In for this case study, the user selects the 2 by one view layout.	162

7.18	Template viewer. Users can choose a starting layout, search for a specific view quantity, various multiple view Layouts can be created with different views number, and the user can later edit the template.	163
7.19	We used the grammar viewer users can create the multiple view layout in this window.	163
7.20	From the layout editor we can control layout's dimensions.	163
7.21	This screenshot for the LayMV tool shows the result of the layout creation process of the multiple view visualisation. Label "A" points to the layout editor, where (Figure 7.22 gives a clear picture for this part of the LayMV tool). Moreover, label "B" points to the visualisation properties window that located at the bottom corner of the LayMV tool, where (Figure 7.23 gives more details about this part of the LayMV tool). Finally, label "C" points to the "Data" node in the tree navigator that belong to the third view of the multiple view layout, where (Figure 7.24 shows a screenshot for the Date window that should appear instead of the visualisation properties window when the user click on the "Data" node).	164
7.22	This figure shows a clear picture for the layout editor window that was labelled in Figure 7.21 as "A".	165
7.23	This figure shows a clear picture for the properties window of the visualisation technique that was labelled in Figure 7.21 as "B".	166
7.24	This figure shows a clear picture for the Data window that was labelled in Figure 7.21 as "C".	166
7.25	Screenshot for the multiple view visualisation that created by the LayMV tool.	167

List of Tables

2.1	Multiple views have been applied to many domains and application areas. This table presents a summary of some of these areas.	20
2.2	Part of speech (PoS) for <i>multiple</i> and <i>view</i> , highlighting they are more frequent in the visualisation corpus (v6Y) in comparison to the Open Access Journal corpus (oAJ). Raw frequencies, and per-million normalised count $_{pm}$ are shown. Non-visualisation words (typically found in any academic texts) show similar proportions across corpora.	22
2.3	Part Of Speech examples (PoS) for <i>multiple</i> and <i>view</i> published in IEEE VIS.	23
2.4	Frequency of occurrence, showing raw and normalised per million values for the first twenty examples of nouns and verbs modified by <i>multiple</i> and modifiers of <i>view</i> . In the v6Y corpus (IEEE VIS TVCG journals 2012–2017).	24
2.5	Words that are prefixed with <i>multi-</i> in the v6Y corpus (frequency shown <i>per million</i>). In these results n-tuples are not included, but are shown in Figure 2.4, e.g., <i>multiple views</i> (without a hyphen) has a frequency of $(268_{raw}, 42.51_{pm})$. To understand scale, v6Y: 100_{raw} is 12.16_{pm}	25
3.1	Strategy to select which multiple view images to evaluate.	47
3.2	Strategy to select which multiple view images to evaluate.	48
3.3	Strategy to help ascertain ‘what is a view’ in a multiple view visualisation.	53
3.4	Strategy to code layout topologies.	55
4.1	Results of tallying the views. Views were also tallied per years, with the frequency and the percentage frequency. Applications with 20+ views are aggregated together (and treat them with a system of 20 views, for calculations).	75
4.2	Results of tallying the specific layouts, per years. Where f is frequency, $\%Vf$ is percentage frequency of that View type, and $\%f$ is percentage overall. The complete dataset is shown in this table. It demonstrates a long tail of many cases with few instances, in fact there are 81 layouts with $f < 1$). The rows are ordered by their overall rank.	79

4.3	Overall data visualization domains in all layouts, the top-ranked data visualization domains are mostly found in the top-ranked layouts (as explained in Section 4.3 and shown in Table 4.1 and the histogram Figure 4.3).	85
4.4	The quantification of data visualization domains in the top-ranked layout arrangements (the order of layout arrangements is taken from the top layout arrangements that listed in Table 4.2 at Section 4.5). The top six rows are highlighted in green because there are more data domains within these layouts.	87
4.5	visualisation view types, calculated by year (2012 to 2018).	91
4.6	Overall visualisation types in all layouts, the top-ranked visualisation types are mostly found in the top-ranked layouts.	92
4.7	The quantification of the visualisation types in the top-ranked layout arrangements, as both were ranked based on their respective overall rankings.	94

Chapter 1

Introduction

1.1 Introduction

In today's digital universe, data are expanding rapidly, and developers and researchers are creating huge databases. In addition, users employ numerous multiple view visualisation systems to explore, explain, and discover information from databases. For example, they use side-by-side views in competitive systems to compare two databases, or look at a single database from different perspectives. And while multiple views have been used in visualisation for more than thirty years, people often do not know how many views they should use. Besides that, they may not know how to lay out their views or adopt best practices for positioning these views on a screen.

There are many reasons why understanding the quantity of views, the quantity of visualisation, and the layout configurations is useful. One major reason is to provide results that can underpin the development of a theory of visualisation. Expert users draw upon a range of tacit knowledge that they apply in order to design effective systems, and refine their designs through many iterations. It is this tacit knowledge that is encoded in their final designs. Our analysis provides quantitative research that can underpin such theoretical models. Another reason is so that we can develop a set of design guidelines which will help not only visualisation experts but also those who are creating visualisations for the first time. The vision of this research is to develop guidelines and share best practice of different techniques for the general public. If, as a community, we want to provide guidelines for best practice, we need to understand exactly what we are doing. As visualisation tools and systems become more widely accessible, and we see a democratisation of visualisation techniques where the public are creating and using visualisations, we need to perform quantitative research that can be developed into

practical advice. Much like an architect would provide guidelines for how to best design a building, or a colour researcher would provide guidelines over the design of colours, so we, as a community, should be providing guidelines in a variety of visualisation topics.

This thesis presents a quantitative evaluation of the number of views used in multiple view systems, their layout configurations, and the types of visualisations used. In this work we concentrate on view juxtaposition, where each view sits alongside each other view, and on the topology of each design layout (e.g., a 2-view system can have one view above the other, or left/right of each other). We focus on tools that were presented in research publications at the IEEE Visualization Conference between 2012 and 2018, inclusively. This seven-year period provides a convenient and reproducible set of images of the modern visualisation tools that have been designed and presented by community experts. In particular, because these works have gone through peer review, we assume that the authors have spent much careful thought over how they present their tools; and consequently, they have been attentive to the selection of their views and the presentation of their multiple view systems. After selecting and extracting the images, we codify them and perform an in-depth analysis of these results.

This study considered many sources, including making use of a general internet search for visualisation images, video sources such as Vimeo and YouTube, and other online image repositories. We believed the inclusion of these sources might provide us with a rich data set of different images, and this is certainly a limitation of what we carried out here. However, they also bring challenges, because image searches can change over time, and results can change according to user or geographic location, which would make it more difficult for others to confirm our studies, and to add more images for future years.

By focusing on the layout of visualisations (as presented at the IEEE Visualization Conference) and by understanding what types of visualisation authors use, designers will have a whole set of guidelines enabling them to develop better visualisation tools, and researchers can create guidelines which will be useful for visualisation learners, designers and others who wish to create visualisations. This set of guidelines can then be used to help designers to create new tools or systems, and guide them to base their

layout visualisations on best practices. Because expert knowledge is currently tacitly encoded in visualisation designs, there are numerous research questions that remain unanswered in the community, in the area of multiple views. However, until now, no guidelines of this type have ever existed.

1.2 Motivation

Multiple view systems are often used by visualisation developers. But it is not easy for developers to know how to lay out and position the views in their systems, how many views they should use, what is the best visualisation type for each view, or what design attributes work best. We believe that developers and learners should have guidelines and frameworks which can help them to make good design decisions.

Consequently, the goal of our work is to help researchers to develop theories for visualisation. And, to achieve this goal, we perform in-depth research to understand the best practices for what we currently do. The thesis investigates multiple view systems and develops guidelines and a system to assist new developers in creating better multiple view visualisations.

1.3 Vision and aims

At the current time, the visualisation community is expanding. Visualisation is not only carried out by visualisation experts who go to visualisation conferences, and who may implicitly understand how to create visualisations, but it is also produced by the general public, by enthusiastic developers, and by employees of companies with typically no experience or history of developing data visualisations.

These novices are often creating visualisations without much or any knowledge of good practice that the academic community has developed over the past twenty or so years. Visualisation democratisation has occurred in part because of the rise of the availability of open-source tools and software such as D3.js, Angular.js, RStudio, the widespread access to commercial visualisation tools, such as Qlik, Tableau and Matlab, and the inclusion of visualisation functions to common tools such as Microsoft's Excel. Consequently, people around the globe are now using visualisation to help

them better understand their data. They are developing systems, using visualisations in their day-to-day lives, and so on. But how do they distinguish between what is good practice and what is not? Consequently, we need to examine current practices within the community, and share those experiences with the general public. We need to create suitable tools to help developers use and apply best practices.

The vision of this work is to develop guidelines for visualisation developers and teachers with the following rationale:

If we can understand the multiple view visualisation area better, and place some quantification analysis on multiple coordinated views, then we will be able to develop a set of guidelines which help developers understand how to create appropriate tools.

For that reason, when guidelines have been created, they can be used by the public, by students learning more about visualisation, or by academics to help them teach best practices. Furthermore, they can be incorporated into design tools, or even underpin specialist automatic design tools. Our long-term vision is to eventually develop automatic tools that incorporate these principles, which particularly utilise multiple views. Moreover, while there are already a number of academic papers that provide guidelines of multiple views, they appear to be based on subjectivity rather than quantitative data. For instance, the well-known paper by Baldonado et al. [144] provides “Guidelines for Using Multiple Views in Information Visualization”, yet it is unclear what quantitative information underpins this work. It appears that that work was based on subjective intuition. Consequently, our focus is on quantitative analysis. By simply considering Baldonado et al., we can accept or deny their guidelines. At the start of our research, however, we specifically set out to test if the rule of parsimony stands true: that users only utilise very few visualisations in a multiple view system.

The aim of this study is to develop theories for data visualisation, and, specifically, to develop guidelines on best practices for multiple view systems; this will help learners better understand and develop multiple view systems. But to achieve these goals, this investigation needs to perform in-depth quantitative evaluation research to understand best practices for what researchers currently do with multiple view systems. In addition,

this research concentrates on providing a tool to create multiple view visualisation based on a new multiple view grammar.

1.4 Scope of research

Multiple views have been used in information visualisation for many years. There are different design strategies from view juxtaposition, superposition of many views, or cleverly merging the view information, such as by overloading or nesting. In this research we focus on multiple view layout using view juxtaposition strategy, where developers display information in many side-by-side views. One of the reasons developers use such duplication is to help users understand better the information which is displayed. Additionally, the manipulation of data within these juxtaposed views is often interlinked. In fact, Coordinated Multiple View systems provide the backbone of most modern visualisation systems. However, because our primary focus is on multiple views, we do not investigate coordination in any great depth, although we acknowledge that it is an extremely important principle, and we do indeed cover some aspects of coordination in Chapter 2 (Related Work). However, coordinating views is a separate challenge, and it relies on quick associated arrays and different coding infrastructures. We also acknowledge that there is certainly much research that can be conducted in this area, but we leave it to future work, and other researchers, to investigate and review aspects of coordination in multiple view systems.

1.5 Research questions and objectives

Our approach begins by addressing several research questions, both overarching and more specific sub-questions. We expand on the holistic motivational research questions below, and include several discussions to some of the sub-questions in each chapter. For example, sub-questions include: How were images selected for the multiple view study? What is the definition of view? How do we count views and organise layouts to quantify them? Does symmetry play a role? How should design attributes be considered? Does the type of (or form of) visualisation used make a difference to the view counts and layout? Below, we summarise our primary research questions (RQ) and outline our methods to investigate them.

RQ1 What is the strategy to code layout topologies and visualisation types? To answer to this question, we sketch all layouts by using the figure-ground method. First, we remove all the content from the views. Then, we code the layout topologies after we determine the views for each layout. Subsequently, we code the visualisation types for each view. This question is addressed in Chapter 3.

RQ2 How many views are used in multiple view systems? Our solution is to quantitatively analyse the number of views that developers use, presented in the visualisation literature, by extracting images from visualisation papers presented in the IEEE visualisation conference series, over a seven-year period, coding the images, and calculating the quantity of each configuration. These results can be used by developers to guide them in decisions over view quantity and design. This question is addressed in Chapter 4.

RQ3 Do developers prefer symmetrical or non-symmetrical layouts for multiple view systems? Our solution is to quantify and analyse the topologies of multiple view layouts. This question is addressed in Chapter 4.

RQ4 What layout arrangements are popular in multiple view systems? To answer this question, we itemise and classify views by their basic topological layouts, and discuss the results. Our resulting taxonomy (and popularity of each configuration) can be used by those who wish to configure multiple view systems, and by developers to create suitable systems. This question is addressed in Chapter 4.

RQ5 What visualisation types are used in multiple view systems? What visualisation types are the most frequently used? Are some layout arrangements more likely to hold certain types of visualisations? Our solution is to code and itemise the types of visualisations (bar chart, line graph, scatter plot etc.) and their use in different layouts. This information can be used to develop and summarise design guidelines and identify best practices in multiple view visualisation. This question is addressed in Chapter 4.

RQ6 What types of visualisations come together in multiple view systems? To answer this question, we calculate the visualisations that occur together by using the basket analysis method. This question is addressed in Chapter 4.

RQ7 What salient guidelines can be learnt from the analysis, to help users to design and develop robust multiple view visualisations? The analysis of the answers to the above questions will assist us in providing a set of guidelines to help learners in the visualisation field and help new developers to build multiple view visualisations. This question is addressed in Chapter 4.

RQ8 What is a multiple view grammar and how can it be used in multiple view tools to create multiple view layouts? This question is addressed in Chapters 5, 6 and 7.

These aims will be achieved through the following measurable objectives:

Obj 1 Preparing and coding multiple view visualisations. In Chapter 3, this study will illustrate the preparation and coding processes, which will help answer research question RQ1.

Obj 2 Quantifying multiple view visualisations. In Chapter 4, this study will explain how we quantify multiple view visualisations, which will enable us to answer research questions RQ2, RQ3, RQ4, RQ5 and RQ6.

Obj 3 Analysing the quantification and introducing a set of guidelines to help developers to create multiple view visualisations. In Chapter 4, this study will show how we analyse the quantification of multiple view visualisations, which will enable us to answer research questions RQ2, RQ3, RQ4, RQ5, RQ6 and RQ7.

Obj 4 Introducing a tool to create and control multiple view layouts based on a multiple view grammar. In Chapters 5 and 6, this study will demonstrate how we formulate a grammar which will help us to create multiple view layouts. In addition, we describe the stages of building a multiple view tool that uses a multiple view grammar, which will enable us to answer research question RQ8.

1.6 Research methodology and the structure of the thesis

This doctoral study went through three stages in order to perform the quantitative study, as shown in Figure 1.1 below: (1) **Preparation**, where we selected which images would be used, and extracted them from the papers into a separate database. This study analyses the layout strategies by creating a database of images from the IEEE Visualization Conference (IEEE VIS). Every year the IEEE (Institute of Electrical and Electronics Engineers) organises an annual Visualization Conference, which includes journal papers published in the IEEE Transactions on Visualization and Computer Graphics, associated conferences and poster publications. We created a database of images from the IEEE VIS Conference between 2012 and 2018, inclusive. We selected 491 images, made judgements on the 491 views, made abstract sketches (totalling 17 sheets of paper), and discussed cases in order to confirm their layouts. (2) **Coding**, where we considered each visualisation in turn, judging the topological makeup of each visualisation, coded them in such a way that we could classify them, and recorded a sketch of their topology; we also coded and recorded the types of visualisations in each layout. (3) **Analysis**, where we organised and classified the views according to their layout, using a method of physically organising the pieces of paper on a table; we then counted the layouts, and performed quantitative analysis.

(1) Preparation: (i) Select files. (ii) Extract images. (iii) Name images. (iv) View identification.

(2) Coding: (i) RQ1, coding the layout arrangements. (ii) RQ1, coding the visualisation types.

(3) Analysis: (i) Results RQ2, how many views? (ii) Results RQ4, what layout arrangements? (iii) Results RQ5, what visualisation types?

We structure this thesis similarly to our study methodology, as shown in Figure 1.1 below. First, we describe related work in Chapter 2. Second, in Chapter 3, we describe how we collected the images for our analysis and how we identified a view in multiple view visualisation, and we present how we have codified the layout arrangements and

visualisation types. Third, in Chapter 4, we present the results for the quantification of multiple view visualisations. We then discuss the overall results and present the design guidelines for how to create robust multiple view visualisations. Fourth, in Chapter 5, this thesis develops and explains a new multiple view grammar that allows users to create and control multiple view layouts. Fifth, in Chapter 6, this work builds a tool that gives users the ability to create, control, save and reload the multiple view layouts. Sixth, in Chapter 7, we present two case studies to test and discuss the ability of the multiple view tool to create multiple view visualisations. Finally, in Chapter 8, we describe our conclusions and make suggestions for future works.

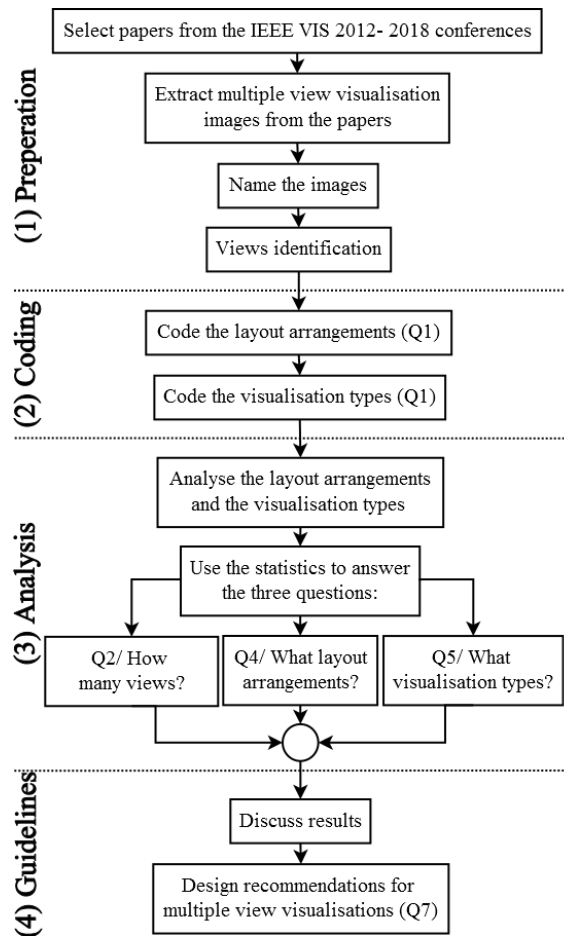


Figure 1.1: (Research Procedure) **(1)** In (§3) **Preparation**, we explain how we extracted 491 images from IEEE VIS 2012-2018 Conference publications, and how we defined a view in multiple view visualisations. **(2)** In (§3) **Coding the multiple view visualisations**, we illustrate how we coded the images by their topology (making sketches of the layout, totalling 22 sheets of paper); discussing cases to confirm their layouts, we cut the 22 sheets of sketches into individual tiles, and organised them on a tabletop, to analyse and tally the quantities. Also, in this section we explain how we coded the visualisation types in order to calculate their quantity. **(3)** In (§4) **Quantification, analysis and discussion**, we analyse and discuss the results from quantifying multiple view visualisations. Finally, **(4)** in (§4) **Discussion, Conclusions and Guidelines**, we discuss and summarise the overall results, and we recommend a set of design guidelines to help beginners in the data visualisation field to create robust multiple view visualisations.

1.7 Contributions

The thesis makes six novel research contributions in the visualisation field, as follows:

- (1) A strategy to select which images of multiple view visualisations to evaluate.
- (2) A new strategy to help ascertain ‘what is a view’ in a multiple view visualisation. Other researchers can apply this strategy in order to quantify how many views are shown in a figure or screenshot.
- (3) Results of the comprehensive quantitative analysis of the layout and the visualisation techniques of multiple view visualisations. These results can be used by other researchers to help them understand the current use of multiple views in the visualisation literature. They can be used to help them advise others and create guidelines for multiple view systems. They can be used to help create automatic-layout systems, or recommender systems. The results also provide a snapshot in time, and in the future a similar study could be performed and compared to our results. Researchers could then observe how the community has changed.
- (4) A set of guidelines to help the community understand, and developers to create, robust multiple view visualisations.
- (5) A new grammar which can be used to create a tool for multiple view layouts and multiple view visualisations tool. The thesis presents a demonstration system that integrates the grammar, and demonstrates how it is possible to use it; other researchers can then extend the grammar and/or utilise it in their own tools. It can be used by other researchers to help develop appropriate visualisation tools, and save multiple view layouts.
- (6) A new demonstrator tool, LayMV, which creates, controls, saves and reloads multiple view visualisations. The LayMV tool uses the multiple view grammar to create and manage multiple view visualisations.

Furthermore, a total of six papers have been published during the time of the research for this thesis (P1-P6). Each of the publications has focused on different aspects of the thesis; these include the strategies to quantify multiple view visualisations, in addition to the guidelines and the tool to create multiple view visualisations. For each, we include below a short description of the content and how it contributes to the thesis.

(P1) H. M. Al-maneeea and J. C. Roberts, ‘Study of Multiple View Layout Strategies in Visualisation’, in Posters presented at the IEEE Conference on Visualization (IEEE VIS 2018), Berlin, Germany, Oct. 2018.

Included in Chapters 3 and 4, this 2-page poster paper contributes the initial results for the quantification, includes some early values, and presents the process. From the poster presentation, and discussion of this work with Professor John Stasko, we changed the quantification process to also include visualisation systems with one view.

(P2) J. C. Roberts, H. M. A. Al-Maneeea, P. W. S. Butcher, R. Lew, G. Rees, N. Sharma and A. Frankenberg-Garcia, ‘Multiple views: Different meanings and collocated words’, English, Computer Graphics Forum, Mar. 2019, issn: 1467-8659. doi: <https://doi.org/10.1111/cgf.13673>.

The second author contributed to knowledge of multiple-view systems and different words for multiple view. While not the main focus of this thesis, ideas from this paper are contained within the thesis in Chapter 2 on Related Work.

(P3) H. M. Al-maneeea and J. C. Roberts, ‘Towards quantifying multiple view layouts in visualisation as seen from research publications’, in 2019 IEEE Visualization Conference (VIS), Oct. 2019, pp. 121–121. doi: 10.1109/VISUAL.2019.8933655.

Included in Chapters 3 and 4, this short paper presents the methodology, discussion of ‘what is a view’, and the main quantitative results.

(P4) H. M. Al-maneeea and J. C. Roberts, ‘A tool to help lay out Multiple View Visualisations guided by view analysis’, in Poster session presented at Eurovis 2020, Norrköping, Sweden, May. 2020.

Included in Chapters 5 and 6, this poster paper introduces the LayMV tool to create multiple-view designs.

- (P5) X. Chen, W. Zeng, Y. Lin, **H. M. Al-Maneea**, J. Roberts and R. Chang, ‘**Composition and configuration patterns in multiple-view visualisations**’, Transactions on Visualisation and Computer Graphics, (pp. 5, 14, 39), doi: 10.1109/TVCG.2020.3030338

Included in Chapter 4, collaborating with Prof Remco Chang, X.Chen, W.Zeng and Y.Lin we shared results of our quantitative analysis process and worked together on a paper focusing on their tool. The writer of this thesis contributed with the quantification results of multiple view visualisations and the ideas and the general background of multiple views.

- (P6) J. C. Roberts, J. W. Mearman, P. W. S. Butcher, **H. M. Al-Maneea** and P. D. Ritsos. “3D visualisations should not be displayed alone – encouraging a need for multivocality in visualisation”. Eurographics UK-Chapter conference. EG UK Computer Graphics & Visual Computing (2021).

This represents a collaboration between many authors. Results from the quantitative analysis (Chapter 4), especially those focusing on 3D visualisations, fed into this paper.

Chapter 2

Related work

This chapter aims to provide a holistic understanding of the topics related to multiple view visualisations. The goals are (a) to understand the development of multiple views; and (b) to understand the design principles of multiple views and different multiple view tools. Therefore, this related work provides some underpinning information that leads towards the goal of developing guidelines related to multiple view visualisations. Specifically, this chapter focuses on the following research questions and is structured in the order of the research questions.

Q1/ What **terms** are used in multiple views, and what do they mean?

Q2/ What are the key moments of **history** in prior research of multiple views?

Q3/ What are the general **design** principles?

Q4/ What **tools** have been designed to create multiple view visualisations?

Q5/ What **theories** and guidelines (such as grammars) have researchers invented, that are to do with multiple view visualisations?

2.1 Introduction

This chapter explores and investigates the multiple view visualisations area, and reviews the breadth of ideas used by developers of multiple view visualisations for deep understanding. In particular, to study multiple views in depth we need to look at multiple views from different perspectives, from understanding what the terms mean, to how they are designed, the tools that are used to create them, and guidelines that researchers have created for them. This chapter therefore takes a broad approach to the topic of multiple views. Figure 2.1 shows how these ideas map together.

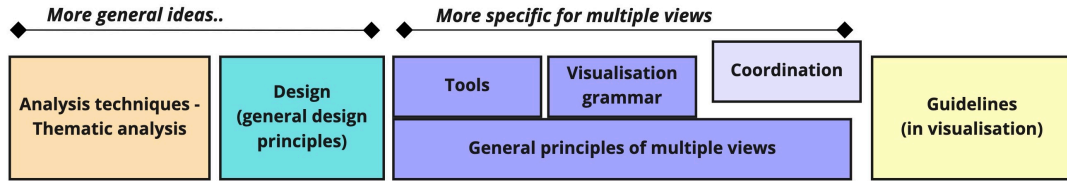


Figure 2.1: The methodology of the related work chapter based on exploring the topics related to the multiple view visualisation area.

- **Background, concepts, history and themes,** Section 2.2: The research investigates multiple views, analysis of terms used around the subject of multiple views, including the meaning of the terms used in multiple view visualisations, and the key moments of history in prior research of multiple views. The work focuses on research questions **Q1/** and **Q2/**, the terminology of multiple views, and their history.
- **Design (general design principles),** Section 2.3: This section focuses on design concepts in general, design principles, and rules about design that would be relevant to multiple views. The chapter focuses on research question **Q3/**.
- **Tools,** Section 2.4: This section focuses on tools and systems related to multiple views. Because coordinated multiple views are used with tools, we will include some information on multiple views. This focuses on research question **Q4/**.
- **Visualisation grammars, theories and guidelines,** Section 2.5: This section investigates the theories and design guidelines for multiple view visualisations. The section focuses on research question **Q5/**.
- **Summary** (Section 2.6): This section summarises the whole chapter.

2.1.1 Scope and methodology of the related work

The **scope** of the research is on view layout. The entire thesis revolves around understanding how people lay out views, and how these views are positioned side-by-side or adjacent to each other. We are also concerned with the types of visualisations

that researchers place within these views. Indeed, there are many ideas associated with multiple views that we could cover in depth, but which are beyond the scope of this chapter.

One of these important areas, which we do not focus extensively on, is the concept of *coordination*. Coordination is certainly an important area of research in visualisation. The technique of “coordinated multiple views” underpins much of interactive and exploratory visualisation [116]. When researchers mention multiple views, they often do so with the phrase ‘multiple coordinated views’ or ‘coordinated multiple views’. However, because this thesis is more concerned with visual layout of multiple views, we consider ‘coordination’ to be beyond the scope of an in-depth review for this thesis. We discussed and considered whether we should add coordination into the quantitative analysis (Chapter 4), but after this discussion we decided to restrict the focus of the analysis to aspects of layout and structure. The area of coordination, along with an in-depth quantitative analysis of how researchers use it, would definitely be interesting, but we leave this analysis for future work.

For reference, there are many papers that focus on aspects of coordination in visualisation; these include: Roberts [116], Boukhelifa et al.’s coordination model [17], North and Shneiderman’s Snap-Together [94], Weaver’s cross-filtered views [148, 149, 150], Andrienko’s coordinated views for geo-spatial information [6], Craig, Kennedy and Gunning’s work on coordinated views in networks [36], and switching and context with multiple views [35]. More recently, researchers have looked into developing multiple view interactions and coordinated views across devices, beyond the desktop and in immersive displays (such as head mounted displays) [119]. Example of this research are Chowdhury et al.’s work on multimodal interactions [32], Tahir’s work on immersive analytics [80], and Gaëlle et al. ’s work on distortion and coordination [103].

The chapter follows a methodology whereby papers are located and read using digital libraries. Because this thesis studies computing aspects of generating multiple views, the focus has been on computing-based digital libraries – notably, IEEE and ACM, along with SpringerLink, Elsevier and other publishers. Google Scholar was used to search through different publications and explore related publications. The research started with outline keywords of multiple views, side-by-side views, view layout, juxtaposition

and so on, and used the papers returned to locate other relevant publications. We followed citation links in papers to discover additional papers. The papers were also stored in Mendeley to help record the academic body of work. For the design section (Section 2.3) we note that the design literature is vast, and there are many design ideas that could be included. Consequently, only a small subset of important works is included, especially those that have been cited by researchers who publish in the visualisation domain. Finally, it is important to notice that the area of generating guidelines and investigating the theory of visualisation is a new topic; consequently, there are fewer references in this area.

2.2 History, background and concepts

Many of the early multiple view systems came from either the statistics community, or the programming community. In the late 1970s and early 1980s many researchers were investigating how object-oriented programming could be used to create different objects, perhaps with a variety of different visual displays, and coordinate them together. Developers wished to develop systems that link information between different displays. For example, Tukey and colleagues in the mid-1970s created the Prim-9 interactive tool [45] to visualise multidimensional data across several views. Statistical information in one view can be selected and displayed in a highlight colour, and can also be shown in another view. In this way, a user can explore the data, perhaps brush to select a group of items that appear close together in one dimensional view, and see how they are spread in other linked views. Apart from Tukey, many other researchers were influential in their idea and tool development. For instance, Becker and Cleveland's early work, that was published in 1987, on brushing scatterplots [12] was influential not only to the statistics community and to help revolutionise interactive statistical investigation, but also to encourage researchers to investigate and develop interactive visualisation tools. Consequently, many of the early researchers were especially motivated by being able to brush and link data between views. Researchers created a broad set of interactive brushing tools, as explained in the reviews by Roberts [116] and Ward [145].

However, in the 1980s and 1990s, few researchers were talking about 'multiple coordinated views'. Most researchers stated that they were performing 'interactive statistical analysis' or 'multidimensional brushing'. For instance, in 1982 John

McDonald was working on the Orion project [86], which he explained as “interactive graphics for data analysis”. Tufte, Becker, Cleveland and McDonald’s work, as well as other statistical researchers of the 70s and 80s, helped to establish Exploratory Data Analysis (EDA) as a branch of statistical analysis.

One of the earliest known occurrences of the phrase ‘multiple views’ or ‘multiple coordinated views’ was probably its use in 1991 by Buja [87, 23], when he illustrated an approach called *painting* multiple views to build a visualisation for complex objects. At the start of the 1990s new conferences were started. The IEEE Visualization Conference started and the UK Information Visualisation Conference was also initiated. Many researchers were developing interactive tools, particularly focused on interactive visualisation and statistical analysis.

We may consider the 1990s and early 2000s, then, to be a second wave of developments in the multiple view story, where many researchers developed interactive multiple coordinated view systems. For instance, Berkin and Jacobson in 1994 [13] published their Linked Windows Interactive Data System (Linkwinds). Other researchers then developed a variety of interactive tools. For instance, Roberts developed his Waltz visualisation tool [111], and Chris North, working with Ben Shneiderman, developed the Snap-Together interactive visualisation tool in 2000 [94]. Dykes created his CDV visualisation tool [40], Mondrian by Martin Theus was published in 2002 [140], and in 2004 Weaver published Improvise [148]. These early tools led to the initiation of the Coordinated Views in Exploratory Visualisation project, funded by EPSRC, which led to Roberts organising the first set of conferences that focused on multiple coordinated views (see cvev.bangor.ac.uk). The coordinated and multiple view conference series ran for five years from 2003 to 2007, inclusive. Several influential publications came from this series of venue, notably the state of the art in Coordinated Multiple Views (CMV) in 2007 [116], which has to date been cited over 700 times, as well as phrases such as multiform [112] to explain the different visualisations, and a model for coordination [17].

During this time, many researchers investigated brushing techniques, and several papers were influential in this domain. Most notable were Martin and Ward’s work on the XmDvTool [83], as well as Wong and Bergeron’s work on brushing in volumetric data [157], Hauser et al.’s [54] work on angular brushing, Li et al.’s [73] work on

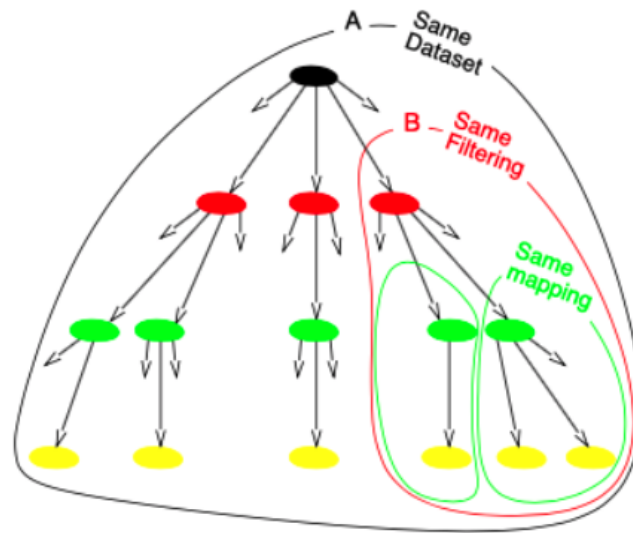


Figure 2.2: Picture from Roberts’ paper “On encouraging multiple views for visualisation”, showing how multiple views can be formed by changing parameters in the filtering of the data, or the mapping of the data, or the presentation and display of the information. Image ©IEEE [105] presented in International Conference on Information Visualisation (IV), 1998.

brushing versus dynamic queries, Doleish’s work on smooth brushing [39], Chen’s [27] work on compound brushing, and Wong and Bergeron’s work on wavelet brushing [156].

During the 1990s and 2000’s, researchers were creating many interactive tools, and exploring ways to interact with the data. Many influential visualisation systems were developed, such as AVS [141], IRIS Explorer, and IBM Data Explorer, using the dataflow model [52]). Consequently, researchers could use these systems to create multiple view visualisations. They loaded the data, filtered, selected and enhanced the data to create demonstration datasets, and then mapped the information into visualisations [52]. Indeed, the dataflow model creates a convenient way to consider how multiple views are generated [105]. Roberts writes

“Different visualisations may be formed through changing the data being filtered and selected; by changing the applied geometry or colourmap, at the mapping stage; or by adapting the display projection itself. Therefore, by taking this idea to its extreme, we may draw a graph representing possible multiple views .. [Figure 2.2], where a particular route through the graph represents a specific visualisation”, Roberts [105].

Roberts’ diagram is shown in Figure 2.2. It shows the extreme position, whereby different visualisations can be created by changing parameters to the visualisation

process. When any parameter is changed, a new view can potentially be created and displayed. This also demonstrates two principles: replication and replacement. Replication occurs when the values alter the visualisation and a new view appears. On the other hand, when, on a parameter change, the values affect the same view, then the visualisation is swapped to the new picture, and the information has been replaced. These ideas helped to develop later research on a model of view comparison [49], and we discuss these ideas in more detail in Section 2.2.6.

Researchers in the 2000s and up to the present time then started to use the techniques developed by these early researchers, and apply them to different domains. Researchers investigated and extended aspects of, and principles around, coordination and brushing. For example, Weaver investigated ways to coordinate and filter in his cross-filtered view technique that he implemented in the *Improvise* visualisation tool [150]. Koytek et al. created the *MyBrush* tool to investigate brushing and linking and bespoke interaction [66]. Piringer et al. [98] investigated focus+context visualisation with linked 2d and 3d scatterplots. And Covertino et al. [35] looked at context switching and cognition in dual-view systems. Qu and Hullman looked at how multiple views can be kept consistent [100]. Finally, Gaëlle et al. [103] explored how spatial distortion can be used across multiple views. Recently researchers have been interested in exploring how views can be used across platforms, for example Langer et al. [69] explore how multiple coordinated views can be used with large displays, or distributed displays [80], and how different senses and modalities can be used to help interact with multiple views [32].

Multiple view systems have been developed for many application areas. Table 2.1 below shows several areas where coordinated multiple views have been used. These systems provide examples of application areas. This is not meant to be a complete list of domains, but provides an understanding of the breadth of different domains. For example, **biological** data often uses network visualisations, and network visualisations are used to display connections between different illnesses, people and transmissions. **Medicine** is another area where multiple coordinated views are widely used. Applications have also been developed in the **energy** domain. Many developers link **geographic** views with other types of display. **Searching** for information on the web can be visualised in a multiple view environment. **CAD** (computer aided design) is an obvious choice for

multiple views, displaying different views on the 3d model. There are many examples where multiple views have been used to display **statistical** or high dimensional data.

Table 2.1: Multiple views have been applied to many domains and application areas. This table presents a summary of some of these areas.

Biology	<p>Lawrence et al. [70] developed the <i>exploRase</i> system for biological data.</p> <p>Graham and Kennedy developed <i>TaxVis</i> to examine sets of multiple classification trees [51].</p> <p>Craig et al. [36] developed coordinated parallel views for the exploratory analysis of microarray time-course data. They also developed a multiple view system of multiple trees [51].</p>
Medicine	<p>Aigner and Miksch worked on using multiple views to aid physicians in treatment processes [2].</p> <p>Martin and Aggarwal investigated volumetric visualisations from different views [84].</p>
Energy	Brehmer et al. [19] visualised energy portfolios.
Geospatial	<p>Dykes explored geospatial data with dynamic graphics [40].</p> <p>Brodbeck and Girardin used coordinated views to analyse geo-referenced high dimensional datasets [20].</p> <p>Plumlee and Ware developed the <i>GeoZui3D</i> system: a geographic visualisation system for ocean data, where users can navigate from different perspectives [99].</p>
Web searching	Roberts , Boukhelifa and Rodgers investigated multiform views for web-based search result visualisation [104].
CAD	Rosenman and Gero explored multiple views of design objects in a collaborative cad environment [121].
Statistics	<p>Ross and Chalmers investigated visualising multidimensional scaling algorithms [122].</p> <p>Matthew Ward developed the <i>XmDvTool</i> to display multidimensional data [145].</p> <p>Siirtola investigated parallel coordinate plots with the reorderable matrix [136].</p> <p>Weaver visualised multidimensional data using cross-filtered views in his <i>Improvise</i> system [149].</p> <p>MacEachren et al. [78] investigated visual analysis of multivariate data, by integrating several methods from information visualisation, exploratory data analysis (EDA), and geo-visualisation.</p>

2.2.1 What is a “view” (so that we can have multiple views)?

This section investigates word usage and meaning. We focus on the terms *multiple* and *view*. We focus on the terms multiple and view. By analysing these two terms, we are able to build a foundation that other concepts can be built upon.

To undertake this research, two bespoke sets of corpora were created from visualisation texts. We used *SketchEngine* [65] to create these corpora, and used the English Penn Treebank part-of-speech tagset. The smallest corpus, which we name *vTA*, was created from titles and abstracts of visualisation papers 1990–2016 [59], and contains a lexicon of 17,556 words, while the complete set of corpora consists of 282,619 words. This provided us with a small, readily searchable corpus, and contains data from 26 years

of visualisation history. Second, to gain a larger, six-year view of the written words used in the community, a large corpus was created of all IEEE VIS TVCG published papers 2012 to 2017, inclusive (which we named v6Y). This large corpus comprises 6,303,737 words, and has a lexicon of 134,663 words and 433,220 sentences. So as to investigate differences in phraseology of visualisation authors in comparison to other academic writers, a third corpus was used (our reference corpus) created from Open Access Journals (doaj.org) which we named oAJ. This was chosen because it has over 2.6 billion words from 2,971,481 articles, covering a breadth of fields including science, social science, medicine, technology and the humanities.

Generally, we use case-insensitive searches (e.g., a search for *view* finds *view*, *View*, *VIEW*), and we search for word lemma forms. For example, a search for “view” will find the following lexemes: *view*, *views*, *viewer*, *viewing*, etc. Our searches are also neutral to spelling alternatives, e.g., we treat *visualisation* the same as *visualization*. Throughout this article we refer to the headword, in this case *view*. we calculate term frequencies, collocated words and ngrams [65]. Collocations provide a way to investigate habitual juxtaposition of words. For example, from a corpus of recipe books, when searching for the word *drink*, we may find instances of *drink juice*, *drink coffee* and *drink tea*. There is evidence that *juice*, *coffee* and *tea* are beverages, and share the collocated verb *drink*. By widening the context of a lemma (e.g., by using a window of five words), we can investigate wider collocated terms; e.g., with the lemma *views*, we find *multiple views*, and *multiple coordinated views* and *multiple linked views*.

2.2.2 How frequently are *multiple* and *view* used?

Our hypothesis was that *multiple* and *view* were used more frequently in the corpora of visualisation papers in comparison to the Open Access Journal corpus. If we were to use our corpora to analyse the ideas underpinning *multiple views*, then we needed to confirm that they were representative for our task. To perform our analysis, we investigated raw frequencies and calculated normalised values (per million words) on *multiple* and *view* as headwords in vTA and v6Y, and in our reference corpora oAJ. Table 2.2 below shows a summary of our results, confirming our hypothesis: that *multiple* and *view* are used more frequently in visualisation texts in comparison to other academic articles. To further confirm this analysis, we looked at other words. Obviously, the

word *visualisation* should be more frequent: it is over two hundred times more likely to occur. Furthermore, we confirmed that many words were found in similar proportions in all corpora, where words such as *possible*, *theory*, *motivation* and *practice* are equally likely to occur in any academic text, as shown in Table 2.2.

Table 2.2: Part of speech (PoS) for *multiple* and *view*, highlighting they are more frequent in the visualisation corpus (v6Y) in comparison to the Open Access Journal corpus (oAJ). Raw frequencies, and per-million normalised count $_{pm}$ are shown. Non-visualisation words (typically found in any academic texts) show similar proportions across corpora.

Lemma	PoS	vTA	v6Y	oAJ	vTA _{pm}	v6Y _{pm}	oAJ _{pm}	$\frac{v6Y}{oAJ}$
<i>multiple</i>	adj.	366	5024	718921	1173.47	610.93	214.60	2.84
<i>multiple</i>	n.	42	679	46523	134.66	82.56	13.88	5.94
<i>relationship between PoS:</i>					8.7	7.4	15.5	
<i>view</i>	n.	462	11440	465899	1481.26	1391.14	139.07	10.00
<i>view</i>	v.	50	991	142670	160.31	120.50	42.58	2.82
<i>relationship between PoS:</i>					9.2	11.5	3.3	
<i>visualisation</i>	n.	3323	42273	71149	10654.22	5140.53	21.23	242.13
<i>possible</i>	adj.	131	3649	1147611	420.01	443.73	342.57	1.29
<i>theory</i>	n.	89	1340	454086	285.35	162.94	135.55	1.20
<i>motivation</i>	n.	5	265	92642	16.03	32.22	27.65	1.16
<i>practice</i>	n.	63	1267	502263	201.99	154.07	149.93	1.02

PoS	Parts of Speech
vTA	Corpus from visualisation titles and abstracts '90-2016
v6Y	IEEE VIS TVCG published papers, 2012 to 2017 inclusive
oAJ	OpenAccess Journals (doaj.org)
vTA _{pm}	Normalised count of VTA per million
v6Y _{pm}	Normalised count of v6Y per million
oAJ _{pm}	Normalised count of oAJ per million
adj. n. v.	Adjective, noun, verb

2.2.3 What parts of speech are used for *multiple* and *view*?

Meanings change depending on how the word is used in a sentence. In each of our corpora we discovered that *multiple* as a noun is used seven times more than as an adjective. Table 2.2 above shows the raw frequencies, frequencies per million and parts of speech (PoS) for each of the corpora for *multiple*, *view* and other specific words. Because there is a similar ratio in each corpus, we suggest that this proportion would be consistent across years. Likewise, in the oAJ corpus the adjective form is more frequent; in fact in this corpus it is over 15 times more common than the noun form, and

Table 2.3: Part Of Speech examples (PoS) for *multiple* and *view* published in IEEE VIS.

PoS	Examples	Source
adj.	colour to link information across <i>multiple</i> views	Koytek and Perin [66]
adj.	understanding the relationships among <i>multiple</i> objects.	Sarikaya and Gleicher [48]
n.	many works use small <i>multiples</i>	Fu et al. [46]
n.	employs a geographical map and small <i>multiples</i> of	Shen et al. [130]
n.	the viewer wants to change to a <i>view</i> that is	Sarikay and Gleicher [124]
n.	similarities and differences across <i>views</i> is	Qu and Hullman [100]
v.	participants had to <i>view</i> the data	Bach et al. [9]
v.	This paper takes a top-down <i>view</i> to understand	Sarikaya and Gleicher [48]

multiple as a noun is used less frequently. We hypothesise that the adjective/noun use in the visualisation literature is closer together because of the phrase *small multiples*. To investigate this hypothesis, we specifically looked at the tuple *small multiples*. From the 679 instances of *multiple* as a noun in v6Y, we find 323 (51.2 per million) from the phrase *small multiples*. The remaining tuples are varied and include “progressive multiples”, “3D multiple”, “nearest multiple [of ten]”, “[in the] X multiple” (where X refers to a specific visualisation type), whereas only 33 (0.01 per million) cases of the tuple *small multiples* were found in oAJ. This analysis supports the hypothesis that *small multiples* are more likely to occur in a visualisation context. Table 2.3 above shows several example sentences for *multiple* and *view*, and their part of speech from the visualisation literature.

The word *view* as a noun occurs 11.5 times more often than as a verb (see Table 2.2), yet only 3.3 times more often in the oAJ corpus. This supports our hypothesis that word *view* is used more frequently in visualisation texts than other publications. But also, authors write more frequently about a *view* as a visual depiction, rather than an alternative meaning of a person’s *view* or *viewpoint*. We also investigated modifiers of *view* and *multiple* and list the top twenty in Table 2.4. From this information, we see that authors frequently write about different view types, such as *3D view*, *map view* or *timeline view*.

The word *multiple* means *numerous*, *of great number*, *several* or *many*. In fact, the prefix *multi-* comes from Latin *multus* meaning much or many. There are many words that are prefixed by *multi-* that are meaningfully to be used in visualisation, including: *multicoloured*, *multicomponent*, *multifaceted*, *multiform* and *multi-use*. However,

Table 2.4: Frequency of occurrence, showing raw and normalised per million values for the first twenty examples of nouns and verbs modified by *multiple* and modifiers of *view*. In the v6Y corpus (IEEE VIS TVCG journals 2012–2017).

<i>multiple (adj.)</i>	f_{raw}	f_{pm}	<i>view (n.)</i>	f_{raw}	f_{pm}
multiple views	268	42.51	multiple views	268	42.51
at multiple levels of	101	16.02	the 3D view	181	28.71
multiple attributes	97	15.39	the timeline view	174	27.60
small multiple displays	80	12.69	map view	174	27.60
multiple variables	69	10.95	the detail view	172	27.29
multiple sources	68	10.79	the other views	162	25.70
at multiple scales	64	10.15	list view	157	24.91
multiple dimensions	61	9.68	matrix view	157	24.91
multiple instances of	59	9.36	different views	153	24.27
multiple features	59	9.36	detailed view	105	16.66
multiple users	58	9.20	graph view	98	15.55
multiple types of	55	8.72	the network view	97	15.39
multiple times	55	8.72	in a single view	95	15.07
multiple sets	52	8.25	the main view	92	14.59
multiple visualisations	50	7.93	slice view	86	13.64
multiple datasets	48	7.61	2D view	80	12.69
and multiple scattering	44	6.98	the summary view	80	12.69
multiple fields	44	6.98	feature view	75	11.90
in multiple ways	42	6.66	street view	73	11.58
from multiple perspectives	40	6.35	projection view	72	11.42

multicoloured and *multicomponent* do not occur in v6Y (in either English or American spelling). A list of the top 60 words prefixed by *multi-* are shown in Table 2.5 below. Another prefix that has a similar meaning to *multi* is *poly*. Words such as *polymorphic*, *polymerisation*, *polygons*, *polysemy*, *polynomials* are all common, but only 21 words that start with *poly* are found in the whole v6Y corpus. *Polygon*, *polylines* and *polynomial* are the most frequently occurring words in this category. The most useful word for our study, in this list, is *polyline*. While the word *polyline* is widely used, it is only used in the context of Parallel Coordinate Plots (appearing 18 times per million words). Finally, the suffix *-fold* means “of many parts” (source: Merriam Webster Dictionary), and therefore words ending in *fold* are potentially of interest to our study. Yet in v6Y the only *-fold* words are *manifold*, *twofold*, *threefold* and *unfold*, but unfortunately these do not have any word collocations with *views*, *display* or *visualisations*, because they are only used to describe theories, models and the structure of paper.

Table 2.5: Words that are prefixed with *multi-* in the v6Y corpus (frequency shown *per million*). In these results n-tuples are not included, but are shown in Figure 2.4, e.g., *multiple views* (without a hyphen) has a frequency of $(268_{raw}, 42.51_{pm})$. To understand scale, v6Y: 100_{raw} is 12.16_{pm} .

	f_{raw}	f_{pm}		f_{raw}	f_{pm}		f_{raw}	f_{pm}
multiple	4972	788.7	multifaceted	43	6.8	multiply	19	3.0
multivariate	1262	200.2	multi-view	42	6.7	multiplications	18	2.9
multidimensional	680	107.9	multilevel	42	6.7	multi-threaded	18	2.9
multiples	362	57.4	multimedia	40	6.4	multiplying	17	2.7
multi-dimensional	294	46.6	multiplied	39	6.2	multi-valued	17	2.7
multi-scale	180	28.6	multi-user	39	6.2	multi-way	17	2.7
multi-resolution	165	26.2	multi-field	36	5.7	multi-stage	16	2.5
multimodal	162	25.7	multiclass	35	5.6	multinomial	16	2.5
multiscale	125	19.8	multi-criteria	33	5.2	multi-step	16	2.5
multi-level	107	17.0	multi-generational	31	4.9	multi-core	15	2.4
multifield	94	14.9	multiple-view	30	4.8	multi-pass	15	2.4
multi-touch	90	14.3	multi-modal	25	4.0	multi-objective	14	2.2
multi-attribute	88	14.0	multi-focus	24	3.8	multiform	14	2.2
multi-class	67	10.6	multi-faceted	24	3.8	multi-layer	14	2.2
multiplicity	60	9.5	multi-chart	22	3.5	multiobjective	13	2.1
multiresolution	57	9.0	multitouch	21	3.3	multicriteria	12	1.9
multilinear	53	8.4	multi-pipeline	20	3.2	multi-volume	12	1.9
multitude	52	8.3	multiple-choice	20	3.2	multi-channel	12	1.9
multi-variate	46	7.3	multi	20	3.2	multi-fields	12	1.9
multiplication	43	6.8	multi-relational	19	3.0	multisource	11	1.7

2.2.4 What are the meanings of a *view*?

To investigate the word meanings, we used the Oxford English (OED), Merriam Webster (MWD), Macmillan (MMD) and Collins dictionaries (CD). *View* has many meanings in English; for instance, the OED has a list of 18 principal concepts and several minor descriptions. However, many meanings are very similar, and there are too many for our purpose. To reduce the number of meaning, we used affinity diagramming. We placed sentences on sheets of paper and through discussion investigated how they can be grouped together. We consolidated the results into five categories, which we label a to e (we use this convention throughout the chapter). We summarise their meanings below, and include a quote used in a paper from the IEEE TVCG papers in our v6Y corpus.

Meaning a. (manner) A *view* is a particular manner or way of considering or regarding a subject (OED). “From this *view* point, the overview visualisation does not only convey the overall picture of opinion distribution and diffusion patterns.” [159].

Meaning b. (query) A *view* is a selection of data from a database. It is data generated by a database in response to a query applied to existing tables, allowing the user



Figure 2.3: Wordcloud of our thesaurus of collocated words that are found in similar contexts to the given lemma *view*, in the v6Y corpus.

to select what data is displayed and how it is ordered. “Each widget provides data *views* with multiple tabs providing different information to the users” [79].

Meaning c. (pictorial representation) A *view* is a pictorial representation (MWD); it is a way in which a piece of text or graphics is displayed on a computer screen (CD). “OpinionFlow allows us to *view* the overall opinion distribution .. over time” [159].

Meaning d. (see) *View* is the ability to see something from a particular place (MMD) “Examples include *view* projections (pan and zoom settings, 3D camera viewpoint)” [62].

Meaning e. (opinion) A *view* is a personal opinion, interpretation, belief or attitude about a particular subject (MMD). “Different domain experts have different *views* of these cultural entities based on their expertise and disciplines” [162].

We advocate that each of these meanings is applicable and suitable for visualisation. Indeed, each is used in the context of *multiple views*.

2.2.5 What words have people used instead of “view”?

To explore this question, we used the v6Y corpus. We created a thesaurus of similar words. Each thesaurus was compiled by computing the similarity score between two

words w_1, w_2 , to find all overlaps where w_1 and w_2 share a collocation, and share a similar meaning, and they were ordered by their frequency. Using this thesaurus, we were able to investigate words that occur in similar contexts. Figure 2.3 above shows a word cloud of thesaurus we generated, from the lemma *view* (as a noun), with the word size dependent on the context score. Words such as *visualisation*, *technique*, *chart*, *plot* and *structure* can all be used instead of *view*. We note that there are limitations to automatically generating a thesaurus in this way because our 6 million dataset is reasonably small for this kind of analysis, so scored with lower results become less relevant and we receive more false positives. But even with these limitations, intuitively these results seem reliable, and they represent a list of possible synonyms for the word “view”.

In the thesaurus we generated, we observed a widespread use of words that have a general meaning, such as charts, plots, graphs, diagrams, images or pictures. One result is that authors often refer to the visualisation picture indirectly. In other words, rather than referring to a particular figure or scatterplot, they talk about the *algorithm*, its *interaction*, or a they refer to a specific *point* on the *graph*, or discuss the *layout* or their *results*. More importantly, though, we observe many specific (named) visualisation types being used. From our analysis, we discover that the most common visualisation type that is used in v6Y is *scatterplot* ($2318_{raw}, 281.9_{pm}$), followed by *histogram* ($1720_{raw}, 209.1_{pm}$), *timeline* ($1073_{raw}, 130.5_{pm}$) and *bar chart* ($892_{raw}, 108.5_{pm}$). Interestingly authors are over thirteen times more likely to write *bar chart* than *barchart* ($66_{raw}, 8.0_{pm}$), yet prefer *boxplot* ($321_{raw}, 39.0_{pm}$) over *box plot* ($43_{raw}, 5.2_{pm}$). Some of the named visualisations are composite types, such as *grid*, *matrix* and *trellis* plot. Other examples are *small multiples* ($323_{raw}, 51.2_{pm}$), *matrix view* ($159_{raw}, 19.1_{pm}$), *scatterplot matrix* ($120_{raw}, 14.6_{pm}$) and *matrix visualisation* ($92_{raw}, 11.2_{pm}$). Authors often name their new visualisation designs and reference them throughout their own paper. However, unless their technique becomes popular and is used by many people (such as that of treemaps [134]), the frequency of these new less-familiar designs will be low. It is beyond the scope of this thesis to carry out a full review of all visualisation types, and other researchers have investigated and classified different types, including the following: Bertin’s categorisation of diagrams, networks, maps, charts/graphs, tables/matrix, symbols, icons, glyphs and pictures [15]; Lohse et al.’s categorisation

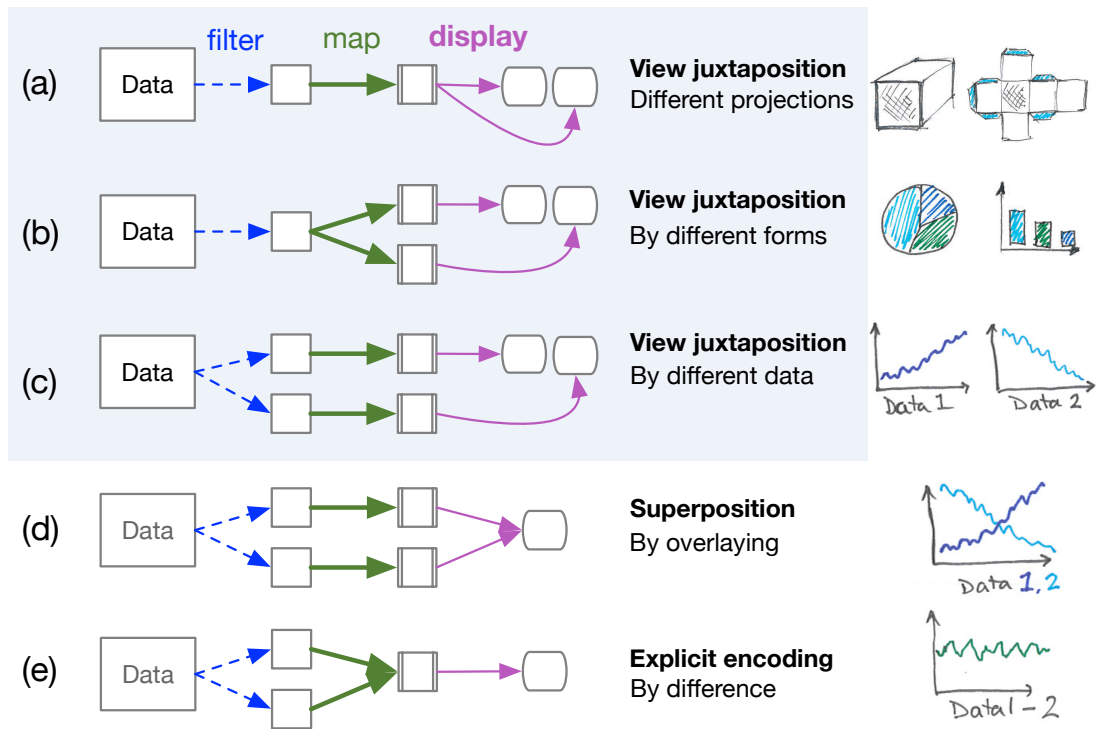


Figure 2.4: Multiple views can be created and displayed alongside each other in different ways: from different projections (a), changing the visual form (b), or by adapting the data (c). Views can also be created to be superpositioned (d) or merged through an explicit encoding (e).

of visualisation types [76]; and Lengler and Eppler’s periodic table of visualisation techniques [72].

2.2.6 Creation of views

To consider different ways to lay out the views, we first focus on how views are created. Multiple view systems are created for different reasons and there are many benefits of using a variety of views. Perhaps one view helps the user understand the data from one perspective, whereas another view is clearer for another task. Sometimes interaction is easier to perform in one view than in another, where, for instance, an alphabetic list of products makes it easier to select a specific named variety, whereas being ordered by price would enable the cheapest to be quickly located. Multiple views can also help users compare data, where for example several views show data across many different years, or from different geographic locations.

In fact, the dataflow paradigm [52, 141] provides a convenient model to conceptualise how different views are created, as seen in Figure 2.4. In the dataflow model, data is loaded, filtered and enhanced, then mapped to a graphical model, which is then displayed. Gleicher et al. [49, 48] describe three categories of view creation: *juxtaposition*,

superposition and *explicit encoding*. We can consider that view juxtaposition is a fan-out strategy, where changes in the visualisation pipeline cause a new *view* to be created and displayed in a new (separate) window. Consequently, view juxtaposition is the focus of our study. For completeness we include two final strategies can also be used to create a new view; these fan-in methods merge the data and view together into one display.

Juxtaposed views sit alongside one another. We highlight three different ways to create these views (see Figure 2.4): (a) The user can change how the objects are *projected*. For instance, architectural multiview technical drawings use up to six different projections (where the primary view has many auxiliary views, such as first angle, third angle, etc.). Alternatively, maps can be displayed using different projections (Miller cylindrical, Mercator, sinusoidal projection, etc.). Another way (b) is for the developer to display the data in different *forms* (hence the term *multiform views* [78, 51]). There are many different forms, including bar charts, line graphs, parallel coordinate plots, scatterplots, timelines, treemaps, etc. Finally, (c) by altering the data in some way, a different “view” will be created. In an interactive exploratory visualisation system (such as through dynamic queries) this can happen in two ways: either a change in the filtering causes the new data to flow into a current ‘view’ (the view is updated), or a new window is created that displays this new information. **Superimposed** views are overlaid onto the same window (e.g., consider two line graphs being merged into one visual). **Explicit encoding** methods merge the data and algorithms further up the pipeline to create specific algorithms (e.g., a difference visualisation).

There are several strategies to allow users to create different visualisations; from code development, visual programming paradigms such as from the module visualisation environments (e.g. [141]) to the many modern tools, developed by researcher, which allow users to display different visualisations in multiple windows, such as Improvise [148], iVisDesigner [102], Jigsaw [138], Keshif for tabular data exploration [160], Snap-together [94], Voyager [158] and XmdvTool [145]. In addition, the many commercial visualisation and statistical dashboard tools also support multiple views, such as Tableau, Qlik, SAS Business dashboard, Microsoft Power BI, IBM Cognos or Sisense dashboards.

2.3 Design concepts and general principles of multiple views

This section discusses the design principles and design guidelines in general, including how people design and look at patterns or pictures. In this context, a designer starts the design process by distributing and organising the parts of his design into the available space and filling the space. Then the designer draws the design in detail to indicate how it should look, and considers how people will look at it so he can organise the parts in order to achieve his goals. Likewise, the same process happens when developers plan to design multiple view visualisations.

Moreover, the designer can follow design recommendations and design rules during the design process; this will help the designer choose a suitable layout for his design, such as the Gestalt principles, which allow the designer to select the most effective design [63]. Gestalt principles have seven rules, namely:

- **Figure-ground**, where people look to objects as being in the foreground or the background, where the most important objects (the figure) are placed at the front, and the less important objects (the ground) are put at the back.
- **Similarity**, where the developer groups items that are similar to each other or have the same function.
- **Proximity**, where the developer places things close together when they are related to each other, and spaces them further apart if they are not related.
- **Common region**, where the developer put objects within the same closed region to group them.
- **Continuity**, where the developer arranges objects in a line or curve if they are more related than elements which are not on the line or curve.
- **Closure**, where the developer puts complex objects or arrangements in a single, recognisable pattern.
- **Focal point**, where the developer draws the most important object in a distinctive way that distinguishes it from other objects.

In addition, designers can follow the design rules or rules of thumb where the designer relies on his practical experience rather than theory [93]. Moreover, designers can

follow heuristic analysis to improve their designs, such as when a designer creates his design and asks an expert in the field to give him feedback and advice in order to improve it [153]. Furthermore, designers can follow Shneiderman's rules, which are a guideline to good design [133], namely:

1. Strive for consistency.
2. Seek universal usability.
3. Offer informative feedback.
4. Design dialogs to yield closure.
5. Prevent errors.
6. Permit easy reversal of actions.
7. Keep users in control.
8. Reduce short-term memory load.

Furthermore, design aspects grab people's attention to essential objects by making them brighter or bigger, by using colour, or by using design interface principles (light colour where the cursor is dark colour for other objects, focusing on one view). We also believe there should be more studies on the relationship between the design aspect and the writing orientation (reading gravity). For example, in the Arabic language, the written words are organised from the right to the left of a page. So potentially there could be differences in how designers understand, read and create aspects of design depending on their experience. Subsequently, any designer needs to consider the skills and experience of who will be looking at the design.

In addition, the designer needs to know about good design principles, and how the design will be used. Is it a global design (where people will see it across the world) or is it a local design (only seen in a local vicinity)? Bearing that in mind, we can decide the design aspect for each situation. For example, the designer can use reading gravity in a design so as to place the important objects in such a way that they will be seen first. However, in some cases, people should look first at an object which is necessary, and should look first to understand the important object; in this case, the designer can put the necessary object first, and then the important object second, following the reading gravity. Designers must note that there are different reading gravities across devices such as mobile phones and large screens [96].

Furthermore, eye tracking has been used to see which part of a picture people look at first (how people look at a picture) [91], where a heat map is created to depict how the eyes move across the design. Based on the eye tracking studies, a number of methods have been devised for placing objects on a layout, and designers can follow any of these methods. These methods can help inspire different design layouts.

- **z Pattern.** This method supposes that people look at the design and move their eyes in the Z path. For instance, they read along a line and when they get to the end of a line they quickly return to the beginning of next line. This concept is named the **Gutenberg pattern** [96]. Consequently, it would be useful for a designer to consider how someone looks at several pieces of information. It may be that a user of a multiple-view system would look at the views at the top left, and those at the bottom right more closely than those to the side. This is much like someone will read (in the West) in a left-to-right way, a z-pattern.
- **F Pattern.** This method supposes that people look at the design as it is, consisting of multiple rows, and they scan the design starting with the first row at the top from the left to the right of the row, then they scan the next row, and so on. This implies that people do not necessarily read or look at the end of the line, and their views get quicker and quicker as they scan down the page [91]. Consequently, it may be appropriate for a designer to position important views at the top.
- **Columns.** In this method, objects are placed in virtual columns, much in the same way as a spreadsheet is used. People can place multiple views in a gridded layout.
- **Golden Ratio.** This is calculated to be about 1.618, and is represented by the Greek letter phi. It is approximated by dividing sequential numbers in Fibonacci numbers $2/1$, $3/2$, $5/3$, $8/5$ and so on, with higher numbers getting ever closer to the golden ratio number. This pattern is found in nature, in spirals in flowers, petals, shells, and is often used in design. A Golden Rectangle allows two rectangles to be positioned side by side with proportions close to $2/3$. The $2/3$ layout is used often by designers to inspire designers. Consequently, designers of visualisation systems may choose to display two smaller views alongside one larger view, in a $2/3$ layout design.
- **Space Filling.** This is often used to pack information close together. For instance, Shneiderman in his Treemap algorithm [135] positioned each of the hierarchical

objects using a slice-and-dice packing algorithm. Every part of the display is used to display the information. Similar techniques can be used for multiple view visualisation.

- **Layouts in Wordpress** (and other web creation tools). These allow users to create the information, and the system re-organises the position of the items. For instance, an author can create text, include an image, and some ‘see also’ text in a sidebar. These tools provide responsive graphical user interfaces (GUIs). While a user defines the site’s layout, the tool provides the code that allows it to be viewed as defined on the large screen, but as (for instance) one column in a mobile device or small screen. In this way, the webpage changes the layout organisation in response to the needs of the user.

2.4 Multiple view tools and their use

Multiple view visualisations help people understand data, and multiple view systems help users interactively explore and compare information across many different projections.

We acknowledge the huge quantity of well-cited research that has been achieved in the area of coordinated multiple view systems, such as ComVis [85], Snap-together [94], and Waltz [111], where user interaction in one view is linked to another view (such as linked highlighting). It is through this linking that a user can better explore and discover interesting facts about the data. In this regard, researchers have created rudiments of coordination [17], researched linked highlighting and linked navigation [94], and linked brushing [12]. In particular, developers have created many types of brushing including: *compound brushing* [27], *multiple brushes* [146], and complex filtering operations such as through *angular brushing* [54] or *cross-filtered views* [148]. Other researchers have directed their attention to keeping multiple views consistent [100], or working across large displays [69].

Furthermore, there has been some research on multiview layout. For instance, Spotfire and IVEE were able to re-group widgets [1], and later Spotfire tools allowed users to drag and drop views. Likewise, Snap-together [94] snapped views together to create custom combinations, and Improvise [148] packs many visualisations into a tight space, while Keshif allows users to create visualisations in different parts of the Web page [160].

Layout is also controlled in dashboard visualisations, and there are many ways to create visualisation dashboards, including D3.js, highcharts, and in tools such as Tableau, SAS or Power BI. Tools like datahero.com can be used to create dashboards using a drag-and-drop interface, and small-multiples can be created in R's lattice or ggplot2. Shiny and shinydashboard or Highcharter (an R wrapper for Highcharts in javascript) can be used to create bespoke multiple view dashboards. However, to date there has been no systematic study investigating view-layout strategies, and no tool that focuses on multiview layout using a specialised grammar.

This section reviews the multiple view tools and systems, and their use. In general, multiple view tools can be divided into three categories based on their functionality. The first type of tools are the multiple view systems, those tools created by the developers to display multiple view visualisations. The second type are the multiple view tools which allow users to create multiple view visualisations. The third type are the tools which allow users to create multiple view visualisations using multiple view grammar.

There is a series of tools which allow people to create visualisation quickly, for instance Excel, D3, Adobe, Tableau and Vega-Lite. Moreover, there are other tools which allow people to create and lay out visualisations. For example, Becker and Cleveland created a tool to allow users to interactively explore and brush datasets [12]. The R tool creates layouts automatically (*data control*) [7], and the closest other tool is Snap-together where the views are snapped together by the user (*user control*) [94].

It is clear that developers have used many layout techniques which can be used for placing the visualisation on the screen. However, the user of the visualisation tool is forced to use the method specified by the developer without the ability to change the visualisation layout in order to be able to see the data effectively. Nevertheless, The developer can give users the control to change the layout in order to improve the visualisation display. In this case, there are two challenges:

- Auto layout method, which has to be determined at the first appearance of the visualisation.
- Provide the opportunity for the user to save the layout and apply it to other data.

Furthermore, multiple view tools are used to create multiple view layouts. For example, we can have a list of points which the layout tool changes into a diagram; and an application like Omnigraffle has an automatic layout tool (part of it), where a user can do something similar, something like a list. It will create a block diagram (as an example) with an arrow from one block to another block. The user can then restructure it in Omnigraffle by pressing a button, which can rearrange the data into a horizontal or a vertical or a tree structure. And there are numerous of online tools which can do that [95]. Moreover, the new version of Microsoft PowerPoint has a design button, where users can create a small outline, add text and images, hit the design button visible on the screen, and redesign all of it [77]. There are three types of layout tools based on who is controlling the process of creating the layouts:

- i) Manual tools, such as Visio [131], Omnigraffle, or any vector drawing packages [163].
- ii) Semi-manual tools. For instance, Omnigraffle is used to create diagrams, PowerPoint can create a diagram from a bulleted list, the Adobe Analytics tool creates different types of visualisations, and another such tool is Lyra Data Illustrator Charticulator by John Thompson [125].
- iii) More automated recommendation systems. For example, PowerPoint can create a diagram automatically - diagram me (like Tableau's Show-me button [101]) and the annotation tool can perform the same function [28].

In addition, there are multiple view grammar systems which allow users to use grammar to create multiple view layouts, such as:

- Iterated function systems (IFS) [11], Lindenmayer L-systems [123].
- The grammar-based system in Vega, Vega-lite [126].
- The grammar of graphics, Leyland Wilkenson [90]. R tool, R studio [3] and Shiny R [155], all based on the grammar of graphics.
- Positional systems: Left, right, up, down e.g., Java's gridBagLayout [165].
- File formats, and saving the structure of the visualisation, such as Web frameworks [42]. These are concepts around HTML such as Frames, Layers-overlay, CSS (structure, DIV), and JSON.

One aspect, in particular, that we feel should be debated and further researched is the question of who has control over the layout. If we examine programming languages, we find structures such as the GridBagLayout in Java, which help programmers to structure their layouts. We see panel layouts in web structures (such as top, left, main, right or bottom panel), and templates to help Web developers lay out their information. However, there has been little research into the best layout strategies for visualisation. Certainly, it will be a developer of a visualisation system who will determine how much control the user has over the tool, and over the layout of views. We propose five options for developers to determine the levels of control a user should have:

- **Developers can predetermine the layout.** This is a fixed strategy, and is usually reserved for fewer views or bespoke systems (designed for a particular purpose or user). For instance, in **two view** systems there is little choice: the views can be laid out left/right or above/below. Note that such systems are also known as *side-by-side*, *parallel* or *dual view* [99, 55, 89] systems, or if one view is more important, then *primary/secondary*, *focus+context* or *overview and detail* systems.
- **Views positioned according to data.** For instance, Roberts [105] positions the views according to a tree of data exploration, while the splom layout [26] (lattice charts) positions the small multiples to allow pairwise comparison of scatter plots, and Polaris [139] and the spreadsheet visualisation approach by Chi et al. [31] display information in grid-based layouts.
- **Group views that are coordinated together.** For instance, views that share a linked highlight, or linked navigation, can be positioned closely and perhaps in the same window. In other word, the type of coordinated manipulation can be used to control the positions of views. Roberts refers to these as “render groups” [105], and Weaver puts them side by side and visually connects them with lines and arrows [148].
- **The screen size can be used to determine the layout.** For instance, small multiples are laid on the screen in an order which wraps onto the next line, and as

the window size is changed so the quantity viewed changes, in the same way that a responsive/mobile-aware adjusts the content determined by the width.

- **The user can determine the layout.** Systems are often created whereby the user can drag (from a toolbox of possible visualisation types) and drop the selection onto a canvas, where the views are snapped to align together, such as with *Improvise* [148], *Jigsaw* [138], *Vinca* [47] and many other tools.

2.5 Theories and design guidelines for multiple view visualisations

To date, no quantitative research has been performed investigating the quantity of views or their layout, and there are no publications that present such results. This thesis aims to fill this gap. Indeed, as developers of, and writers about, coordinated multiple-views systems [111, 116], we have had first-hand experience of discussing questions with other researchers about the quantity and layout strategies of views, and students have also asked us similar questions, such as, “How many views should we have in our tools?”. However, we were unable to give a value and thus answered vaguely, saying “Enough to provide an expressive tool, but not too many to confuse a user”.

Consequently, it can be difficult for developers to know how to lay out their visualisation tool, or decide how many views to use. Ostensibly there is conflicting guidance. On the one hand, Roberts [105] suggests that “multiple views should be encouraged”. He encourages visualisation environments to be developed that can easily create a view, allowing users to investigate many different parameterisations. On the other hand, Baldonado et al. [144] say we should “use multiple views minimally”. They added, “A single view provides a user with a stable context for analysis; multiple views incur the cost of context switching”.

Yet both models can work together. They both have similar goals, namely for a developer to create usable yet functionally-rich multiple view tools. Roberts emphasises the term “lightweight”, saying that views can be easily thrown away [110]. His encouragement is directed the developer to create techniques to *manage* this exploration, and organise the potential explosion of views. On the other hand, Baldonado et al. put the onus

on a developer to critically think about the design and to limit the quantity of views used; they comment that a developer should “justify the user’s learning costs and the computational and display space costs of an additional view by appealing to the rules of diversity, complementarity, or decomposition”. However, none of these researchers provide a concrete number. How many should be “encouraged”? How many does “minimal” mean? Where is the balance between ‘view encouragement’ and ‘fewer views’, between creating systems that manage these views, or merely using fewer views? This thesis offers an attempt to answer these questions by quantifying how many views are used in practice, and how current researchers present screenshots on their tools.

However, research over view layout strategies in visualisation is very limited. We do note some recent research that has investigated phraseology in visualisation, which has included a number of multiple view phrases pertaining to layout. We note specifically the study by Isenberg et al. [60], who discuss, in particular, phraseology around focus+context, and the work of Roberts et al. [109], who classify terms in the general field of multiple views. What is clear from the related work is that more research is required in the area of multiple views and the layout of views, so as to develop better models and guidelines for visualisation designers. Indeed, it is unclear which are the most popular layout configurations, which is the main focus in this research.

Furthermore, there have been a number of papers that focus on the theoretical aspects of multiple views, such as rules and principles for the use of multiple views in 2007 [144], investigating juxtaposition, superposition and explicit designs in 2001 [49, 48], the phraseology of multiple views in 2019 [120], and even more recently the structure of view layouts [81, 82].

In addition, some researchers may have a good knowledge of one interpretation of multiple views, while not realising the breadth of the subject. They may not appreciate that there are different interpretations and concepts around multiple views. For instance, developers working in storytelling will be aware that users form different conclusions from the same results [129], but this concept may not be understood by someone who is programming or developing coordination visualisation tools. On the other hand, the programmer will know much more technical detail about how to code a CMV system, but may not realise about multiple interpretations. This thesis argues that to help

developers to create the best possible system they need to have a good understanding of a breadth of ideas. While researchers do realise that there is a broad set of words for this area, they may not understand what they mean or how frequent or infrequent they are actually used. Therefore, part of our study focuses on words and phrases from the literature, and aims to quantify how often they are used.

2.6 Summary

This chapter has focused on a review of various visualisation topics. It has answered five questions: “Q1/ What terms are used in multiple views, and what do they mean?”; “Q2/ What are the key moments of history in prior research of multiple views?”; “Q3/ What are the general design principles?”; “Q4/ What tools have been designed to create multiple view visualisations? ”; and “Q5/ What theories and guidelines (such as grammars) have researchers devised, which are conceded with multiple view visualisation? ”.

In addition, we have reviewed the multiple view tool and visualisation grammar, including the multiple view systems created by developers to visualise their data. These multiple view tools allow users to create their multiple view visualisations and layout tools. Furthermore, this chapter has revised the theories and design guidelines for multiple view visualisations.

In the following two chapters, this thesis explores multiple view visualisations in depth to find the popular structures and visualisation types of multiple view visualisations; this will help with creating design guidelines for multiple view visualisations.

Chapter 3

Data gathering and quantification preparation

This chapter focuses on data gathering and preparation for the experiments. The goal of this chapter is to explain how preparation for the experiments was made; by considering what data to gather and how to collect it. What is required is a suitable dataset of images (screenshots) of multiple view systems, such that they can be quantified. It is not only a matter of selecting images, but also deciding what images to select, and making the philosophical discussion on how to decide whether an image contains one or several views. The study focuses on answering the following questions:

Q1/ What is the strategy to select multiple view images to evaluate?

Q2/ What is the strategy to define and determine a *view* in multiple view visualisation?

Q3/ What are the strategies to code multiple view topologies and visualisation types?

3.1 Introduction to data gathering, quantification and preparation

Studying multiple view systems requires **a set of multiple view images (of graphical user interfaces) to judge**. The images need to be of a suitable quality so that they are clear to be judged and they need to display a data visualisation. It would not be suitable to capture all images from the research papers and supported material, because among them they would include low quality images, and images, diagrams and photographs, which do not present data. What is required is a set of images that represent visualisation

systems or solutions and are clear enough to make judgements over. Consequently, a strategy needs to be developed and written down, to allow consistent decisions to be made. The developed strategy would allow consistent decisions to be made across different files, articles and academic papers, and the written process would also be useful to help other researchers replicate the results.

In addition, to the goal of quantifying the multiple views, there needs to be a strategy that would allow researchers to define '*what is a view*'. It is not often clear to decide whether a visualisation system contains merely one-view system or contains several views. In other words, the philosophical question of 'what is a view' needs to be explored. What is required is a strategy to enable the clear and consistent **identification** of individual facets of a multiple view display. In other words, the defining strategy will be used to decide what is a single view, and is so counted individually, or there are more than one view.

Once views have been identified there needs to be a method to *code* the results. The very act of codifying the views helps to confirm and categorise them. The separate code will enable the types of views to be discussed, counted and recorded. Additionally, the consistent numbering scheme would allow the work to be discussed and other researchers to repeat the experiment. Consequently, a consistent code scheme needs to be developed such that quantification can take place.

The whole process is shown in Figure 3.1. The schematic diagram highlights the three main stages (1) selection and naming, (2) identification and (3) coding, along with their subparts, and the research questions they are answering.

The **first stage**, the **selection process**, starts with building the first database of all the PDF files from publications of IEEE VIS 2012-2018 Conferences. Then, builds the second database which just contains the papers, the workshops, and the posters that have clear and distinguishable screenshots for its multiple view visualisation layouts, which we use to build the third database that contains the multiple view images. Section 3.2 explains the **selection process**, and suggests a guidelines to filter the papers by selects papers that have clear multiple view images (stage 1.1), and the **extraction process** that copies the multiple view images from the papers then names the images (stage 1.2).

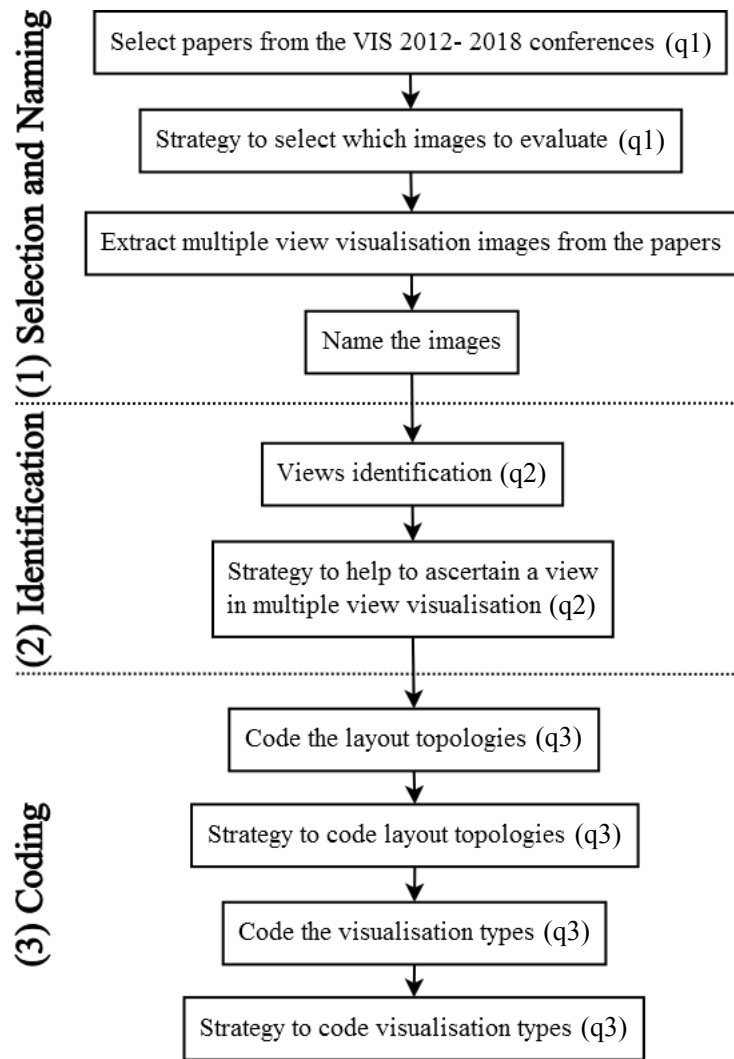


Figure 3.1: Methodology of data gathering and quantification preparation chapter.

The **second stage**, the **view identification process**, revolves around finding a strategy that helps with characterises a view in multiple view images. The strategy will be used to classify individual views. Section 3.3 investigates the **view identification process**, by explains what is a view that we can have multiple views, and develops a set of rules to identify the view (stage 2).

Finally, the **third stage** is the **coding process**, where sketches of the view layouts (from the fourth database) are made, and the types of each visualisation view are recorded. Section 3.4 explains the **layout coding process** and Section 3.5 explains how to **code the visualisation types** (stage 3.2).

3.2 Image selection and storing data

What is required is a general set of images, captured in a way that can be replicated by other researchers. This is a *representative set of images*. Each image needs to present a result of a user displaying some kind of data. To gain a broad view, the images are treated as a bag-of-images. (Much like in linguistics word analysis is often treated as a bag-of-words, the idea here is to create a set of images that can be treated in any order). Metadata is kept such that when it is required the type of data or the task for the data-visualisation can be analysed. But at the start, the concern is towards the image layout. For instance, treating the views as merely ‘images’ permits a separation of ideas: on the one hand how they are laid out, and then looking further at the metadata to classify it as (for instance) a scientific visualisation (SciVis), information visualisation (InfoVis) or visual analytic (VA) tool.

In fact, the community has previously classified the visualisation tools by SciVis, InfoVis or VA, it actually does not seem a suitable situation to classify the visualisation tools in this way. This is because, not only the distinction whether something can be classified as information visualisation or scientific visualisation is often unclear, but also the community itself seems to be moving away from this classification. This is demonstrated perfectly by the recent of the IEEE Visualization Conference to be named IEEE VIS, rather than of InfoVis, SciVis and VAST. Finally, it is not suitable to keep photographs, schematic diagrams, or icons or symbols. It is critical to only include images that represent ‘data-visualisations’ that have been designed by a human being to display data. Consequently, what is required is a set of images that are:

- **Representative.** While the selected images are a sample of all possible visualisation images, they need to contain a cross section of different types of visualisation interface, such to be classified as a representative sample.
- **Reproducible.** The bag-of-images need to be reproducible, such that other researchers could re-create the same database themselves. Explaining the capture method enables other researchers to follow the methods and reproduce the set. This would enable other people to validate the results, and the guidelines that are

created from the analysis. It means other researchers can capture the same set of images and to be able to classify them in the same way.

- **Of good quality.** What is required is a set of images that are good quality and have a good resolution. When the image resolution is too poor it would be difficult to classify them. As an estimate the images need to be greater than 640 pixels.

There could be several ways to collect images of different visualisation systems. One way could be to use a general search engine (such as Google) to create a list of images. However this strategy would capture images of a wide variety of quality. It is also difficult to understand how they were created and for what purpose. Furthermore there are lots of examples of bad visualisations on the web and it would mean that one of the first tasks would be to decide on the quality of each visualisation. This would not only be time-consuming but the process would be difficult to recreate by other researchers. While exact urls could be captured, which would help in the reproduction process by other researchers, and often many of the websites do use permalinks, it would still add unnecessary complications to the capturing process. On the other hand, specific websites such as Wikipedia or the specific infovis-wiki (infovis-wiki.net/wiki) could be used to capture the images. But, these websites use visualisations to explain specific topics, and would not necessarily create a database of representative images. In other words, they have been specifically crafted to help tell a particular story.

Another strategy could be to capture images from visualisation tools. For instance, searching for visualisations that have been developed by D3.js, RStudio, ShinyR, or Tableau, for instance, could provide a specific list of images. Indeed, it could be possible to collaborate with the Tableau research team to gather specific visualisations that have been created by their tools. But again, in each of these cases, the generated set of images would not necessarily provide a representative set of all visualisation tools. Furthermore, each of the tools (JavaScript-bases or Tableau and so on) would have an implicit bias in how they were created. This is because each of the tools only offer specific functionality, which would constrain how the visualisations are laid out. In other words the selected images would only represent layout schemes that were offered by those tool developers.

Another way, which was eventually chosen, was to capture images from academic papers. Conferences and publication venues such as IEEE VIS (visualization), ACM CHI, EuroVis or journals such as IEEE Transactions on Visualisation and Computer Graphics, or Computer Graphics Forum could be used. There are many advantages of using this strategy. Because these papers are placed on digital libraries, they can be readily captured and stored. The list of papers is also easily stored and so can be used by different researchers. The resulting list of images would be reproducible by different researchers. Second, on the whole, the images in these papers are of good quality. They have been through a review process, which will guarantee that the images are of a reasonable quality. Third, because they are used to present research results, they represent the current state of the art and use by researchers. There are some limitations with this strategy. First the papers only represent the results from the academic community. There are some excellent examples of non-academic researchers creating very creative visualisations that would be missed. Also, because academics are taught by other academics they would only present the ideas from perhaps a small narrow group of society. Third, the academic papers require that they are formatted in a particular way. This could affect the type of visualisation tool that is described, and the way that it is presented. After much deliberation and discussion, it was decided that research papers would provide the best and most suitable set of images for the study.

This investigation started by considering all papers presented at the IEEE visualization Conference between 2012 and 2018. This period was chosen because we wanted to focus on modern tools, rather than history systems, and also for convenience because we had the files available. In addition, the conference folders include all workshop, tutorials and other information, making it a large, broad and reproducible dataset. However, taking all images from the IEEE VIS Conference creates a vast corpus of information. It consists of TVCG papers, conference papers, posters and supplementary materials. Consequently we decided to reduce the quantity of images selected. Practically, we wanted to have a diverse set of visualisation images that this study can evaluate. However we realised that could only evaluate a few thousand images, so we needed to limit the years of evaluation and a strategy to reduce the quantity of the images to evaluate. In fact, there are over 3392 PDF files on the seven years of USB memory sticks from the IEEE VIS Conferences. Estimating that each file probably has more than one image, and that not every image presents a screen capture of a visualisation tool, a consistent

strategy to select suitable images was required. This was needed also to reduce the quantity of images to evaluate to make it possible to review. After deliberation and experimentation, the selection strategy was to keep PDF files that had suitable images, before extracting the images from the files.

These papers have been filtered by choosing just the papers which have visualisation that achieved the following evaluation criteria: (1) the image should be clear and >640 pixels, (2) it needs to show a complete visualisation layout, and (3) the visualisations need to be applications or websites rather than the visualisations created by gathering snapshots for individual visualisation.

In addition, it is important to include images of visualisation tools, and to ignore images that have been edited or manipulated through an image processing program (Photoshop, Gimp and so on). The set of pictures need to be selected where they are judged to be original, directly taken from the visualisation tool, and not created or adapted significantly to fit in with the paper formatting guidelines. Some images are clearly adapted, while others have more subtle changes applied to them. What is required is to investigate proxy indicators. Perhaps the same visualisation picture exists in different media such as a slide in PowerPoint, or when the paper is in two-column format and it is a wide-screen tool, cut into separate parts. When investigating the different images, other traits were noticed. For example, editing or misalignment of separate parts to the image, which may imply that image editing software had changed the image. Ideally, what is required are pictures that were clearly screenshots of a tool. However, if there were any indications that manipulation may have taken place, then these files were removed from the list.

After this filtering process for all the papers, 473 papers were saved. The names of the papers were changed, by following the method of using the number of authors' surname (Authors) in each paper, and the year of published. This **selection process** is defined by five stages, as shown in Table 3.1. This process can be used as a strategy, by other researchers, to select multiple view images in multiple view visualisations analysis.

One of the challenges we faced was to decide whether to include images of systems with only one-view. On the one hand a one-view visualisation does not necessarily represent

Table 3.1: Strategy to select which multiple view images to evaluate.

1	Removed all files of supplementary materials.
2	Removed papers that did not have visualisations, or only had illustrations and schematic diagrams.
3	Removed papers that only had images that were clearly put-together or had been edited (by an image processing tool). Telltale signs were investigated. These included: miss-aligned sub-images or several image resolutions in different parts of the figure. Papers were removed that had displayed their images from several sub-figures, for instance, authors often take several screenshots of their tools and put them together in several sub-figures. This decision was taken because it is difficult to quantify how much editing had been achieved by the authors.
4	Removed papers with low resolution or very small figures, where it is difficult to determine the views inside these figures, or it is hard to characterise the visualisation types within the views, which would have been unclear to classify.
5	Files were kept that had at least one candidate image.

a visualisation system. It could be created by a user, and itself could actually be a cut and paste from another multiple view system. However when they are not included, it is more difficult to calculate suitable statistics over the average of views used. Originally, the decision was taken to not include them. But after publishing a poster paper at the IEEE Visualization Conference, and receiving several comments and questions on the work, the decision was taken to include them. Notably, one eminent visualisation researcher questioned deeply about ‘what is a view’, and pointed out that if one-view systems were replicated in a multiple-view visualisation system, then one-views systems where they stand alone, should be considered. They asked “Why didn’t you include one view systems?”, and it was hard to discuss and convince them otherwise. They recommended adding one view systems to the data set. And after deliberation, the same conclusion was made: to consider single-view systems equivalent to multiple-views systems. Therefore, one-view systems were henceforth included in the short paper publication “Towards Quantifying Multiple View Layouts in Visualisation as Seen from Research Publications” [82] and this thesis.

Furthermore it is statistically relevant to include the 1-view systems, because it gives us an appropriate baseline. Moreover, it gives us the ability to understand and tell a story about what people are doing with one view systems comparison with the two, three, etc. view systems. Finally, one-view systems are not a majority, they represent a small

proportion of the whole (69 from 491 layouts), and so this inclusion helps express a richer story.

Through this sifting process 473 papers were kept, and one image extracted from each, apart from sixteen papers that had pictures for two different tools and one paper that had pictures for three different tools. Works were included from SciVis, InfoVis and VAST without exclusions based on applications area.

Following the selection process the images needed to be extracted. While automatic schemes could be used to save the images, for instance Adobe allows all images to be saved from a PDF, the decision was taken to save them manually. This also meant that the names of the files could be changed at the same time, and stored in the desired image format. Images were saved in PNG format, and a bank of 491 images, of screenshots of selected multiple view visualisation tools from the chosen papers, were stored in year-based folders. Each image was labelled with a unique abbreviation using a consistent file name based on the surname of the authors, that is used in \LaTeX to cite the papers (as previously explained). The file formatting strategy, and examples are shown in Table 3.2.

Table 3.2: Strategy to select which multiple view images to evaluate.

	Author1[-Author2[-Author3][-ETAL]]Year.png
Where:	Author: represents the surname of the author Year: represents the year of publication
For example:	Erbacher2012.png [41] This image was extracted from a paper which has only one author. Lehmann-Theisel2013.png [71] This image was extracted from a paper which has two authors. Hong-Lai-ETAL2014.png [56] This image was extracted from a paper which has more than two authors, and these are the surnames for the first two authors. Kucher-Kerren-Paradis-ETAL2014.png [67] If there are papers shared the same surnames for the first two authors, in this case, we include the surname for the third author.

The naming convention meant that it is easy to reference the images, locate the associated publications and cite them later. If there were several suitable images which were different, then both of them will be collected, and added a F (for Figure) followed by

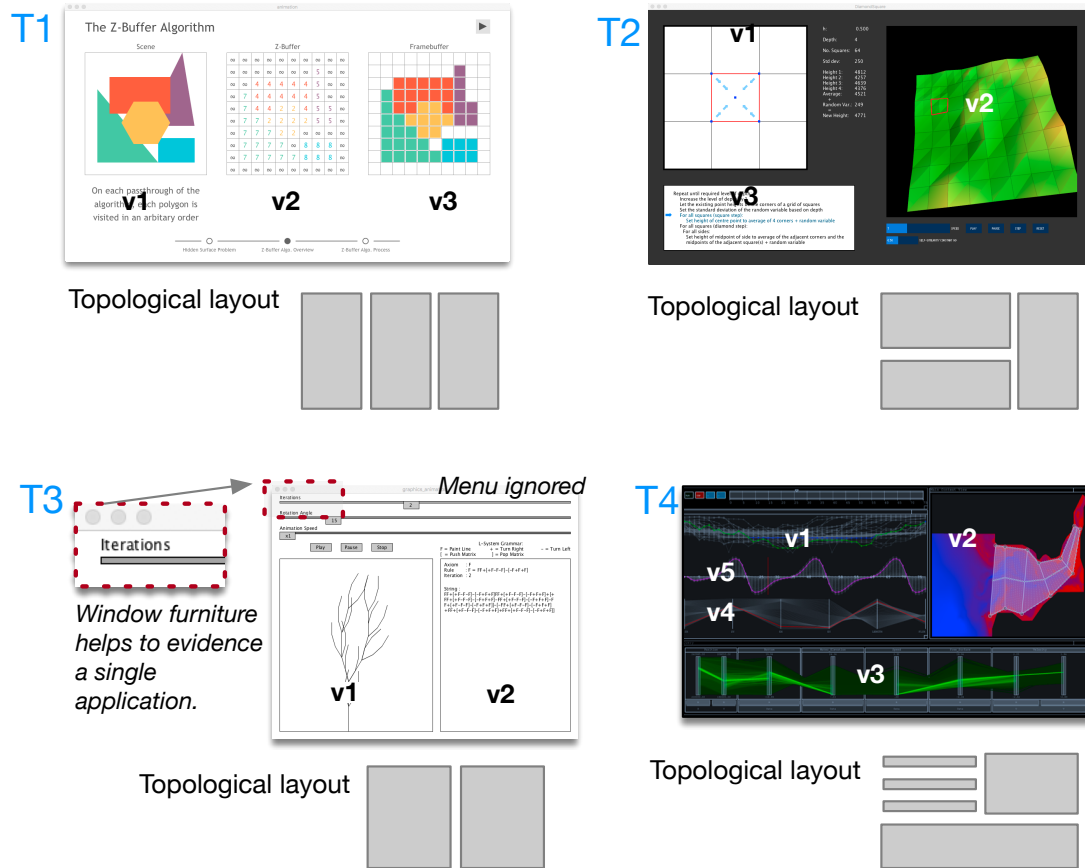


Figure 3.2: Four visualisation tools (T1-4). Visualisations T1 and T2 clearly show three views, while T3 has two windows and a menu that is ignored (three explanatory visualisation tools created by our students [107]. T4 shows the Vinca estuary visualisation tool [47] demonstrating five views.

the number, to the file name ($-F1.png$, $-F2.png$, etc.), unless the images were similar where we chose the first image only. This process resulted in a set of 491 images. Through this approach we gained a wide cross-section of application areas, data types and display styles.

3.3 Developing general guidelines for view identification

Now a representative set of suitable images has been stored, the next step is to classify and code them. But ‘what is a view?’ How is judgement made over the image represents a one-view system or something that is a montage of sub-views? Is the system named a one-view tool or a multiple-view tool?

In visualisation, a “view” is the basic component of the visualisation layout. The question “how many views are used in a multiple view system” along with the corresponding question of “how are they laid out” has been on the minds of many researchers for many years. Consequently researchers have discussed long and hard over the subsequent question “what is a view”? However, while these discussions have been made by researchers, they have been done at conferences and during the conference breaks. It is surprising to know that there has been no formal record of these discussions, and no formal strategy has been written down to discern whether (for instance) you may have a one-view or two-view system.

To address these issues, and to code the multiple view layouts it is important to identify an individual *view* in a swath of adjacent and other views. It is not necessarily easy to identify, or count, views on a visualisation. Sometimes it is clear, that there are separate dividing parts to the view, that for instance, one visualisation is a scatterplot and another a bar chart. But other times it is less clear how many sub component parts, or facets, the visualisation contains. Furthermore, designers can also overlay visualisations, or even place visualisations in an irregular way which can complicate deciphering the layout. However, while complications exist in how to interpret a view, because developers want users to understand their data display, they are deliberate in how they position their views. They want to craft decipherable and understandable visualisations that are laid out in a clear way.

In addition, designers will engineer the display such that the graphical marks stand out from the background and contain clearly perceptible parts. In perceptual terms, the background is known as the ‘ground’ and the marks that are displayed ‘the figure’. Figure and ground are often discussed in perception, for instance Colin Ware discusses *figure-ground* [147]. The graphical marks that encode the data are the *figure*, which stands out from the background colour, texture or shape. Even if the background is dark and the graphical marks light in colour, the user should perceive that they are different.

Therefore the first discernment strategy is to consider elements of *design*. When glancing and looking at the screenshot it is possible to notice the spacing that a designer has used to separate the different views. In most cases a view will be surrounded by a light colour, or small gap. Moreover, when a designer creates the tool they will put

things that are associated closer together. They will place graphical marks in visual groupings. For instance, rectangles make bars for a bar chart, which are placed close together to make the bar chart design. It is often clear to realise that they are part of the same “bar chart” because they are close together. These concepts therefore are following the Gestalt principle of *proximity* [147]: things that are close together and probably more closely related. So the designer will follow best design principles, and consequently position the chart pieces close together. This means that the graphical marks, bars, axis, tick marks, titles, and labels, etc. will all be close together to make one view. Alternatively the designer will make this explicit by placing a view in a User Interface window, panel or frame: they will enclose the whole visualisation design inside a border (this presents the Gestalt principle of *enclosure*).

For example, Figure 3.2 (T1-4) shows four visualisations. The first three visualisations show explanatory visualisations [108]; this multiple view visualisation demonstrates explanatory visualisations of T1 the z-buffer algorithm, showing three different projections; T2 fractal surfaces algorithm, and T3 Lindenmayer grammar-based modelling. Although the views inside visualisation tools T1 and T2 are not displayed in separate windows there is clear separation (space) between the multiforms. T3 makes this separation explicit by using a box around the each view. Finally T4 shows a five view system of the Vinca estuary visualisation tool [47], each view is a separate window that can be positioned by the user, and there is a subtle bounding box around each view and clear space around the map, Parallel Coordinate Plot (PCP) and three line graphs, creating a five-view system. Alongside this figure we include our abstract topology of its design drawn as gray-coloured rectangles.

Another strategy is to consider that each view affords a different task. This is sensible, because again the designer will want to get the user to focus on a particular task in one window. Furthermore a developer would often create the tool in a modular way. One design principle a developer may follow is to apply *separation of concerns*. Where they will implement the code in separate classes or modules. View would be implemented by a different class, or one module would deliver spatial interaction (such as navigation on a map view) or another module focuses on searching (such as used by a list view). This represents a ‘separation’ of tasks. For instance, in Figure 3.2T4: the map (v2) shows a view of an estuary and affords geo-spatial interaction; v3 is a parallel coordinate plot

(PCP) allowing multivariate data manipulation; the line charts (v1,4,5) show temporal data. This ‘separation of concerns’ idea can help to discern individual views.

When the views become very small it can be challenging to count them. This is especially true when there are many views, such as displayed in a matrix, Scatter Plot Matrices (SPLOM), trellis, grid or small-multiple display [109]. When do these small-multiples become one ‘view’ or when are they separate and individual views? And, how can we make the judgement that a small-multiple-view is actually a single view (yet made up from many small parts) or that each cell is a separate *view*?

While there may be different opinions [109], the decided strategy was to ask the following key question: “if an individual small-multiple was removed from the whole plot, would it make sense?” In other words, was it an independent or dependent view? At one extreme, it is possible to imagine that an individual cell of a reorderable matrix [137] or a cell of a spreadsheet would not make sense on its own. But, with some scatterplot matrix plots, it may be possible to remove the cells and they would make sense on their own. Therefore, a scatterplot matrix is classified as a matrix view — count the cells and group it with other views having “lots” of cells. Matrix views have structure, where the position of the cell has a specific meaning.

With a grid of small multiples, their order is less strict, and the same question needs to be asked: do the small-multiples make sense on their own? This answer probably depends on the size of the individual pictures; when they are very small, they probably will not have their own axis, legends, labels, etc. (where this information may be displayed once for several visualisations) and so will not make sense on their own. But if they are comprehensible on their own, then we will count the views. In fact, as humans we prefer to see simple shapes and therefore we abstract complex shapes into simple groupings, we also perceive things close together as being connected or similar. Designers will probably follow the Gestalt principles, and will use them when they create their visualisation tools.

Most visualisation tools have interface components, and may include menus, buttons, slider bars, legends, colourmaps, legends, etc. Sometimes these are integrated with a view, sometimes they are shown in their own window. In most cases it is possible to

ignore these menus. However sometimes the menus take up a significant space. These facets could be coded as a “menu”, however this would not permit the coding to be applied consistently, because other views have this interaction/menu integrated into the views. It could be coded as “null” views (or information panels), but again this may skew the results. Consequently, a multi-criteria solution was chosen. If the menu is on the side, or along the top, and can be easily ignored without changing the topology of the view layout we ignore it. If a menu-window is enclosed between other views then we treat this as “null” space and merge it into the closest neighbouring view. This allows all menus to be treated as part of (at least) one view, and every visualisation treated consistently.

To perform meaningful and consistent manual coding of view quantity the rules needed to be laid down, which is named the **view identification** process. Five rules were identified. These determine ‘what is a view’, and researchers can use these rules as a strategy to code the multiple view layout in their research. They are listed in Table 3.3.

The rules were printed, kept close, and especially referenced with a visualisation that was difficult to judge.

Table 3.3: Strategy to help ascertain ‘what is a view’ in a multiple view visualisation.

1	Views are usually visually separate from another view. Count the views that are clearly separated by spacing, a gap that is coloured in the background colour, rendered in a rectangle, or placed within a window. For instance, many views are encapsulated in a window, notice a window through its furniture (such as a cross to close that window or tab).
2	Views have different tasks, there is a “separation of concerns” where different views are added to gain a variety of perspectives on the data. Count the views that clearly have different tasks.
3	If we can name them, we have different views. Point to the views and name them. By naming the views you are treating the view as a “whole”. For example, you could say “scatterplot, line graph and bar chart” and you would have three views.
4	Consider how a programmer would code it. If they cannot be separated visually, they may be able to be separated functionally. This is separation of concerns at the <i>functional</i> level, e.g., <code>draw.scatterplot()</code> .
5	Ignore interface components. Ignore menu windows if it is sensible to do so (such as a menu along the top of all views). If controls, menu items or legends are part of their encapsulating view, ignore them (they are part of that window anyway). If the menu is contained in a separate window that is located in-between other views then ignore the menu and merge it’s screen real-estate space with the closest neighbouring view.

3.4 Coding the layout arrangements

The next challenge is to provide a ‘code’ to enable the views to be counted. This section discusses the coding scheme for the first part of third question (q3): “What is the strategy to code multiple view topologies?”

To answer this question each image was carefully and systematically evaluated in turn, considering the topology, and sketching layouts. The coding schemes can be used for coding individual layouts, and that will help to address research question Q2, Q3 and Q4. Now, being able to identify a “view” this study needed a way to record the views and their configuration.

An inductive strategy [44] was chosen to develop codes to answer research question Q2, Q3 and Q4. The codes were developed through refinement and critical thought. This study used two investigators (student and supervisor) to evaluate the images. In order to explore the different possibilities we decided to start with a sketchy visual coding scheme. This enabled the codes to be created as the analysis happened. Sketching (of the layouts) was also used to externalise the thoughts and structures on the visualisations. These sketches were particularly useful to explore difficult layouts.

To proceed, each image was displayed in turn on a large screen and carefully judged. Each of the 491 images were analysed in turn. For each image, several judgements were made, and notes on the quantity and layout of the views of a multiple view image were taken. A simple sketch was made, as a simple representative picture of the layout. These sketches represented the topology and the ‘figure ground’ principles were used to judge them. If a quick judgement could be made then one simple (indicative) sketch was made on a piece of paper. However, if a judgement could not be made and the topology was unclear, then every possible topology was sketched. The image files were evaluated as follows: framing each view in the visualisation layout and after removing the container (using the figure-ground method), after that, the structure of the layout for each multiple view image was evaluated, and every sketch was also labelled with the paper reference. Then, the sketches were organised into sheets in according to the year of publication and in alphabetical order as shown in Figure 3.3. If there was a dilemma on how to sketch the topology we drew all possible arrangements, which were later

discussed. This meant that the ‘codes’ were developed as part of the judging process. The strategy is summarised in Table 3.4.

The alternative way would be to follow a deductive coding scheme [44] and pre-choose the categories by calculating in advance the topological permutations and then count. The deductive method was not chosen, because at the outset it was unclear on what structures were being used, and it would have been impossible to discover possible new schemes. In addition, because of the need to emphasise the role of the visualisation designer in any guidelines that would be created, it was decided to make sketches of the layouts, which act as visual codes. These sketches would be exchanged for labels at the end of the whole process and stored into a spreadsheet.

From early discussions about the topology, and initial sketches, it was realised that some images were easy to judge, while others were not clear. Therefore a training phase was employed. The two investigators classified individually 20 (randomly chosen) images, based on the criteria as given in Table 3.3, and sketched small pictures representing the topology. Out of this set, five were unclear. But after discussing these cases, the topologies were readily agreed.

Table 3.4: Strategy to code layout topologies.

1	Every image was displayed on a computer screen.
2	Judge topology (using the view rules in Table 3.3)
3	Code the layout in sketches, write file name alongside.
4	With dilemmas, sketch all possible layout configurations.
5	Discuss each dilemma, agree on one topology, and keep agreed topology.
6	Cut the sketches into individual <i>tiles</i> .
7	Arrange tiles on tabletop, discuss ideas, and categorise layouts.
8	Record quantities.

While encoding the data we ignored the size of the facets. Our goal was to focus on multiple view layout design, and our topological coding scheme (as explained above) ignores recording the exact sizes of the facets. In other words, the structure of the layout was evaluated, ignoring their relative sizes. There could be different ways to code the different sized parts of the views, such as having a sub code, or a percentage value to determine the split. However, this extra information would complicate the coding scheme making it more difficult to be consistent in coding and also the analysis would be more challenging.

In addition, the relative sizes are implied in the structure: where the size of an individual view, of a multi-faceted visualisation too, would naturally be smaller. Therefore the sketches code the topological layout and hide some of the fine nuances that may occur due to size differences of some of the views. For example, when considering a side-by-side two-view display with the left view taking up less space than the right view to be structurally the same as another visualisation that has equal 50/50 split of the size of the each view.

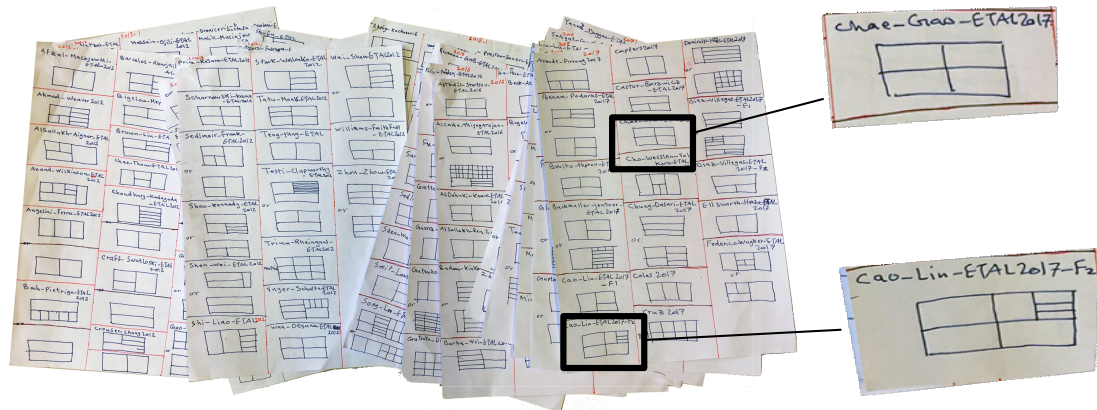


Figure 3.3: We sketched each topology in a small picture; we sketched all possible alternatives, which were later discussed.

When evaluating all 491 images, over 22 sheets of paper of sketches (with on average over 22 sketches per page) were created. At the end, two people went through and discussed each design, and we focused on the sketches which were ambiguous. We discussed 124 of these ambiguous cases in detail. After discussion we reached a final judgement on each layout and agreed their topological structure, and updated our sketches. Random checking of another ten was achieved to make sure that agreements were made over the necessary judgements. A photograph of the sheets is shown in Figure 3.3.

Subsequently, 491 sketches were judged. These agreed sketched images were then cut up into individual **tiles**, keeping only the agreed topologies, such that we could discuss them and move them around on a tabletop. By physically moving these tiles, it was possible to have a frank discussion about their layout strategies. The simple abstract drawings provided an easy way to compare the structures without being distracted by the actual view design or content.

To record these classifications, an appropriate nomenclature was needed. This nomenclature is described now, because it is part of the coding scheme, but practically it was developed coincidentally with the recording of the data. The shorthand version occurred because we needed an easy way to refer to each layout. There are different potential ways to name the view layouts. Real names could be used, such as “one view”, “dual view” or “three view”, but this strategy would not make a convenient shorthand version, because ordering of them would be difficult. The solution was to label them with a number (the view quantity) followed by a letter (indexing a view layout).

For example, “1A” is a one view (there is only one possible layout), and “2A” is a vertical 2-view layout, while “2B” the horizontal layout. While the number is logical, it is unclear how to allocate letters to view configurations. The chosen solution is arbitrary, but consistent and convenient. It was decided on a first-come basis. When counting the views, the first new type was allocated an A, the next unique type a B, and so on. The naming scheme is listed in Figure 3.4.

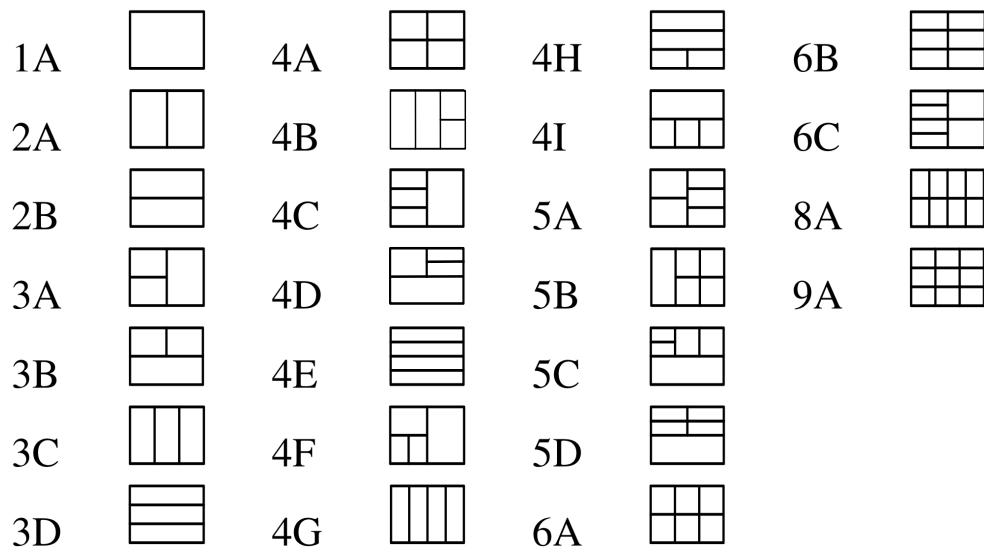
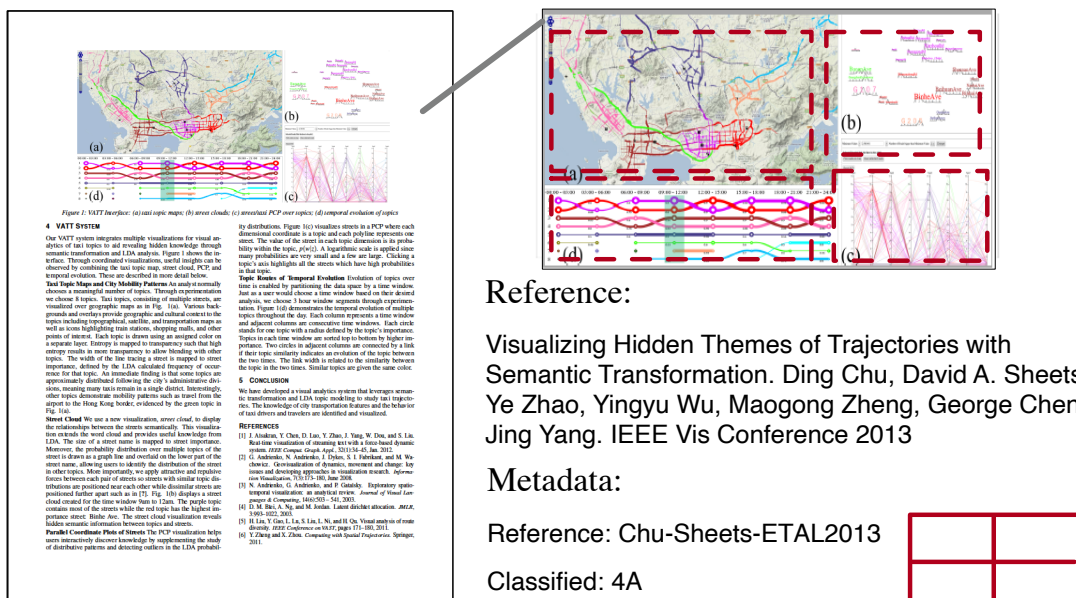
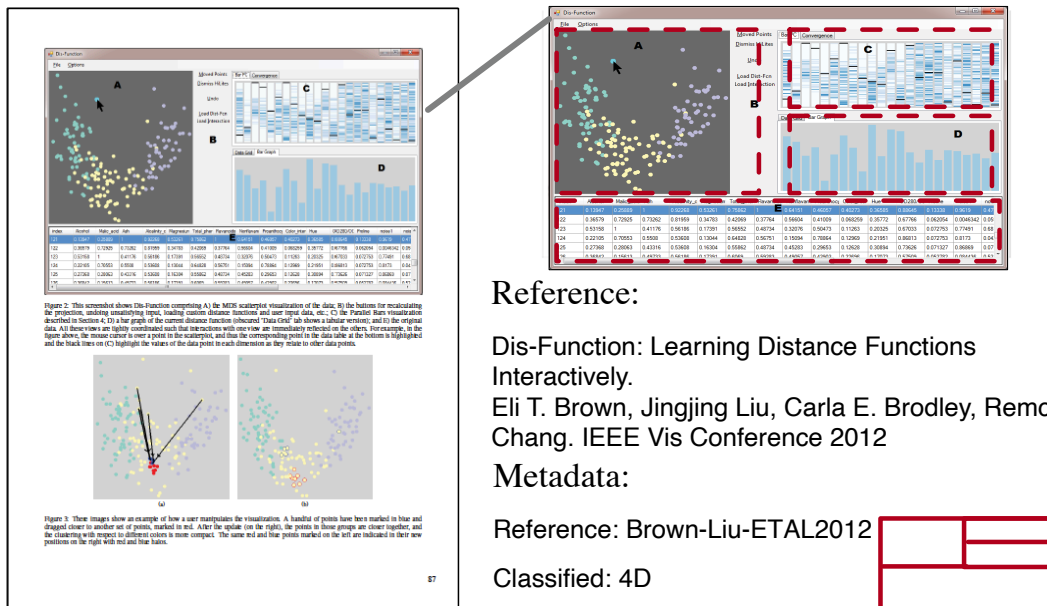


Figure 3.4: Nomenclature and icons for the most frequent layouts.

Five examples are provided in Figure 3.5, Figure 3.6, Figure 3.7, Figure 3.8 and Figure 3.9. These figures provide a summary of how specific images are coded. And, as explained in Table 3.4, for each layout, the authors’ surnames and the publication year are used to reference the layout.



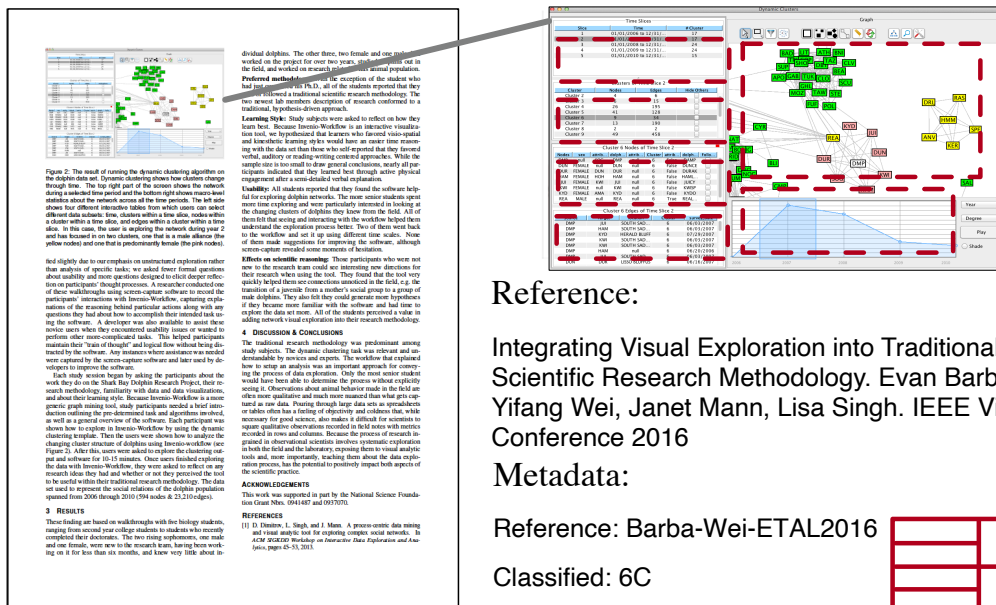


Figure 3.7: Image (from Barba et al. [10]), an example of coding 6-views layout.

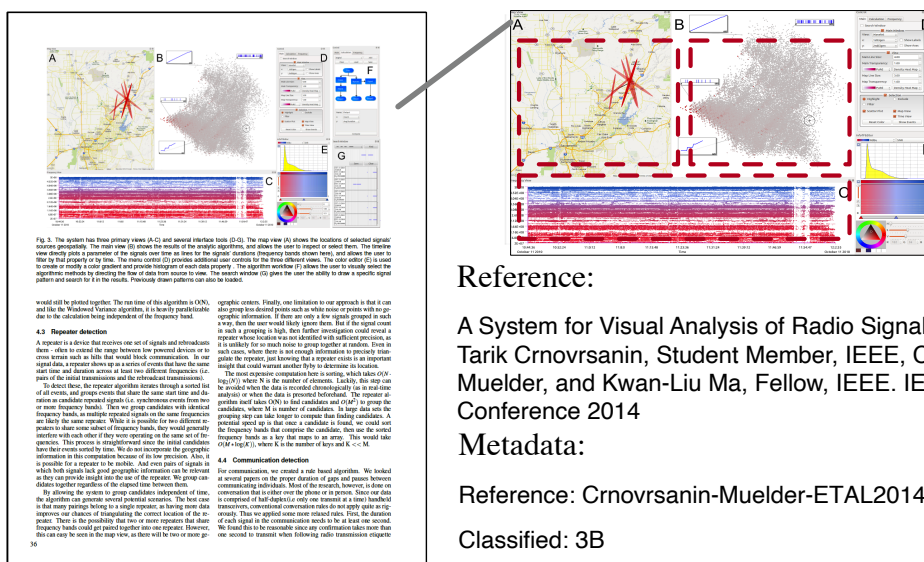


Figure 3.8: Image (from Crnovrsanin et al. [37]), an example of coding 3-views layout.

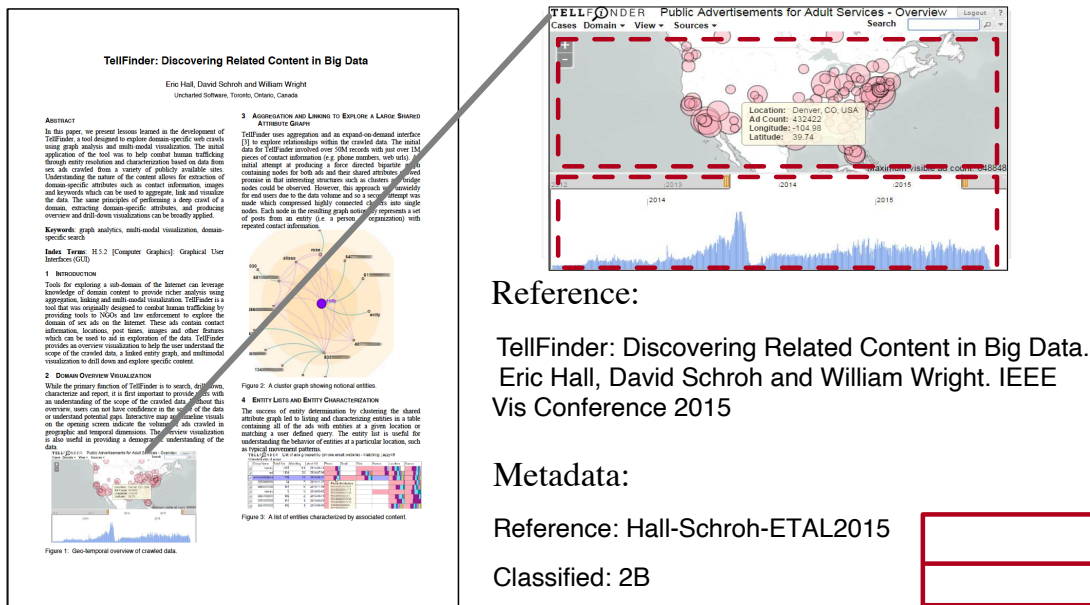


Figure 3.9: Image (from Hall et al. [53]), an example of coding 2-views layout.

Figure 3.10 shows an additional 21 examples of how multiple view visualisations are coded. These images are chosen as a convenient sample of some of the images from the full database of 491 images. The thumbnails come from across the database. They demonstrate both breadth of types across years and breadth of view-quantity within years. Three layouts were chosen from each year, running through from 2012 to 2018. When classifying the images the full resolution of each image was used, which were displayed individually, and on a large screen.

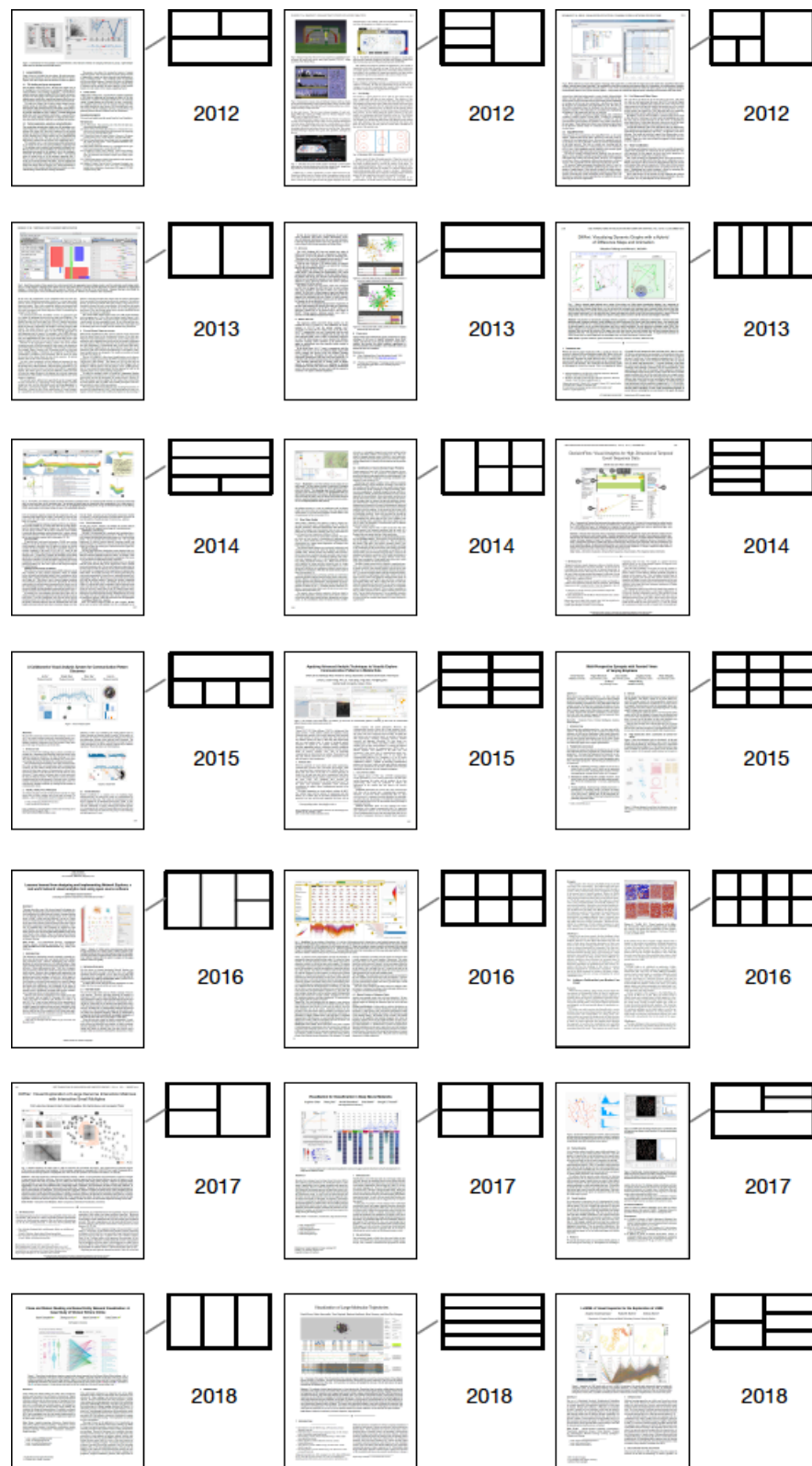


Figure 3.10: Figure showing 21 image thumbnails and their respective layout classification. Sample images from our database of 491 images.

3.5 Coding the visualisation types

Finally, the second part of the third question (q3) is addressed: “What is the strategy to code the visualisation **types**?”

There are potentially many ways to classify the visualisation **types**. For instance, Bertin’s (diagram, network, map, glyph symbol) [15] scheme would provide a broad classification; Roberts [113] describes several data and visual schemes; ManyEyes [143] focus on the data and describe views as 1D, 2D planar, volumetric, temporal, tree, network etc.; while Kerracher and Kennedy [64] classify user actions. But each of these taxonomies do not allow us to name specific view types that we want for our classification. Certainly there are many lists of visualisation chart types on the Internet; names in applications (e.g., D3.js, Excel, Tableau, Qlik and R) or general lists on Websites such as Wikipedia and infovis-wiki.net. But there is little consistency between each of these schemes.

In fact, there is no agreed set of names for visualisation types. For example a *line graph* could be written equivalently as *line plot* or *line chart*. There is little consistency to how these phrases are written in the community, and no agreed form to what they are or how they should be written. Because this issue is not the main focus of this thesis it is explored in a companion paper [109]. Furthermore there is no ontology of visualisation types. Different designers may name the same visualisation differently, tool builders make up new names for ideas that are actually commonplace, and bloggers use names that are convenient for their story (that can be erroneous). For example, *focus+context* could be written as *focus-and-context*; *bar chart* written *bar-chart* or *barchart*; and there are many names for similar concepts (for instance) trellis plots, splom, matrix views and small-multiples all share similar traits [109]. This naming issue is a broader challenge than is possible to tackle in this chapter. Particularly, it is a challenge that should be tackled by the wider visualisation community, and recently some researchers have started to address it already. For example, Isenberg et al. [60] discuss this namespace problem when analysing keywords, and Roberts et al. [109] investigates the wider namespace of multiple views.

Consequently, an inductive (top down) strategy was chosen, to develop codes to answer

the sixth research question (Q6). The strategy was to **look** at the visualisation, **name** it (by saying it, out loud) and **record** its name; only adding new names and new types to the list. The visualisations were named, and the more confusing or challenging ones were discussed with colleagues in the research group. This meant that all bar charts are put together, even if they were horizontal, vertical, stacked, etc. In addition, it was important to make certain that only unique names were used and of consistent formatting (e.g, lowercase, without hyphens). Any uncertain views were further investigated: going back to the paper that displayed the visualisation and searching for the description by the authors. If it was still unclear it was named as “other”.

Visualisation types were grouped together that shared similar traits. For instance, all network diagrams, graphs and associated node link diagrams were placed into one category called *node link diagram*. All stream graphs were placed together in the *area chart* category. Likewise, splom and scatter plot matrices were placed together within the *matrix* category. And, because there was a large quantity of medical images, medical renderings and volume visualisations, they were all placed together in the *rendered image* category.

On the other hand, small multiples, that demonstrate several separate visualisations, were placed in their own category: a *grid*. Which was different category from *matrix* views (that show correlations between sets, such as showing ABCD x ABCD). A *table* category was also added, to distinguish tabular visualisations, which display mostly numeric values. In addition, *scatter plots* and *point charts* appeared to be placed along a time-based axis and use complex symbols, while *bubble charts* differ, because they present many circles and are not usually displayed on an axis.

Lastly, some of the visualisations were impossible to categorise, because it was unclear how they were formed; and they were different to other visual depictions, subsequently they were classified in *other* category.

3.6 Discussion

The goal of this chapter was to present how images were captured. Each image represents a visualisation tool or technique, that clearly originated from applications (whether on a desktop or website), that were created by through a snapshot/screen-grab operation or directly output from the tool. Many sources were considered, including using a general Internet search for visualisation images, video sources such as Vimeo or YouTube, or other online image repositories. However each of these sources change over time, and sometimes the search engines change the search results dependent on geographic location, so it would be difficult for others to confirm our studies, and add more images for future years. In addition, it was required to store a set of images that had been created by visualisation experts, and not the general public. It could be difficult to judge the provenance of images from the Internet. Consequently, it was decided that the IEEE visualization Conference series would be used to create a convenient and reproducible set of images, where the papers have been through peer review.

Certainly the output media (PDF for a digital library or printing in this case) could affect decisions over the layout. In fact, deliberations were made, long and hard, over these issues. A reader can readily imagine a situation where a developer may decide to layout the views of their visualisation tool differently for different media presentations. For instance a wide-format PowerPoint slide has different aspect ratio to the space for a single column in a the IEEE two-column paper format, or even a web page user can change the aspect of the window. However, because there are many different options to position the tool in the paper — from the long-thin teaser image, a page width, or column width — an author would be fastidious over how they present their tool in the most effective way.

In addition, this study considered how to code the images, whether manual or automatic encoding. We considered code the layouts using automatic image-processing algorithms, but the focus was on design layouts and not on algorithm development, and a user would be looking at the tools and making decisions over them, therefore, the decision was taken to use a manual coding process approach.

This work concentrated on view juxtaposition [50] where each views sits alongside each

other, and on the topology of each design layout (e.g., a 2-view system can have one view above another, or left/right of each other). It would have been possible to include other design strategies. For example, Gleicher et al. [50] proposed several strategies to allow comparison, first view juxtaposition, second overlay and third comparative view. Indeed the idea of ‘overlay’ is often used in geographic systems – where a map is the lower level and additional information is displayed on top. Subsequently it could be theoretically possible to classify the multiple-view systems in a similar way. However, while, discerning different facets from a design strategy of juxtaposition is clear, it is far less clear how to visually separate facets in the other strategies. For instance, the idea of ‘overlay’ is often used to display geographic information, where different data variables are laid on top of a lower map level. It would be difficult to visually separate the individual variables, because they all visually merge together. Likewise the same challenges would occur to discern individual facets from an ‘explicit difference’ visualisation.

Additionally, automatic methods of analysis could be possible. It would be possible to develop a deep learning algorithm to decipher the different view facets. While this could be a successful strategy it changes the task from analysing views to writing appropriate artificial intelligence algorithms. The goal of this research is to also develop a set of guidelines that could be used by other researchers. Additionally, it is important to help formalise the debate over the question of “what is a view”. This would not be achievable through the deep-learning process, because the methods of the AI would be hidden, and it is usually difficult to understand how the learnt algorithm actually is making decisions.

Finally, it would be possible to develop a tool to help judge the views. In a collaboration with Xi Chen, Wei Zeng and Yanna Lin and Remco Chang (Tufts University) they have created a tool which enables different users to classify multiple views. The output of which is then used as input to a recommender system. The reader should view this published paper for more information about this work, as it is not reported in this thesis [28].

3.7 Summary

This chapter focused on the preparation process for multiple view quantification, including image selection and coding process. Consequently, this chapter answered three questions: “Q1/ What is the strategy to select multiple view images to evaluate?”, “Q2/ What is the strategy to define and determine a *view* in multiple view visualisation?” and “Q3(RQ1)/ What is the strategies to code multiple view topologies and visualisation types?”.

Images were selected from the IEEE VIS Conferences. Selections came from: published papers, workshops, and posters from 2012 to 2018 (first database). By providing guidelines to select multiple view images for multiple view analysis, it allows other people to create similar sets of images. Two databases were created: the first database has 473 papers that contain the multiple view images and the second database has 491 multiple view images. This chapter also discussed the question “what is a view” and developed guidelines for others to follow. These principles enable researchers to look at the views in the images in terms of separate components. These guidelines can help users to determine and subsequently quantify the views in multiple view systems. Next the work explained how the *codes* were defined. This provided a way to record and classify the images. The coding process started with the observation of each multiple view image and made a judgement on the quantity of the views in that image and the visualisation type in each view, then a representative picture of the layout was sketched, and each labelled it with the paper reference. Then each of the 491 images were systematically considered, which generated 22 sheets of paper (the fourth database).

This chapter, focused on providing an objective structure for the preparation process of quantifying the multiple view visualisations through three sets of guideline that will help another researchers produce similar databases to extended this work, such that researchers can make easy judgements on multiple view visualisations.

The next chapter uses these codes, the created databases and developed processes to quantify the view facets, and develop answers to the remaining research questions (RQ2, RQ3, RQ4, RQ5, RQ6 and RQ7) that related to the quantification process.

Chapter 4

Quantification (data collection), analysis, and design guidelines for multiple view systems

Many developers build multiple view systems, and the method is widely utilised in the world of visualisation. Each visualisation presents data in a unique manner, and often, user interaction across views is coordinated. However, it is not always clear to know how many views a developer should use, what would be the best layout or what is the best visualisation technique to visualise their data.

This chapter will quantify and analyse various multiple view systems published in papers and analyse the data stored in the databases from the preparation and coding chapter (Chapter 3). Additionally, the chapter presents a set of design principles that will assist developers and learners in creating multiple view visualisations. This study focuses on the following research questions and is structured around these challenges:

Q1/ How many views are used in multiple view systems?

Q2/ Do developers prefer symmetrical or non-symmetrical layouts for multiple view systems?

Q3/ What layout arrangements are popular in multiple view systems?

Q4/ What visualisation types are used in multiple view systems?

Q5/ What types of visualisation come together in multiple view systems?

Q6/ What are the guidelines to design multiple view visualisation?

4.1 Introduction

It is not always obvious to a developer how to lay out and place views in their systems. Developers and learners alike should have standards and frameworks to assist them in making good design decisions. Subsequently, it is important to develop visualisation theories and, more particularly, developing recommendations for view layout best practices. However, to accomplish these objectives, researchers must do fundamental research to ascertain existing best practices. While, as shown in Related Work (Chapter 2) much research has been achieved to develop new multiple view tools, and researchers have theorised over aspects such as coordination in multiple views, there has been no quantitative analysis of layout organisation in current multiple view tools. Even theory papers, such as the well known set of guidelines by Baldonado [144] are not based on quantitative data analysis, instead are based on lessons learned from using and developing multiple view systems. Subsequently, by quantitatively evaluating current tools, researchers will have access to data that can underpin theories and help them develop appropriate guidelines.

This chapter presents a quantification and analysis of the multiple view systems stored in our database. This chapter follows from the preparation and coding research that was presented in Chapter 3. The database contains views that have been presented in print. Consequently, the analysis is limited by the data that we have available. The database could be expanded to include animations, commercial tools, websites and so on, but such expansion would make the database larger, would make it more challenging to reproduce the dataset, and could include visualisations that are not useful. By relying on the visualisations created from published works (in a subset of all publications) it is possible to assume that the papers have gone through peer review, and have been carefully crafted by the authors to present their work effectively. Furthermore, the database could be expanded in the future to include other tools. But the database, of multiple view images, stands as a representative sample of multiple view tools as published in leading visualisation conferences and journals, that are respected by the community.

Different aspects could be analysed from the data, but because the wish is to provide guidelines over ‘view layout’ it was decided to focus on the quantity and position of

views in the multiple view systems, the layout arrangements for multiple view systems, the visualisation types and the correlation between the visualisation types in the multiple view systems. There is certainly ‘future work’ to perform. For example, there are many extensions to this work, including: the use of coordination could be investigated; ways to visualise how coordination is incorporated in a multiple-view system; colours and design aspects of the multiple view systems; the differences between multiple view tools and layouts on different output screens/devices.

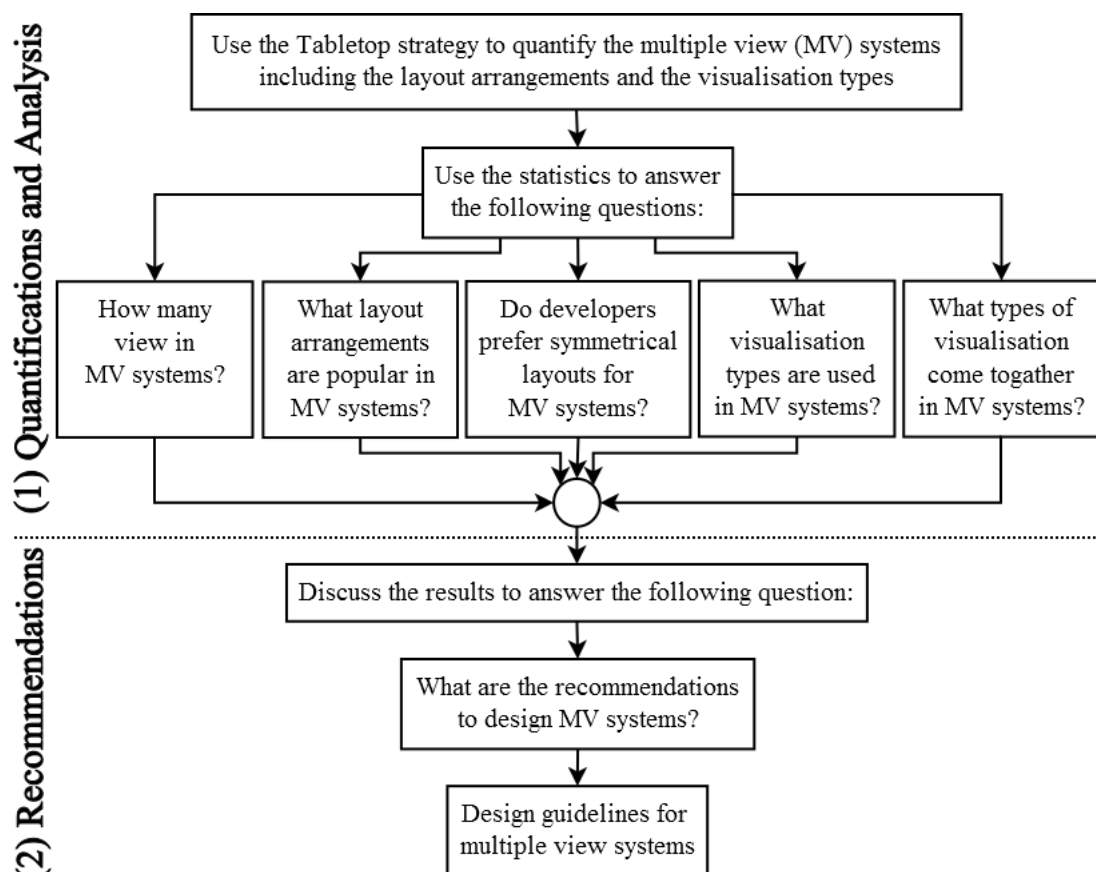


Figure 4.1: The methodology of the quantification and analysis process chapter.

This chapter uses a systematic analysis strategy, whereby each question is focused in turn. The chapter is divided into two broad parts, as shown in Figure 4.1. The first part (quantification and analysis) provides us with a comprehensive understanding of the components and characteristics of the multiple view systems. Moreover, the second part (recommendation) defines a set of steps that can be followed to create effective multiple view systems.

Quantification and analysis.

- Section 4.2: This section explains the methodology that this study followed to count and classify the multiple view layouts and its views.
- Section 4.3: This section starts with quantifying the number of views in all the multiple view layouts, then this section classifies the layouts based on the number of views to find what are the most popular layouts based on views number.
- Section 4.5: This section classifies the layout arrangements to find what are the most popular layout arrangements in multiple view systems.
- Section 4.4: This section investigates the symmetry design of the multiple view layouts to find if developers prefer symmetrical or non-symmetrical layouts, and What is the reason for this preference.
- Section 4.7: This section classifies and counts the visualisation types in the multiple view systems to find what visualisation types that developers have been mostly used to visualise their data.
- Section 4.8: This section investigates the correlations between the visualisation types in the multiple view systems to find the collocational and non-collocational pairs for visualisation types, this investigation will help to know what types of visualisation come together in multiple view systems.

Developing guidelines.

- Section 4.9: This section discusses the overall results and the design decisions that had been made by developers to create multiple view systems. This discussion will help to create a set of design guidelines to help developers create robust multiple view systems.

4.2 The tabletop strategy to quantify multiple view layouts

Developers code data for a variety of reasons. One of the reasons ‘coding’ is chosen is to categorise the data. Coding enables information to be summarised in a way that can be quantified and counted. This will enable the views to be analysed and quantified. It will enable a set of recommendations to be created, that could be used by other researchers in the future. When developers code or re-code data, they organise it to obtain a better knowledge of it. But it is not only a matter of simplifying or summarising the information into a reduced form, but the process of coding itself has the additional benefit of helping the researchers to understand the data in a deeper way. The data could be ‘coded’ using an automatic approach, whereby we use an artificial intelligence algorithm to learn typical behaviours. However manual coding was chosen to perform the analysis. Manual coding enables each view to be evaluated, judged and discussed. Each view was considered, coded and discussed. While much time and effort was spent on this process, the actual process gave insights into the data that would have not been achieved through automatic methods. If programming languages were used to apply an Artificial Intelligence (or deep learning) algorithms, the AI would learn. It would not have helped develop a deep understanding of the information. In addition, to code the views it requires a deep understanding of the processes, and this would have been required in order to write the AI algorithm. So the act of coding, labelling it, and even discussing the labels as a research team, benefited and deepened our (as researchers) understanding of the underlying data. The coding process therefore is just as important as the codes that are produced.

Along with the manual coding exercise, physical tokens were created. These ‘tiles’ were formed by sketching the layout on a tile, cutting it out, and positioning it on a table. This helped to create a deep understanding of the data and it meant that different categories and orderings could be discussed. This physicalisation was an important step to help develop and understand the data better. Placing the tiles on a tabletop, positioning tiles nearby other similar tiles helped to develop and classify the layouts. This process also helped to confirm the quantity of multiple view layouts in our database, and to count

the multiple view systems and its components. A photograph of the table-top is shown in Figure 4.2.

This is a tangible method. Each sketch is cut into tiles (from the sheets totalling 17 sheets of paper), placed and organised each of the multiple-view topologies as separate sketched tiles on a tabletop. In fact, there were so many tiles that two tables were required to hold all the tiles. Each tile was physically moved and located in place. Placing similar layouts together. This helped to develop a better understand the frequency of each layout, and also gives a physical area chart of the quantities in each strategy and shows the relative quantity of each layout. From this tabletop collection of tiles it was possible to record the quantities in a spreadsheet for further analysis. The tabletop display provides a visual quantitative summary of view layouts.

The tiles were arranged and classified the layouts tiles on the table by arranging them into groups. Similar tiles were grouped together: first by quantity of views and then by their topological structure. This strategy provides a visual way to to investigate the structures and layouts.



Figure 4.2: Using a tabletop strategy, I cut these sheets into individual tiles such that I could discuss them and move them around on a tabletop. I layout each of the multiple-view topologies as separate sketched tiles. The simple abstract drawings gave us a easy way to compare the structures without being distracted by the actual view design or content. This provides a visual summary of the range of layouts.

4.3 The Quantification and the analysis of views number in multiple view layouts

This section focuses on the question: **q1/ How many views are used in multiple view systems?**.

The tiles were grouped together on the tabletop based on the number of views. Through discussion of the tiles, and by considering the different quantities of the codes, they were arranged into 11 groups. For the process of placing the tiles on the table, and for the higher-view counts, a final category named “lots” was used. This extra category includes all layouts with more than 10 views. The table-top presentation of the layouts is shown in Figure 4.2. This photograph provides a useful visual overview of the quantity of layouts. It shows that three and four view-layouts are most frequent. This strategy helped with tallying the quantities of multiple view visualisations, enabled us to discuss different cases and categorise them appropriately. Then, the view layouts/tiles were counted, and tallied. It is then easy to create a histogram that presents the the frequency of each layout structure, as shown in Figure 4.3.

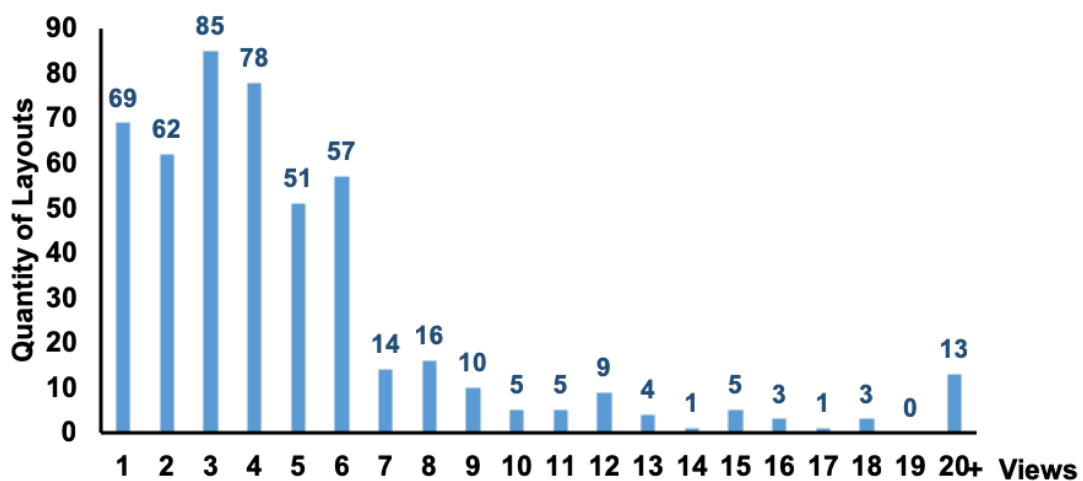


Figure 4.3: Histogram showing the frequency distribution of the views. From 491 multiple view systems, in our study we find that a 3-view system is most frequent.

Initially, when the tiles were transcribed to numerical values, and placed in an excel spreadsheet, views up to 20 were counted. The decision was taken, that the space allocated to individual views – when the view quantity exceeds 20 views – is very small. Therefore the data was binned into a single category of 20+, as shown in Table 4.1. And

the data reorganised and recorded the quantity of layouts based on the published year besides to the views number, and tallied each of the different view topologies up to 20 views.

From this data, and careful analysis of the layout structures of the 491 visualisations and grouping, the quantity of the visualisation views can be calculated for all layouts.

From our data gathering, the histogram Figure 4.3 and Table 4.1, it is now possible to answer q1: “how many views do people use”. Because there are different ways to interpret the statistics, the question is broken into several sub-questions.

What is the most common layout? To answer this question the view quantities are ranked (see Table 4.1). The most common layout is a 3-view system. Four-view systems are next, followed by one-view systems and dual-views. Six view systems are slightly more frequent than five view systems. There is a clear statistical separation (in the histogram) between 6-view and 7-view (and more) systems. In fact 84.68% of the systems are 6-view or less. These results are important. They suggest that the majority of developers use 6-views or less, and most of them choose a 3-view system. They also suggest that (in general) fewer views are used more often, which supports the rule of parsimony by Baldonado et al. [144]. In addition, Figure 4.3 shows a breakdown of the quantities. 44 of the layouts had 11 or more views, and from looking at these designs it is possible to see that each these visualisation are small multiple displays.

What is the average quantity of views used? The naive arithmetic mean calculates to 4.9. However, like most averages, this is misleading and this number hides much detail. We have a positive skew in the distribution of the view count (skew is 1.049), and it is clear from Figure 4.3 that there is a very long tail. Such a positive skew is understandable; when the views are counted it is impossible to get a value less than a one view system, and it is far less likely to see systems with huge quantities of views (it is just impractical to have a system with hundreds of views). This situation can be demonstrated by modelling a Normal distribution from 1 to 20, with an average of 3 (as per the most frequent occurrence), and comparing the observations with this model. In fact a statistically similar result to the coded observations is achieved: a Pearson correlation is calculated as 0.960 with a Ttest $p(0.885)$.

In conclusion, the quantification shows that designers do use many views, and considering guidance to use multiple views parsimoniously, this is an interesting result. In addition, per year data was analysed, but it was not possible to make any conclusions from the six year period. It will certainly be interesting to take a historic look at the visualisation systems, but this was out of the scope of this current study and will need more years worth of data to make sense of the information, therefore the per-year question is left to future work.

Table 4.1: Results of tallying the views. Views were also tallied per years, with the frequency and the percentage frequency. Applications with 20+ views are aggregated together (and treat them with a system of 20 views, for calculations).

Views	2012	2013	2014	2015	2016	2017	2018	Freq.	% Freq.	Rnk
V_i	f_i	f_i	f_i	f_i	f_i	f_i	f_i	f	$\%f$	
1	10	7	12	12	9	8	11	69	14.05	3
2	10	6	7	9	5	11	14	62	12.62	4
3	8	11	6	10	14	16	20	85	17.31	1
4	12	8	11	13	10	10	14	78	15.88	2
5	7	4	6	9	4	10	11	51	10.39	6
6	8	7	3	9	8	8	14	57	11.60	5
7	1	1	1	3	3	3	2	14	2.85	8
8	3	1	3	3	3	3	0	16	3.25	7
9	3	1	0	1	1	4	0	10	2.03	10
10	1	0	2	0	0	1	1	5	1.01	12
11	0	1	0	2	0	2	0	5	1.01	12
12	0	1	2	0	3	1	2	9	1.83	11
13	0	1	0	0	1	1	1	4	0.81	15
14	0	0	0	0	1	0	0	1	0.20	18
15	0	0	1	0	2	0	2	5	1.01	12
16	0	1	0	0	0	1	1	3	0.61	16
17	0	0	0	0	1	0	0	1	0.20	18
18	0	0	2	0	1	0	0	3	0.61	16
19	0	0	0	0	0	0	0	0	0.00	20
20+	0	0	4	2	1	5	1	13	2.64	9
Total	63	50	60	73	67	84	94	491	100	
$\sum_{i=1}^{20} V_i f_i$	272	222	342	327	371	470	421	2425		
Avg.	4.31	4.44	5.7	4.47	5.53	5.59	4.47	4.90		

4.4 The Quantification and the analysis of the symmetrical multiple view layouts

During the physical tiling process, some of the topologies seemed more symmetrical than others. This is a sensible conclusion. One of the design principles, as explained by

the Gestalt scientists, is that humans prefer symmetrical designs [147]. But is this the case, and how is it possible to evaluate this idea? It could be possible to ask researchers specific questions such as ‘do designers prefer symmetrical multiple view layouts when they create a multiple view system?’ or ask questions of designers, web developers or visualisation tool developers. But such a set of questions would probably be repeating the work of the Gestalt scientists. Therefore, it was decided to evaluate the views in the database, and use a new sub-code for this purpose. Each view was classified by the new ‘symmetrical code’. For example, they were coded as being symmetrical, if the view layouts were more balanced, such as having two identical sides vertically or horizontally. This is called mirror symmetry [57]. For example, both codes 2A, which is a horizontal dual-view system, and code 2B, which is a vertical dual-view system are symmetrical (see, Table 4.2). However, views such as 4D or 4F are not symmetrical.

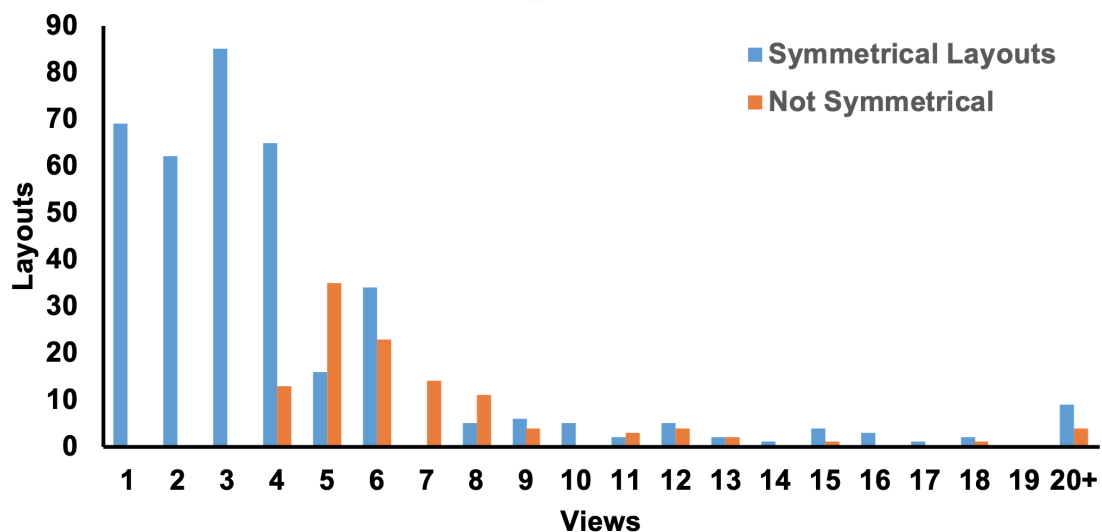






Figure 4.4: Histogram of symmetrical versus non-symmetrical views.

Symmetry occurs when the view elements are arranged in the same way on both sides of an axis. Perfect symmetry is when elements are mirrored over the axis and exactly the same on both sides. Symmetrical balance encourages an equal weight of both sides. The hypothesis is that more designs are symmetrical than not. In other words, that visualisation tool developers may follow practices of *balance* and *visual flow* [43] that are found in human computer interaction (HCI), to help them layout their views. For instance, when a user looks at an image their eyes move around and get attracted to different parts of the display. Their eyes are pulled, by the visual imagery and layout of views, to flow in a particular direction. The positioning of the graphics and interface encourages the visual flow of their gaze towards a particular *visual direction* [8].

In addition, humans often prefer symmetrical pictures, and therefore the developers may prefer symmetrical views. For instance layout 2A  contains a vertical split and therefore it is vertically balanced; 2B  is horizontally balanced, 4B  is balanced vertically, whereas 4F is  is not symmetrical. The results of the symmetrical analysing of multiple view layouts are shown in Figure 4.4, and to explore this hypothesis we plot the quantity of symmetrical views (whether vertical or horizontal symmetry) against the non symmetrical. When the results are observed, the conclusion was formed that there may be a trend to utilise more symmetrical layout strategies. But the results are not conclusive, because there is a natural tendency for layouts with more views to be less symmetrical. And it is difficult to tease apart these two observations.





Symmetry in multiple view layout can be a very suitable and sensible design principle. It can create or maintain balance, calmness, and stability. It can communicate integrity, professionalism, and solidarity. Asymmetry, on the other hand, can develop strong points of interest, uniqueness, and character.

4.5 The quantification and the analysis of the layouts arrangements






This section focuses on the results of the topological structure, and by using the tabletop layout of the tiles, it is now possible to examine the view layouts. Initially when counting the layout arrangements, the tiles were exchanged into the nomenclature, as explained in Section 3.4 and shown in Figure 3.4. This nomenclature provided a way to record the quantification of the layout arrangements.

The topologies were organised and counted to find the common layout structures, and the results for the relative quantity of each layout arrangement are shown in Table 4.2, the results ordered by the overall usage percentage ($\%f$) of the layout arrangements, from top to bottom. The results provide even more fine-grained details, and to calculate these results we performed some vertical aggregation.

When considering the different styles, there are many different layouts (many are individual). Therefore, to aggregate the views together, to summarise the main view layouts, the topology of each view layout was considered. If the views were symmetrical

(left to right or top to bottom) then they were counted as the same layout type. So, for instance, if the topology was the same on the left (such as ) and to the right (such as ) the same label was chosen; in this case 3A. Consequently, 3-view layouts with one long horizontal section and two shorter ones () were counted as a same design whether the long-thin section was on the top or bottom. For a 4-view layout with two parts horizontal and two parts vertical split () were considered the same design as a view with the opposite arrangement.

While this method does not consider if the views are more left biased or right, it is a pragmatic decision that allows the names to be quickly tallied. But even with this aggregation scheme, there were many unique configurations, especially when the quantity of views increases above four.



















































Analysing the layouts in fine detail is interesting but challenging. There are many layouts, and as the quantity of views increase so does the number of arrangements. From Table 4.2 it is possible to easily observe which are the higher ranking layout arrangements such as 2A , 3A , 3B . It is interesting that the popular layouts are not necessarily those with fewer views, for instance layout 6A  (is a small gridded layout) and layout 5A  (is a layout with little structure) are within the top ten layout arrangements. However, the views in layouts 6A and 5A are relatively few compared to other layouts that contain more than seven views.

In addition, this study recognises that there are many familiar structures, and when the views quantities become large, there are many more topological arrangements. There are 44 layouts which show side-by-side views, and a further 18 that a two-way split (top to bottom). Moreover, four layout arrangements have 3-views, and nine layout strategies with 4-views. This research also perceives a two-thirds design strategy being prominent, and more than half of the layouts have a significant left/right division somewhere in their strategy. The others follow a 3-way split. Moreover, this investigation notes that 16.29% of the layout structures are unique, where $f = 1$, and there is less similarity with higher view counts. Table 4.2 shows the trend, that as the view count increases there is less agreement in design strategies.

Furthermore, this study notices some interesting trends from Table 4.2. Most designs

with a long and thin division of space use this space for a timeline view. To confirm this situation, we looked at all of these instances in the 2016 data. There are 9 such designs and 8 are timeline views. For example, take for instance the paper by Park et al. [97] who name their top design as a timeline filtering view. Furthermore there are many familiar structures. 35 views show side-by-side views, and a further 13 have a two-way split (top to bottom). Four layouts have 3-views, and nine layout strategies with 4-views. Two-thirds design strategy are prominent, and more than half of the views have a significant left/right division somewhere in their strategy. The others follow a 3-way split.

Table 4.2: Results of tallying the specific layouts, per years. Where f is frequency, $\%Vf$ is percentage frequency of that View type, and $\%f$ is percentage overall. The complete dataset is shown in this table. It demonstrates a long tail of many cases with few instances, in fact there are 81 layouts with $f < 1$). The rows are ordered by their overall rank.

Layout	Y12	Y13	Y14	Y15	Y16	Y17	Y18	f	$\%Vf$	$\%f$		Rnk
1A 	10	7	12	12	9	8	11	69	100.00	14.05		1
2A 	9	4	3	5	4	10	9	44	70.96	8.96		2
3A 	1	3	3	3	8	7	9	34	40.00	6.92		3
3B 	6	3	2	4	4	4	6	29	34.11	5.90		4
4A 	4	3	1	5	4	2	5	24	30.76	4.88		5
6A 	1	2	1	4	2	3	7	20	35.08	4.07		6
2B 	1	2	4	4	1	1	5	18	29.03	3.66		7
3C 	0	2	1	2	1	5	3	14	16.47	2.85		8
5A 	2	0	1	1	3	2	2	11	21.56	2.24		9
4B 	2	0	3	1	3	1	0	10	12.82	2.03		10
4C 	2	0	2	0	1	1	4	10	12.82	2.03		10
3D 	1	3	0	1	1	0	2	8	9.41	1.62		11
5B 	1	1	2	3	0	1	0	8	15.68	1.62		11
6B 	1	1	1	1	0	2	2	8	14.03	1.62		11
4D 	2	2	0	0	0	3	0	7	8.97	1.42		12
9A 	2	1	0	1	0	2	0	7	70.00	1.42		12
4E 	0	0	2	2	1	0	1	6	7.69	1.22		13
4F 	2	1	1	0	0	1	1	6	7.69	1.22		13
5C 	0	0	0	1	0	2	3	6	11.76	1.22		13
6C 	0	1	1	0	2	0	2	6	10.52	1.22		13
4G 	0	2	0	1	1	1	0	5	6.41	1.01		14
4H 	0	0	2	2	0	0	1	5	6.41	1.01		14
4I 	0	0	0	2	0	1	2	5	6.41	1.01		14
5D 	1	1	1	1	0	0	1	5	9.80	1.01		14
8A 	0	0	2	0	1	2	0	5	31.25	1.01		14

4.6 Understanding tasks and domain

This section looks at the purpose of the visualisations. What tasks the developer may ask the user to perform, and what domains they are in. At the end of the chapter, a domain analysis is performed, and a discussion made over which domains use multiple view layouts.

Users create multiple view visualisations for a purpose. When creating multiple view visualisations, the choice of a layout arrangement by the developer is based on the tasks required to be performed on the visualisation. Maybe the developer wants the user to *search* and *locate* a specific value in the dataset (represented by a particular graphical symbol). The developer may want the user to be able to *identify* a specific data value. It is important to understand why the developer is creating the tool: they are creating a tool to address a particular need in a subject domain, and to perform a specific task. For example, a developer working with oceanographic data would like users to distinguish between tidal flows; when water-flow up an estuary, in comparison to how the water flows down the estuary when the tide changes direction.

Many researchers have investigated visualisation tasks, especially tasks performed in a multiple view system. For example, Shneiderman in 1996 explains the importance of a task analysis in his seminal work on a “The eyes have it: a task by data type taxonomy for information visualizations”[132]. To aid in the selection of layout arrangement, Wehrend and Lewis [151] categorised the operations that a user may need to do to analyse data as follows:

- **Locate:** the user is aware of a dataset entry and indicates it with a point or a description.
- **Identify:** like locate, but the user identifies the dataset item without prior knowledge.
- **Distinguish:** separate visual elements should be used to represent distinct objects.
- **Categorise:** objects may be distinct due to their affiliation with various categories, each of which should be defined by the user.

- **Cluster:** the system may detect that categories and items associated with them are shown connected or clustered together.
- **Distribution:** the user defines categories, and the items that fall within those categories are dispersed accordingly.
- **Rank:** the user is prompted to mark the arrangement of the presented items.
- **Compare:** the user is prompted to make a comparison of items based on their characteristics.
- **Compare inside and between relations:** the user is prompted to make comparisons between comparable entities or different groups of items.
- **Associate:** the user is prompted to create relationships between presented items.
- **Correlate:** the user may notice that items have common characteristics.

Shneiderman [132] developed a taxonomy of information visualisation tasks based on seven data categories (1-dimensional, 2-dimensional, 3-dimensional, temporal, multidimensional, tree, and network) and seven user tasks. He defined the following tasks: *overview*, *zoom*, *filter*, *details-on-demand*, *relate*, *history*, and *extract*, all of which are based on the visual information seeking mantra.

Soon after the publication of Shneiderman's taxonomy, Zhou and Feiner [164] provided another task classification. They distinguished presenting intentions (the objectives of a user while interacting with a visual representation) from low-level visual techniques (the precise action done on a particular item shown in the display) through an intermediary level called visual tasks. Visual tasks may be thought of as abstract visual methods since they specify a desired visual effect in the representation, while a visual technique is a method for achieving that intended effect, whether by the user or the system.

Zhou and Feiner define visual tasks in terms of their visual accomplishments and implications. Visual accomplishments describe the presenting intentions that a visual task is intended to serve, while visual implications describe the visual methods that

might be used to complete the visual task. Visual accomplishments may be classified into two categories: *inform* and *enable*. Inform tasks are classified further as Elaborate and Summarise, while enable tasks are categorised as *explore* and *compute*.

Morse et al. [88] created a method for translating this visual taxonomy to actual knowledge retrieval tasks represented by 50 questions. They tested the role of visualisations in the area using subsets of these questions and simple visual prototypes. They attempted to thoroughly assess visualisation skills by creating tests based on this classification.

Visual tasks are substantially different from perceptual operators [88, 58], which represent the perceptual activities that a user does in a visual environment. Perceptual operators highlight what a user must do (search, decide, verify, compare, lookup, add, subtract). At the same time, visual tasks define the support that a visual representation must offer for user task completion.

Byrne et al. [25] developed a taxonomy of online user tasks based on studies of web application users' most frequently performed activities. This study defines patterns for user tasks in terms of six sub-tasks: using information, locating on the page, going to the page, providing information, configuring the browser, and reacting to the environment. These patterns can be used to create a comprehensive vocabulary for user activity in this application domain.

Amar and Stasko [5] recently discussed the concept of the analytic gap, which refers to the difficulties that visualisation systems encounter when facilitating high-level analytical tasks such as domain learning and decision making under uncertainty, which is typically not covered by existing research on the design and evaluation of information visualisation systems.

The authors assert that, although Wehrend and Lewis' and Zhou and Feiner's low-level tasks are necessary, they do not provide a consistent foundation for filling analytic gaps. Thus, they suggested a new taxonomy [5] with higher-level activities that may assist designers and assessors of visualisation systems. The limitations of the current visualisation technologies were classified into two broad categories: the Rationale Gap and the Worldview Gap. The first is described as the gap between experiencing a connection and being able to explain one's trust in it, as well as the relationship's utility.

Indeed, users must be able to connect data sets to the domains within which choices are made. The second is described as the discrepancy between what is presented and what is required to arrive at a clear representational conclusion for decision-making. Indeed, users must be able to devise a strategy for exploring a visualisation as well as for developing, acquiring, and sharing knowledge or information about critical domain characteristics included inside a data collection.

Then, for each gap, the following three high-level activities should be supported by visualisation systems (although overlap is possible): a) activities that need exposure to ambiguity, concretisation of connections, and formulation of cause and effect; and b) tasks that require a worldview: domain parameter determination, multivariate explanation, and confirmation of hypotheses.

Amar et al. [4] recently presented a taxonomy of ten low-level activities based on 196 analytical questions discovered by students while studying data using commercial visualisation tools.

Effectively evaluating information visualisation systems requires a knowledge and depiction of the activities that a user does when examining data.

Norman's theory of action [92] states that the connections between tasks and objectives are apparent. During system interaction, the user's behaviour follows a seven-stage cycle: the user has goals; formulates intents; verifies possible actions and chooses the most appropriate one based on their intentions; executes the selected action; perceives, interprets, and evaluates system results until the task is completed. Occasionally, a user objective may be assigned to a single task, but it will often need the coordination of several tasks. Clearly, users' requirements and objectives should be considered throughout the design and development process. Evaluation is the process of validating: a) how well a system covers users' objectives efficiently; and b) how effectively, efficiently, safely, and satisfactorily users' activities utilising (tasks of a) system match the precise user goals [61]. However, few writers specifically investigate the collection of user activities for the aim of assessment. We are particularly interested in addressing the usage of task comprehension and representation (for example, the well-known HCI task model) for assessment reasons.

Winckler et al. [154] explored how to model these tasks using a formal approach and its associated environment to take advantage of the possibility of automatically creating many scenarios that include all of the achievements and consequences associated with each visual task.

In an ideal world, evaluations of visualisation methods would include the following capabilities:

- Identify the user's objectives and determine if the user can accomplish them via the usage of an application that employs an information visualisation method.
- Determine whether interaction mechanisms made accessible to the user by visualisation methods are necessary for the user job to be accomplished.
- Recognise the graphical rendering functions used by visualisation methods to display data.
- Establish a connection between user objectives, interaction methods, and graphical display.

By classifying multiple view visualisations based on an integrated analysis of the above categorisations and the characteristics of datasets (such as the domain of the data that are used in the multiple view systems), this can help to find out which layout arrangement would lead to a better solution for an application problem.

The first part of the analysis looks at the relationship between the layout arrangements and the domain of the data used in the visualisation system. This research can be a starting point for the collocation of the visualisation task and the layout arrangements, which will help the developers find the best layout for their data.

Table 4.3: Overall data visualization domains in all layouts, the top-ranked data visualization domains are mostly found in the top-ranked layouts (as explained in Section 4.3 and shown in Table 4.1 and the histogram Figure 4.3).

Data visualisation domain	Σ %			In layouts with quantity of views									
				1	2	3	4	5	6	7	8	9	>9
Visualisation	33	35.11	<div><div></div></div>	4	6	8	4	3	4	1	0	0	3
Machine learning	9	9.57	<div><div></div></div>	1	0	4	1	2	1	0	0	0	0
Information systems applications	7	7.45	<div><div></div></div>	1	2	1	0	2	1	0	0	0	0
Physical sciences and engineering	7	7.45	<div><div></div></div>	0	0	4	1	0	1	0	0	0	1
Enterprise computing	5	5.32	<div><div></div></div>	1	0	0	2	2	0	0	0	0	0
Life and medical sciences	5	5.32	<div><div></div></div>	0	2	0	1	0	1	0	0	0	1
Computer graphics	3	3.19	<div><div></div></div>	1	0	1	1	0	0	0	0	0	0
Document management and text processing	3	3.19	<div><div></div></div>	0	0	0	0	0	2	0	0	0	1
Law, social and behavioral sciences	3	3.19	<div><div></div></div>	0	0	0	0	0	2	1	0	0	0
Artificial intelligence	2	2.13	<div><div></div></div>	0	0	0	0	0	2	0	0	0	0
Computing methodologies	2	2.13	<div><div></div></div>	0	1	0	0	1	0	0	0	0	0
Education	2	2.13	<div><div></div></div>	1	0	1	0	0	0	0	0	0	0
Modeling and simulation	2	2.13	<div><div></div></div>	1	0	0	1	0	0	0	0	0	0
Software and application security	2	2.13	<div><div></div></div>	0	0	0	1	0	0	0	0	0	1
Systems Security	2	2.13	<div><div></div></div>	0	1	0	0	1	0	0	0	0	0
Data management systems	1	1.06	<div><div></div></div>	0	1	0	0	0	0	0	0	0	0
Human computer interaction (HCI)	1	1.06	<div><div></div></div>	0	1	0	0	0	0	0	0	0	0
Network performance evaluation	1	1.06	<div><div></div></div>	0	0	1	0	0	0	0	0	0	0
Network security	1	1.06	<div><div></div></div>	0	0	0	0	0	0	0	0	0	1
Probability and statistics	1	1.06	<div><div></div></div>	1	0	0	0	0	0	0	0	0	0
Summarization	1	1.06	<div><div></div></div>	0	0	0	1	0	0	0	0	0	0
Visualization systems and tools	1	1.06	<div><div></div></div>	0	0	0	1	0	0	0	0	0	0

The CSS (Computing Classification System) is used. This system provides categories for the computing field from the ACM Digital Library. The ACM provides a tool within the visual display format for applying CCS categories to upcoming papers. This tool helped us classified and quantified the layout arrangements based on their data domain. We analysed the quantification to discover the relations between the layout arrangements and the data domain.

Table 4.3 (top), shows the relations between the number of views in layouts and the domain of the data in the multiple view visualisations located in the publications of IEEE VIS 2018 Conference as a sample. Where, the most data domains are found in layouts with six views or less.

Table 4.4 (bottom), shows the relations between the layout arrangements and the domain

of the data in the multiple view visualisations located in the publications of IEEE VIS 2018 Conference as a sample.

Furthermore, as seen in Table 4.4, several trends can be observed. Most of the multiple view systems are related to the “human-centred computing”, “computing methodologies” and “applied computing” fields. Furthermore, most multiple view systems are associated with “visualisation”, “machine learning”, “information system application”, “physical sciences and engineering”, “enterprise computing”, and “life and medical sciences” topics.

In addition, most of the domain topics are found in multiple view systems with less view. This find supports our results in Section 4.3 that developers of multiple view systems prefer to use fewer views in their designs.

However, the developers did not utilise every CSS category. For instance, “semantics and reasoning”, “randomness, geometry and discrete structures”, “Printed circuit boards” and “symbolic and algebraic manipulation” are not used. A possible reason is that the research on these topics does not require to use of visualisation, or that these are possible topics for future visualisation application.

This quantification can be a starting point for finding the relation between the layout arrangements, the data domain, and the visualisation tasks; by increasing the samples of multiple view systems and including more papers to be evaluated, we leave this research for future work.

Table 4.4: The quantification of data visualization domains in the top-ranked layout arrangements (the order of layout arrangements is taken from the top layout arrangements that listed in Table 4.2 at Section 4.5). The top six rows are highlighted in green because there are more data domains within these layouts.

Data visualisation domain	Layout arrangements																								
	1A	2A	3A	3B	4A	6A	2B	3C	5A	4B	4C	3D	5B	6B	4D	9A	4E	4F	5C	6C	4G	4H	4I	5D	8A
Visualisation	4	3	2	5	2	2	3	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	0
Machine learning	1		4	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Information systems applications	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
Physical sciences and engineering	0	0	0	1	0	0	0	2	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0
Enterprise computing	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
Life and medical sciences	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
Computer graphics	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Artificial intelligence	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Computing methodologies	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Document management and text processing	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0
Education	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Modeling and simulation	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Systems Security	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
Data management systems	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Human computer interaction (HCI)	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Law, social and behavioral sciences	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
Network performance evaluation	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Probability and statistics	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Software and application security	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Summarization	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Visualization systems and tools	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

4.7 Quantification and analysis of visualisation types

This section focused on the analysis of the results of counting the visualisation types in all the layouts, in addition to quantifying the visualisation types in the top-ranked layout arrangements.

Many different types of visualisations have been created. Bar charts, line graphs, scatterplots are all popular visualisation types. But how can we classify views by types? What is the best way to decide on a view type? While there are no formal, or widely agreed lists of visualisation types, there are different ways to collate such lists.

One way to create a list of visualisation types could be by using an Internet search. Results from the search include returns such as aggregated lists of “top five visualisations” or “10 visualisation types made simple” and so on. Most of these lists have been created by keen bloggers, and seem to be of varied quality. However they all include some popular types: bar chart, line graph, scatterplot, pie chart, area chart and bubble chart. Anecdotally this already provides a casual researcher with some hints at the most popular visualisation types. But what types of visualisations do researchers use, when they create multiple view visualisation tools? We could use data from these Internet searches to create our own ‘informal’ list of types, or look to lists in academic publications.

Another place to look would be the research literature. Several researchers have looked at creating taxonomies for visualisation. However many of these taxonomies concern with other visualisation issues, for example Shneiderman’s task by data-type taxonomy [132] includes general visualisation concepts from linear visualisations, planar, volumetric temporal and so on; which has been used to classify visualisation types (e.g., for the LibGuide collection ¹). Ed Chi’s taxonomy of visualisation techniques using the data-state reference model [29] helped to develop the dataflow visualisation model. Several researchers are highly influenced by Bertin’s work and categorisation. Bertin [14] provides information about different categories of visualisation, and created a visual naming scheme to realise this categorisation, and their categorisations and taxonomies are based on Bertin’s ideas. For example, Tory and Möller’s “Rethinking visualisation:

¹LibGuide, http://guides.library.duke.edu/datavis/vis_types

a high level taxonomy” incorporate Diagrams, Networks, Maps and Symbols and split the visual forms into linear, planar, volumetric, etc. Brodlie [21] extends Bertin’s ideas in his work titled “Visualization Notations, Models and Taxonomies” and Roberts [114] classifies both data-types and visual types. While these researchers do classify the visualisation types, they are not broad enough for our purpose. Other researchers have classified specific areas of visualisations, such as Kuncher and Kerren’s taxonomy of “text visualisation techniques” [68] or the periodic table of visualisation methods, which is located under visual-literacy.org².

There are challenges in looking at lists from researchers. We researched words associated with multiple views. That research, as explained in Section 2.2, helped to inform the research in this PhD. For example:

- Researchers are regularly inventing new visualisation types, creating new and original names for them, and it could be difficult to keep an up-to-date list. Some of these visualisations do then become popular. For example, Treemap was developed by Ben Shneiderman in 1991 as a technical report, and then published in ACM Transactions on Graphics [135]. Treemaps are now common place in many visualisation tools, and thanks to work by other researchers (such as cushion treemaps and SequioView [142]) Treemaps are used by many Microsoft Windows machines to view the hard drive. While the Treemaps story is inspiring, there are many visualisation types that are not used so widely. We are interested to investigate popular visualisation types, and so any new visualisation type would have far fewer instances of use.
- Another challenge, which we learnt from the “Multiple Views: different meanings and collocated words” work, was the names given to a visualisation type, are not uniquely agreed by researchers or tool developers. For example, some researchers say “scatter plot” others use it as one word “scatterplot”. Some people write “SPLOM chart” and others write “scatterplot matrices”.

Another way we could get the list of visualisation types, is to look to the visualisation tools. For instance, tools such as ManyEyes [143], the IBM visualisation aggregation

²https://www.visual-literacy.org/periodic_table/periodic_table.html

tool that allowed the public to upload different visualisations, held a list of visualisation names. Polaris [139], Voyager [158], Keshif [160] all include many visualisation types, even packages such as ExcelTM include many visualisation types, such as Maps, Bar chart, line graph, pie chart, sparklines and radar, along with some less familiar types such as funnel and waterfall charts. While it can be useful to look at all these lists, we leave it to further research to develop a inclusive list of visualisation types. In addition, many of these types would probably not be found in our database, because they are infrequently used.

Consequently, it was decided to take a bottom up approach, and develop the ‘codes’ for the visualisation types, based on what we observe in the data.

The visualisation types were named, and the more confusing or challenging ones were discussed with colleagues in the research group. Visualisation types that shared similar traits were grouped together. For instance, all network diagrams, graphs and associated node link diagrams were placed into one category called *node link diagram*. All stream graphs were placed together in the *area chart* category. Likewise, splom and scatter plot matrices were put together with the *matrix* category. And, a large quantity of medical images, medical renderings and volume visualisations, were placed together in the *rendered image* category.




























Table 4.5, Table 4.6 and Table 4.7 show the visualisation types, calculated per year (2012 to 2018), per layouts (based on number of views in the layouts) and per layout arrangements, respectively, where each of the visualisation types in every view of the multiple view layouts were counted. Consequently, if a two-by-two grid visualisation (4A ) contained four bar charts, then a value of four was recorded. In this way the types of visualisation that are close together, and record the most popular visualisation types, can be analysed.

Table 4.5: visualisation view types, calculated by year (2012 to 2018).

Vis.Type	12	13	14	15	16	17	18	Σ	%	
Bar Chart	29	22	24	40	72	60	89	336	13.86	
Scatter Plot	44	37	56	30	30	85	46	328	13.53	
Line Chart	23	18	39	17	33	62	36	228	9.40	
Heatmap	48	5	36	25	40	42	22	218	8.99	
Node Link	18	27	13	50	28	17	30	183	7.55	
Small Multiple	3	16	21	21	18	53	36	168	6.93	
Map	12	10	26	27	17	15	13	120	4.95	
Text	12	6	7	9	18	25	32	109	4.49	
Area Chart	9	12	13	7	25	14	21	101	4.16	
Other	6	6	29	13	5	10	3	72	2.96	
Rendered Image	1	7	23	5	15	1	12	64	2.64	
Parallel Coordinate	6	5	4	11	16	12	8	62	2.56	
Table	7	8	2	11	11	5	14	58	2.39	
Histogram	2	6	14	11	4	12	6	55	2.27	
Treemap	7	4	18	5	4	10	6	54	2.23	
Pie Chart	16	7	0	3	3	5	8	42	1.73	
Hierarchy	3	4	4	10	3	9	6	39	1.61	
Star Plot	1	11	1	2	0	10	4	29	1.20	
Timeline	1	4	2	5	5	8	3	28	1.15	
3D	3	3	3	6	4	4	3	26	1.07	
Matrix	2	0	2	3	4	5	8	24	0.99	
Point Chart	5	1	2	7	6	1	1	23	0.95	
Bubble Chart	2	1	1	0	6	1	11	22	0.91	
Image	8	1	1	2	2	4	0	18	0.74	
Glyph	0	0	0	7	2	0	3	12	0.49	
Video	4	1	1	0	0	0	0	6	0.25	
Total	272	222	342	327	371	470	421	2425		

One of the challenges was the issue of how to classify visualisations with many views. Are these multiple view systems with many individual views? Are they small-multiple displays? Or perhaps they could be a matrix view, such as a scatterplot matrix or SPLOM. Small multiples occur when they demonstrate several separate visualisations.

Grid-based visualisations are different from *matrix* views. Matrix views will show correlations between sets, such as showing ABCD by ABCD. We also added a *table* category, where the tabular values were mostly numeric.

In addition, from the data it was noticed that *Scatter plots* and *point charts* are often on time axis and show more complex symbols, while *bubble charts* present many circles and are not usually displayed on an axis. Lastly, some of the visualisations were impossible to categorise, because it was unclear how they were formed; and they were different to other visual depictions, subsequently they were tallied in the *other* category.

Table 4.6: Overall visualisation types in all layouts, the top-ranked visualisation types are mostly found in the top-ranked layouts.

Types	Σ	%	<div></div>	In Layouts with Quantity of Views									
				1	2	3	4	5	6	7	8	9	>9
Bar Chart	336	13.86	<div></div>	3	10	27	50	35	60	17	14	4	116
Scatter Plot	328	13.53	<div></div>	7	8	35	29	31	40	11	11	9	147
Line Chart	228	9.40	<div></div>	2	9	23	23	29	34	11	6	0	91
Heatmap	218	8.99	<div></div>	4	10	24	39	14	33	1	6	13	74
Node Link	183	7.55	<div></div>	12	15	19	20	37	36	8	12	0	24
Small Multiple	168	6.93	<div></div>	0	0	6	4	7	7	2	1	0	141
Map	120	4.95	<div></div>	8	26	16	9	10	13	6	3	1	28
Text	109	4.49	<div></div>	0	3	13	16	19	29	7	5	0	17
Area Chart	101	4.16	<div></div>	3	6	10	12	11	2	6	4	0	47
Other	72	2.96	<div></div>	2	1	14	6	4	6	1	5	3	30
Rendered Image	64	2.64	<div></div>	1	4	6	14	4	7	0	0	0	28
Parallel Coordinate	62	2.56	<div></div>	5	5	9	12	7	8	3	5	0	8
Table	58	2.39	<div></div>	1	2	3	17	10	13	5	2	0	5
Histogram	55	2.27	<div></div>	0	3	6	8	3	6	6	8	0	15
Treemap	54	2.23	<div></div>	3	6	10	7	4	3	3	0	2	16
Pie Chart	42	1.73	<div></div>	6	3	7	4	5	5	1	0	9	2
Hierarchy	39	1.61	<div></div>	2	4	4	6	4	6	0	0	1	12
Star Plot	29	1.20	<div></div>	0	0	0	3	4	4	0	0	9	9
Timeline	28	1.15	<div></div>	1	0	4	4	2	5	5	0	0	7
3D	26	1.07	<div></div>	4	4	8	6	1	0	2	0	0	1
Matrix	24	0.99	<div></div>	1	3	3	5	6	2	1	1	0	2
Point Chart	23	0.95	<div></div>	1	0	3	6	2	0	0	5	0	6
Bubble Chart	22	0.91	<div></div>	2	1	2	4	2	7	2	0	0	2
Image	18	0.74	<div></div>	1	2	2	6	0	4	0	0	3	0
Glyph	12	0.49	<div></div>	0	0	0	2	1	8	0	0	0	1
Video	6	0.25	<div></div>	0	0	0	2	3	1	0	0	0	0

Now, it is possible to answer the research question “what visualisation types are used in each view?” From Table 4.5 and Table 4.6 most of the top-ranking visualisation types are traditional visualisation charts (bar chart, scatter plots, line chart, heatmap, and node link diagram). In fact, nearly 14% of overall views include a bar chart, and the top five visualisation types take up over 53% of the total visualisation types in all views. It is clear that *bar charts* and *scatter plots* are the most popular visualisation types in multiple view systems.

Observations from Table 4.7 provide understanding that most of the top-ranking visualisation types are in the higher-ranking layout arrangements, and the five top-ranking visualisation types nearly appeared in all the layout arrangements. Furthermore, the complex layout arrangements with more views appear to have more simple visualisation types such as bar chart rather than complex visualisation types such as hierarchy. Nevertheless, the visualisation types with more details, such as a map, are in large views inside layout with fewer views.

In addition, there are some potential other trends that can be noticed, especially between the layouts and the visualisation types:

- Up to 48% of the overall visualisations are found in three, four, five and six views layouts.
- Most node link diagram (more than 86%) in layouts with views number is less than or equal to eight views.
- Node link diagram and map can be use on own in one view layouts, 17.39% and 11.59% from the one view layouts are node link diagram and map, respectively.
- However, small multiple, text, histogram, star plot, video and glyph visualisation types do not appear on their own, and they were not noticeable in any of the one-view layouts (represent 0% from the overall one view layouts).
- 84% of the small-multiple visualisation type are in layouts with view numbers greater than nine views.

Table 4.7: The quantification of the visualisation types in the top-ranked layout arrangements, as both were ranked based on their respective overall rankings.

Visualisation Types	Layouts Arrangements																									
	1A	2A	3A	3B	4A	6A	2B	3C	5A	4B	4C	3D	5B	6B	4D	9A	4E	4F	5C	6C	4G	4H	4I	5D	8A	
Bar Chart Scatter Plot Line Chart Heatmap Node Link Diagram Small Multiple Map Text Area Chart Rendered Image Parallel Coordinate Table Histogram Treemap Pie Chart Hierarchy Star Plot Timeline 3D Matrix Point Chart Bubble Chart Image Glyph Video	3	7	9	9	15	15	3	3	9	10	11	6	3	7	4	4	2	2	2	9	3	2	1	5	0	
	7	5	10	13	8	26	3	8	6	3	4	4	9	7	1	9	0	3	1	1	6	4	0	3	8	
	2	3	9	6	3	10	6	7	7	0	8	1	4	8	3	0	6	1	0	5	0	2	0	1	0	
	4	8	10	9	19	9	2	4	6	4	1	1	1	15	3	13	4	2	0	1	1	2	3	2	16	
	12	13	6	4	5	12	2	5	5	2	1	4	8	3	0	0	1	4	11	2	5	1	1	3	0	
	0	0	1	4	1	2	0	1	0	2	0	0	3	0	1	0	0	0	1	0	0	0	0	1	0	
	8	20	7	8	3	4	6	0	3	1	2	1	1	1	1	2	1	0	0	2	2	0	0	1	2	8
	0	2	8	1	3	3	1	3	4	1	4	1	4	0	2	0	0	0	0	0	9	0	1	5	5	0
	3	4	9	0	3	1	2	0	2	4	1	1	1	0	0	0	0	0	0	0	1	0	4	0	2	0
	1	4	6	0	6	4	0	0	0	0	1	0	0	0	0	0	0	0	0	4	0	4	0	3	0	0
5	2	2	6	5	5	3	0	2	1	1	1	1	2	0	1	0	0	1	0	1	1	1	1	0	0	
1	0	2	0	4	1	2	0	0	2	1	1	1	4	2	3	0	1	5	1	7	0	0	1	3	0	
0	2	3	2	5	0	1	1	0	1	0	0	0	1	3	0	0	2	0	1	1	0	0	0	0	0	
3	6	6	3	0	1	0	0	1	1	0	1	1	0	1	0	1	2	3	1	0	0	0	1	0	0	
6	2	1	5	1	2	1	1	0	1	1	1	0	0	0	1	9	0	0	0	0	0	0	0	0	0	
2	3	1	1	2	3	1	1	1	1	1	1	1	0	0	1	1	0	1	0	2	0	0	0	1	0	
0	0	0	0	0	1	2	0	0	0	1	0	0	1	0	0	9	0	0	0	0	1	0	1	0	1	8
1	0	1	3	0	0	0	0	0	0	0	0	0	0	4	2	0	0	0	1	0	0	0	2	0	0	0
4	3	3	1	3	0	1	4	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0
1	2	2	0	1	1	1	1	1	1	2	1	0	2	1	1	0	0	0	0	0	0	0	0	0	0	0
1	0	2	1	0	0	0	0	0	2	1	0	0	0	0	0	0	3	0	0	0	0	1	1	0	0	0
2	0	2	0	1	0	1	0	0	0	0	2	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0
1	2	1	1	1	5	0	0	0	0	1	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	6	0	0	0	1	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- Also, this study found that people usually use small multiple visualisations (more than 83% of the small multiple) when there are more than nine views in the layouts.
- Two-view layouts contain 21% of the map visualisations, as the map visualisations have lots of details and can fit easily in large views provided by the two-view layouts.
- 96.15% of the 3D visualisations are in layouts with seven views or less.
- Over 66% of the Treemap visualisations are in layouts with seven views or less, and Treemap visualisations tend to be in large and square views.

4.8 Quantification and analysis of collocational pairs of the visualisation types

This section focuses on answering the research question “Which types of visualisation come together?” by quantifying all the collocational pairs of the visualisation types in the multiple view layouts, and then analysing these results. The idea of this collocation analysis is to ascertain what views are typically found together.

Assuming that the pictures of visualisation tools in our database are typical and present a representative sample, and from our previous visualisation-type classification the next stage is to perform a basket analysis [18] to understand which views appear together. The results are shown in Figure 4.5, as a correlation matrix plot. This matrix provides a way to locate the collocational and non-collocational pairs of visualisation types in layouts with six views or less. After discussion, the focus was to look at six-viewed and less, layouts because they represent 85% of the overall layouts. In addition, the remaining 15% of the layouts are omitted from this investigation, because they represent only a smaller proportion of the overall layout and only have weak correlations, and therefore their omission does not impact significantly on the analysis of the results.

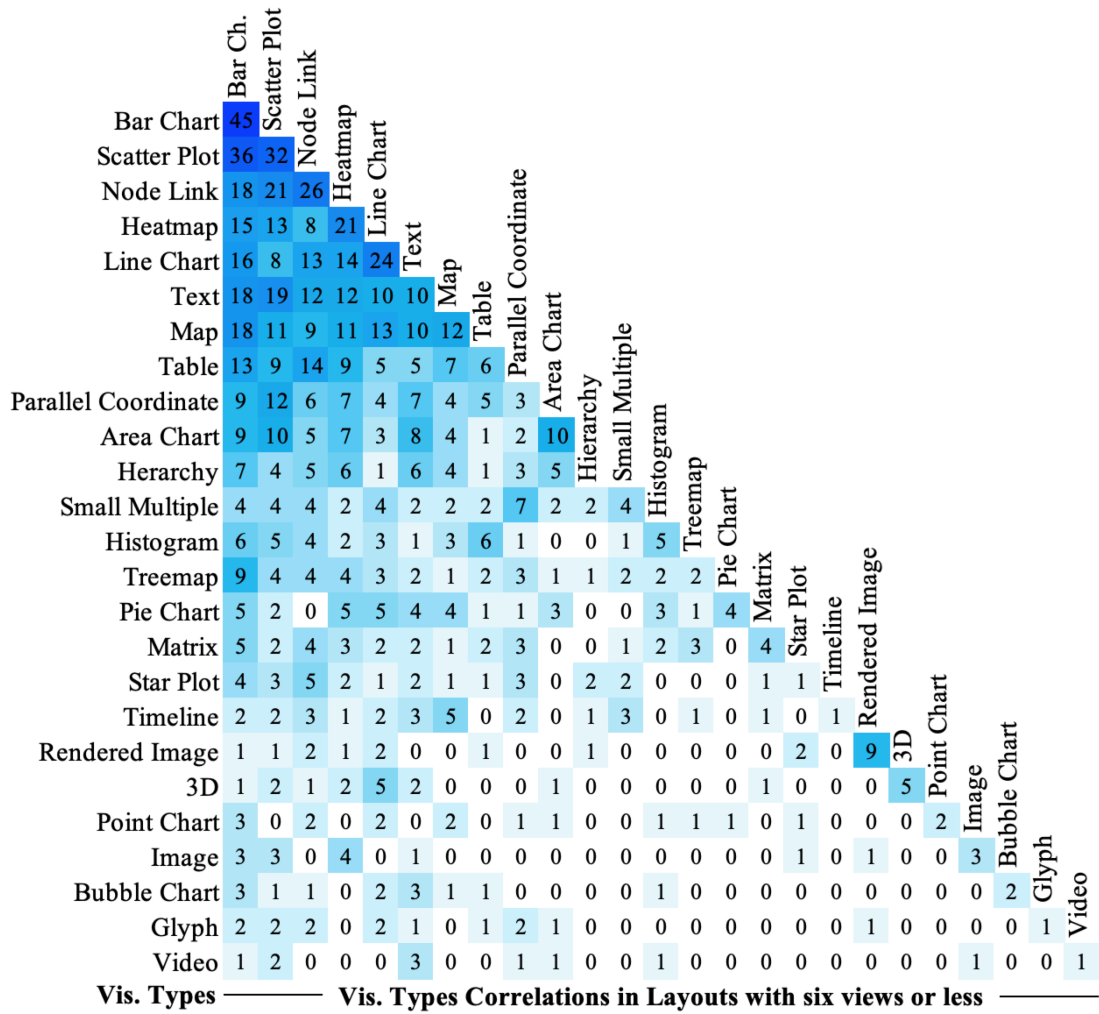


Figure 4.5: Correlations matrix of visualisation types, highlighted the collocational and non-collocational pairs for visualisation types in layouts with six views or less.

From the correlation matrix (Figure 4.5) it is possible to ask the question “which pairs of visualisation types go together?” The overarching correlation can be evaluated from all the visualisation types of 1058 correlations. Over 83% of the overall correlations were with bar chart, scatter plots, node link diagram, heatmap, line chart, text and map; and 44% of the overall correlations were between these visualisation types. Additionally, from Figure 4.5, we observe the following trends of the collocation of visualisations:

1. There is a high correlations between the visualisation types and itself. This can be observed by looking at the top instances down the diagonal ($x = y$). In other words, researchers duplicate view types to create their multiple view system, but

only in the popular visualisation types (bar chart, scatter plot, node link, heatmap, line chart, text, map, table). Consequently, designers appear to created multiple view systems with duplicate visualisation types of the more common types. For example, 18% (45 collocational pairs) from the overall correlations of the bar chart (253 collocational pairs) were collocated with other bar chart views.

2. While there are many instances of strong correlations with the popular visualisations, there are few correlations between bottom-ranking visualisation types and itself. See the lower portion of the diagonal ($x = y$) in the matrix view (see Table 4.5 and Figure 4.5). It appears that designers include one of these less-familiar visualisations alongside other more well-known visualisation types.
3. Finally, there are a few top-ranked correlations. Collocations between bar chart (bc), scatter plot (sp) and itself ((bc, bc), (bc, sc) and (sc, sc)) are the strongest correlations, 45, 36 and 32 correlations, respectively. This represents 10.68% from the overall collocational pairs.









4.9 Developing of design guidelines for multiple view visualisations

This section focused on providing design guidelines for multiple view visualisations. For this purpose, the information was distilled information from the investigation work, and ideas also from previous knowledge and experience, to introduce a set of eleven recommended design guidelines that can be followed to create robust multiple view visualisations.

It is clear to see that developers of multiple view systems have used many different layout strategies to position their views on the screen. Yet, there is a commonality in their design, and lessons learnt that we can subsume from the analysis.

- **Use fewer views.** The results demonstrate that more authors present systems with fewer views, we calculated that 84.68% of the layouts are 6-view or less, and 3-view layouts are the most popular, see Figure 4.3 and Table 4.1. Even though the results demonstrate that there is a preference for fewer views, some developers

still display systems with a considerable quantity of views, layouts with 7-views and more equal over 15% of the total amount of layouts.

- **Choose one of the main eight layouts.** The results demonstrate that there are eight top visualisation layout strategies are simple layout arrangements with fewer views, which are: 1A , 2A , 3A , 3B , 4A , 6A , 2B , 3C  (see Table 4.2). All these arrangements are symmetric with fewer views.
- **Use a symmetrical design.** If a developer wants to use different layout arrangements, it is better to employ symmetrical layouts. There appears to be a preference for symmetrical layouts over non-symmetrical layouts, see Figure 4.4. Perhaps people prefer the symmetrical views, and the designers are following principles of *balance* in their design decisions. Besides, symmetrical balance encourages an equal weight of both sides.
- **Variety is good** Visualize your data using a variety of different visualisation types; see Table 4.5, Table 4.6 and Figure 4.5 for a list of twenty-five different visualisation types that can be used in multiple view visualisations.
- **Use well-known visualisation types.** The results demonstrate that the two simple visualisations (bar charts and scatter plots) are widespread, and that together with line charts, heatmaps and node-link diagrams they form over 50% of all views in multiple view systems, see Table 4.5 and Table 4.6.
- **Include, at least one principal visualisation.** Have a principle visualisation that takes up most of the space (or more than other views), this is why visual designers often encourage the user to look at certain parts of the picture, and hint to the user to *flow* in the right direction. In this way, it is clear for the user to see what is the main visualisation, and where they should spend all their time. Other views will then depend on this view.
- **Make it simple.** Use simple visualisation types, particularly if you want to but lots of things together, it is more helpful to keep it in the simplest form. A developer should choose between bar charts, scatter plots, line charts, heatmaps and node-link diagrams, see Table 4.5 and Table 4.6.

- **Put complex visualisations in a simple layout.** If you are creating a complex visualisation with more details, put it in spacious view and have less views (including simple visualisations) around it within a simple layout arrangement. From Table 4.7, we notice that complex visualisations tend to be in simple layout arrangements with fewer views. This has happened because complex visualisations contain lots of details to be focused on, which is required to be in a view with ample space (in large view) so that the user can notice the details more clearly in less time. As example, 71% of the maps are in 2A layout arrangement which has two large views.
- **Divide a complex task into simple tasks.** Use views with simple visualisation types to achieve complex task, such as bar chart and scatter plots. Make sure that there are clear relationships between views with an obvious story to tell. Table 4.5 and Table 4.6 show that the top-ranking visualisation types are simple visualisations. This tendency shows the aim of developers to follow the principle of simplify the complex system's function into several simple tasks, where each simple task represented by a simple visualisation.
- **Separate views.** Put things that are associated closer together to make one view. And, use a light colour, or small gap or shadow to separate views, see Section 3.3.
- **Determine what is the purpose of the visualisation views.** This study realises that the purpose of the visualisation would change depending on the quantity of views used: with fewer views a user interacts with specific views; with many views the user is less likely to manipulate one view, but gains an overview of the information from all views. Moreover, if the views have equal importance it should have equal size.

This set of the design guidelines can help the novices in data visualisation field and developers to allow them create multiple view visualisations, and helps them make objective design decisions.

4.10 Discussion

This analysis has comprehensively investigated views as presented in the visualisation literature. The work has answered the proposed research questions (the last six questions), presented quantifications for multiple view systems, counted the layout types, quantified the visualisation types, investigated the connection between view layouts and the types of visualisation they hold, and finally looked to the correlations between the visualisation types to enumerate the collocational pairs of the visualisation types. In addition, the study analysed these statistics, which is help us to answer the last question and provide a set of design guidelines from our quantitative analysis. This work helps the community move closer to developing a theory of visualisation, and to provide guidelines on visualisation phraseology and use. Certainly the in-depth evaluation and presentation of results will give quantitative data to researchers to develop new sets of guidelines over layout strategies.

One challenge with the results is that, in order to calculate the views, there is a degree of generalisation. For instance, the scale and proportions of the views is not kept, but generalised into a layout. This helps to make decisions over the views, but some of the find detail is lost. However the work does discover several well-used strategies (see Table 4.2) and confirm that side-by-side, three-way, two-thirds and view layouts are popular. However, similar generalisations or constraints occur in the development of all visualisation systems. Software, that developers use to create the visualisations, often restricts what is possible. Often the developer is given a few layout options. This challenge can be overcome by giving the user control to change the layout to improve the display of the visualisation. In this case, there will be two challenges:

- Auto layout method which has to be determined at the first appearance for the visualisation.
- Give the possibility for the user to save the layout and apply it on another data.

The analysis acts as a starting point to help designers create better visualisations, acts as a taxonomy of visualisation layouts, and provides a quantitative analysis of how many views developers have used in their visualisation systems. One of our long term goals

for this study is to provide guidelines for newcomers. Certainly the in-depth evaluation and presentation of results will give quantitative data to researchers to develop new sets of guidelines over layout strategies.

The results demonstrate that developers are creating individual layout designs, but the majority of the design layouts are similar; which gives hope to provide quantitative view guidelines. The work also demonstrates that while the method enacts a reasonable simple quantification of views, the method helps to develop many conclusions. These include that visualisation layouts from one system to another are similar, and while in practice they include different visualisation types, they are structured similarly. However, there is more diversity when the views are investigated in detail. Furthermore, it is possible to observe that certain layout configurations afford particular visualisations and tasks. For example, those that contain a long thin view are typically used for a timeline, or a line graph.

It is clear, however, that there is still much information hidden in this data that could be extracted. In particular it is possible that the results and analysis will restart the debate over how many views are suitable, and will help to focus the minds of the developer as they create multiple view systems, to contemplate how they are laying out their views, and how many views they are using. Finally, this study provides a good starting point for more research. We encourage researchers to investigate various layouts and to consider our results when they are designing their next visualisation tool.

4.11 Limitations

As with all quantitative research there are limitations. One limitation with the analysis approach is that it works on historic data. In other words, evaluations are only calculated on views over the past few years. Furthermore the analysis is restricted to visualisations that are used in the academic domain. The focus on the past seven years of tools presented at IEEE VIS. While the work did investigate a broad range of structures only topology, view type and position is investigated. Furthermore, the work focused on view juxtaposition, and not on superposition, overlay or nesting.

Furthermore, the visualisation domain is continuously evolving. Many visualisation

tools are presented on the Web, developers are non-academic and therefore would not present their work at the IEEE VIS Conference, and there is a noticeable rise of immersive and interactive visualisation experiences. In fact, there are many opportunities to look beyond WIMP based (fixed screen) solutions [106], and this change will have an impact on the area of multiple views. A *view* will be embedded in an ephemeral 3D world that users will be immersed within. A similar challenge is that visualisation is becoming democratised, with the general public creating more of their own visualisations, and telling their own data stories. Thus many questions remain; such as how views are used by non-academics, or used by researchers in other fields, etc. It will be interesting to perform a similar study in the future to see how visualisation researchers are using *views*.

Furthermore, there are many more questions that have not been explored, and there is much further work to be done. Indeed, there is much more detailed analysis that could be performed in this area for example while the work started to investigate the connection between view layouts and the visualisation forms, as described in Section 4.7, it would seem sensible, from a design standpoint, that there is a strong correlation between view type and position in the layout. With (for instance) long and thin structures, such as timelines or line graphs, would be placed in long and thin layouts.

Additionally, it is possible to imagine that there may be a connection between the view layout strategy and its position of the layout in the article. Where, for instance, visualisations that have more views are placed along the top of the article, with those with less views in a column. Again, the analysis was uncertain. It was not possible to make this correlation, because location information was not encoded in detail. This is also left to future work.

4.12 Summary

This chapter focused on the quantification and analysis processes for multiple view visualisations evaluation, including counting the views number, the symmetrical layouts, the layouts arrangements, the visualisation types and the collocational pairs of the visualisation types, and analysed these statistics. In addition, this chapter posed a series of questions, exploring and investigate the quantity of views that people use, and their design layout configuration. The result was a framework of ideas and design strategies for multiple views in visualisation that would be useful as a resource for new researchers and a reference for experts as a set of design guidelines for multiple view visualisations.

Moreover, this study presented in-depth results of a quantitative analysis of how developers laid out the visualisations in their multiple view systems, and what are the visualisation types that are been used in these systems as reported in the papers, workshops, and posters published in the IEEE VIS Conferences 2012 to 2018. Consequently, this chapter answered six research questions: “Q1(RQ2)/ How many views are used in multiple view systems?”, “Q2(RQ3)/ Do developers prefer symmetrical or non-symmetrical layouts for multiple view systems?”, “Q3(RQ4)/ What layout arrangements are popular in multiple view systems?”, “Q4(RQ5)/ What visualisation types are used in multiple view systems?”, “Q5(RQ6)/ What types of visualisation come together in multiple view systems?” and “Q6(RQ7)/ What are the guidelines to design multiple view visualisation?”.

This chapter started with quantifying the views that been used in multiple view systems, counted the *number of views* in each layout to find the most popular layout based on number of views. Then, this evaluation chapter looked to the *symmetry feature* in these multiple view layouts, quantifying all symmetrical layouts and non-symmetrical layouts to find which type are mostly used by the designers.

Moreover, this research classified the layouts based on its structure, grouped together all the layouts that has the same views number and structure. This process helped to find that the most popular layout arrangements in the multiple view systems has less than seven views, while there are less layouts with 7-view or more, with more variety in layout structure.

Furthermore, this study enumerated the types and the numbers of the visualisation techniques that are used in these selected multiple view layouts. These statistics helped us to find the most popular visualisation types in layouts, also to find the most popular visualisation types in each layout arrangements. In additional, this study observed the correlation between the visualisation types, and recorded all the collocational pairs of the visualisation types to find what visualisation types come together.

Subsequently, from analysis our statistics of counting the multiple view visualisations, we can summarise our findings in five headlines:

- More authors present systems with fewer views, 84.68% of the layouts are 6-view or less. The most frequent view was a 3-view system.
- Authors prefer symmetrical over non-symmetrical layouts.
- Dual view, three-view and four-view layouts are popular.
- More simple visualisations are used. bar chart, scatter plots, line chart, heatmap, and node link diagram account for more than 53% of the overall view types in multiple view systems.
- There are strong correlations between the simple visualisation types, secondly, between the simple visualisation types and the other visualisation types.

Finally, this study delivered another contribution through this chapter which is providing an objective design guidelines that will help developers to create robust multiple view visualisations (this is the answer of question six), such that developers can make objective design decision when they create multiple view visualisations.

In the next chapter, the results and design information captured here are applied to develop a grammar for multiple view layouts. Furthermore, a new tool is developed, based on the knowledge from on the quantification and analysis processes for multiple view systems that done in this chapter, the grammar can be used to create, control and save multiple view layouts.

Chapter 5

Design and development of a grammar for the MV Layouts tool

This chapter focuses on developing a grammar for multiple view layouts which will give developers more flexibility in creating, saving and reloading multiple view layouts. The chapter focuses on the following research questions and is likewise structured to answer these challenges:

q1/ What is a multiple view grammar?

q2/ Why users need a multiple view grammar, and how is it used?

q3/ Why we did choose to create our multiple view grammar over another grammar?

q4/ How is our multiple view grammar structured?

q5/ What is the advantage of using multiple view grammar, and what is the limitation?

5.1 Introduction to design of the grammar

There are very few tools that help developers and users create multiple view visualisations. Another option is for developers to use the code to create and organise multiple view layouts. That means the user has less control over the layout. Sometimes the programmers code movable windows, where users can drag the windows around. For example, North and Shneiderman's Snap-Together [94] allows users to place views side-by-side. And other systems such as Weaver's Improvise [148] allow developers to code side-by-side views. However, while these systems are extremely command-rich, they are not very intuitive to control where the views are positioned. For example,

Improvise requires a high level of code knowledge to control effectively. Systems such as D3.js (d3js.org/) [16] allow people to code and craft multiple views, and libraries such as Vega and Vega-lite enable users to more easily craft multiple view visualisation systems. However, again these systems require a reasonable knowledge of the underpinning code. Systems such as Adobe's data-illustrator [74] permit users to drag and drop and craft visualisations without programming. Libraries such as R and Shiny (shiny.rstudio.com/) grant people the ability to describe multiple-view layouts through markdown descriptions, which are more approachable for general coders. However, while each of the systems have advantages, have a general purpose rather than being focused on multiple views.

The goal of this research, and presentation in this chapter, is to develop something that can be expressive and simple (like Shiny-R and markdown code), something that is comprehensive like Vega-lite but more specific to the purpose of creating multiple-view visualisation layouts. And something like Snap-Together and Data Illustrator, which allows users to readily craft designs through simple drag and drop interfaces.

This chapter explains a new layout grammar: named Multiple View Grammar (MVG). The grammar represents a way to express multiple view layouts, and fulfils several purposes. The first purpose is to allow developers to create software that can save, and reload multiple view layouts, as well as It would allow developers to share them across platforms. Second, it allows developers to create systems where multiple-view-layouts can be edited. For instance, the grammar can be directly integrated into a visual editor, or a visualisation editing system, to allow users to craft and describe their own layouts.

The overarching philosophy of the grammar design is one of 'cutting'. For example, consider holding a sheet of A4 paper and cutting it into sub parts. Each part represents one view in a multiple view system. When cutting the sheet of paper, a pair of scissors is constrained to only cut in straight lines through the whole sheet. While a simple concept, it is can create very complex multiple view structures.

The MVG grammar is based on the concept of cutting a view horizontally (H) or vertically (V). So starting with a page and using a vertical cut, it will create two views. The idea is that users of this grammar can repeat this process on previously generated

views to produce more views to build the multiple view layout. Figure 5.1 shows this principle in action in a state-diagram. Starting with a Horizontal cut (for instance) moves to the left state. Then choosing one of those sub parts, gives another “View” and moves back to the centre state. Then the user can choose to cut this View into sub-parts, and so on. By stipulating the proportions of the cuts — left cut and right cut, for the Horizontal cuts, or top/bottom proportions for the Vertical cuts — it is possible to control the layout of the multiple view layout.

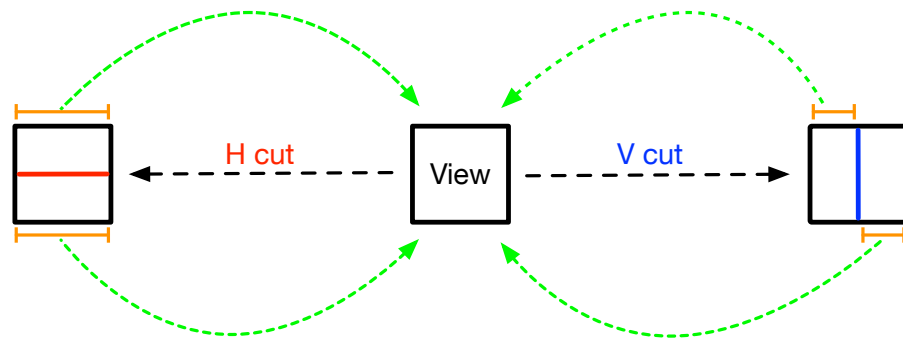


Figure 5.1: The picture demonstrates the Multiple View Grammar. It presents a state-diagram to explain the cut algorithm. The user can cut the main view horizontally (so it becomes two views) or vertically to create two-view vertical layout. This strategy can be repeated on the generated views to create more complex layouts.

This chapter presents the specification of the multiple view grammar based on the multiple view layout model. Design decisions are explained and discussed, and the MVG grammar is explained with examples. This chapter is structured as follows:

- Section 5.2: This section explains the characteristics of multiple view grammars in general, and what are the rules that should be followed to develop multiple view grammar?
- Section 5.3: This section discusses the design decisions to develop the MVG grammar and the consideration of alternatives.
- Section 5.4: This section explains the design process of the MVG grammar and how it is structured.

- Section 5.5: This section gives examples for the MVG grammar to explain how it is work.
- Section 5.6: This section discusses the advantages and the limitations of MVG grammar.

5.2 Developing the rules for the MVG grammar

This section covers the MVG grammar. The most important feature for the MVG grammar is that the grammar expressions can describe any multiple view layout and gives the ability to the multiple view tool to create, save and reload the layouts, which will help users to use the same layout with different data and/ or different visualisation techniques.

It is important, with any grammar expression, that every time the grammar is used it should produce the same layout each time. In other words, the grammar needs to be deterministic. There may be situations where a non-deterministic grammar is useful. For instance, it would be feasible to consider creating a system that creates a random layout of, for instance, a 3x4 configuration. However, any such non-deterministic conditions can be stored in a deterministic expression, even though the result is non-deterministic. In other words, one way to achieve this is that the grammar saves the final output choice, rather than the many options. Alternatively, it is possible to envisage a situation whereby a command is stored in the grammar to express that “the user wants a random 3x4 layout”. Furthermore, deterministic grammars are easier to parse. This process required a parser to transform the expressions of the MVG grammar into actual multiple view layouts, which required a set of algorithms that convert the MVG expressions to multiple view layouts.

Many developers have created a range of grammars, across computer science, for different purposes. These different structures allow developers to craft and save descriptions. For instance, markup languages, such as HTML, \LaTeX , are human readable and were originally designed to be edited by a human. However, there are editors that can be used to output them, whereas, other languages such as PDF or Postscript are really too complex for a human author to write, and require editors to

create. Over the past years, there has been a desire by many programmers to create languages and grammars that are human readable, yet able to be read by a computer and creates very complex output. Systems such as Markdown, AsciiDoc, and Jeckyll fit into this category. For example, an AsciiDoc (asciidoctor.org) document is written in plain English, with simple commands to markup and describe the document content and its structure. To use the AsciiDoc information, there is a wider tool chain that loads the document, parses it and formats it based on a layout specification. So, an author may describe their document with a human-readable file (an `.adoc`), and use a tool to interpret that code. AsciiDoctor pulls in a style sheet, which describes how it will look and output an HTML file. The same `.adoc` can be used to output a PDF, or an HTML file with a different appearance.

Even considering Microsoft Word, they have moved from a non human-readable file format `.doc`, and now to a hierarchical file structure `.docx`. Indeed, the `docx` system is really a zip file of different files, put together in one folder hierarchy. This means that it is much easier for 3rd party software to read and write the files, and for humans to edit the information. It also helps to future proof the system, as it is not reliant on a specific binary formatting. E.g., it would be possible to create a Java program and save it as a class file. This is a very simple solution. However this solution is not future proof. As by changing any of the class structure means that the files that had been saved under a previous version would not be readable.

This tool-chain structure was inspirational to us, and it would be possible to create a tool chain for describing multiple views, storing it in the grammar (MVG), and then using the multiple view structure, and so on.

Consider creating a multiple view file (our MVG). This file can be read by a grammar tool, edited by a grammar editor, incorporate data (saved in an external file) and incorporate a visualisation library (such as D3.js) to output the information.

Subsequently, the requirement was to create something that can be written out by a user to a file, and this file should be readable by a human, and potentially editable by an ASCII editor. In this way it would be possible to create a graphical user interface that then can save the grammar to the file, where it could be loaded to display a specific


multiple view layout, and could be edited either by a specific grammar editor that we create, or by an ASCII editor. In summary, four specifications are made, for the design of the MVG grammar:

- 1/ It needs to be deterministic.** The MVG grammar parser will produce the same output; in other words, every MVG grammar expression should always produce the same multiple view layout each time we run the MVG grammar parser. The same thing should happen when we save and reload the multiple view layout. Moreover, the MVG grammar expression which represents the saved layout should be saved into a file with a deterministic format, and by using deterministic algorithms every time we run the file we will have the same predictable layout.
- 2/ It needs to be human-readable.** The MVG grammar needs to be different from the Computer-implemented grammars, where in the Computer-implemented grammar it would be possible to develop a grammar that was only machine readable, with such grammars the computer writes it out, and it is not going to be edited by a user.
- 3/ It needs to be machine readable and parsable by a computer.** Similar to the markup languages HTML and CSS which used DIVs components to describe the multiple view layout, where the code of these languages are still readable by human, the MVG grammar should be parsable by a computer. For that, the MVG grammar parser used deterministic algorithms to transform the MVG grammar expressions into codes, these codes are coded using JavaScript and CSS languages, which can be read by a computer to produce the actual multiple view layouts.
- 4/ It needs to be functionally rich to allow users to be able to describe some difficult cases.** The purpose of creating multiple view visualisations are varied, and that required a variety in its layout structure to achieve all these purposes. For that, the user should create all the possible layout structures, including complex layouts, this variety in layout structures came from the difference in the number of views in each layout and the difference in the layout's structures. For that, the MVG grammar should allow the user to create any required multiple view layout to achieve his goal from creating his visualisations.

These rules provided a way to constrain the design and create reliable grammar expression that creates one existing layout every time the parser transforms the grammar expression into multiple view layout. At the same time, this grammar is human readable and can be run by the computer to create any potential layouts.

5.3 Design of the MVG grammar

When we started designing a layout grammar for our multiple view tool, there were two choices; the first choice was to use an existing grammar such as Vega-lite grammar, and the other choice was to create our grammar. We thought, we could use Vega-lite grammar to describe the layout, but the problem with Vega lite grammar is, at that time especially, Vega-lite was new, and there had not been the widespread take up there is now. Even now, it does not mention much about layout, rather it focuses on creating visual depictions.

Also, we contemplated using the HTML infrastructure. Perhaps use a Div to describe a grid system, put in HTML with CSS potentially use Frames within the HTML structure. For instance, a 4B view (1-1-2) view  could be setup in frames using the code showing in Listing 5.1. However, the frame structure was made obsolete in HTML5. There are several reasons for this. First, because it caused compatibility problems across browsers. Second, it provided accessibility challenges. Third, all this functionality can be better implemented using CSS. This is because what was wanted for the web2.0 was a way to split content, while keeping information close that related together. In other words web developers wanted the content to display (equally well) on a large-screen setup or a small mobile phone. Consequently, these goals are different to ours. We want to specifically control the layout of the views, and put information side-by-side when required. However there are some interesting and relevant ideas with the HTML frame constructs. First, frames allowed developers to control the position. Through using the command * developers express that the frame fits the available size. This is similar to the \LaTeX command `hfill`. Second, frames could control exact window sizes, or proportional sizes. For example, `<frameset cols="20%,*,20%">` defines a column with the first and last column being 20% each, and the centre part to fill in the space (which would be 60%).

Listing 5.1: Multiple views described in HTML frames

```
<frameset cols="*,*,*">
  <frameset rows="*,*">
    <frame src="frame1a.html">
    <frame src="frame1b.html">
  </frameset>
  <frame src="frame2.html">
  <frame src="frame3.html">
</frameset>
```

However these structures do not fulfil the goal or requirements. What is required is a way to describe views that can be saved and loaded. The HTML description was quite verbose and would be needed to be adapted to fit the desired purpose. However, it could be possible to adapt the Div structure to fit the goals. Nevertheless, a description file is required, that could act as an intermediary to create, save and reload multiple view layout, which is made specifically for multiple view systems. This file should describe various layouts. Consequently, the use of Frames and also Vega-lite were rejected.

There are other web structures that are useful. JSON files allow information stored in a text file to be exchanged on the web. It is a common file format, used throughout the web, and would allow us to save the information of the view structure. It would also allow users to save additional information about the visualisation and any associated data. It can act as an intermediate file between the design-environment (e.g., drag and drop, front end) and the grammar editing interface, and the saved file. It has the advantages of being web-enabled, and can incorporate the visualisation information.

As we mentioned before and shown in Figure 5.1 above, the principle of cutting a view into two views vertically or horizontally to create multiple view layouts and repeating this process will split a different part of the layout as a hierarchy. The MVG grammar could describe this process by first determining the type of the cut, whether horizontal or vertical. Second, the area ratio of generated views from the parent view needs to be specified. For example, when the goal is to create a layout with two equal side-by-side views, in this case, the grammar expression should be “**V(50,50)**”. Where the letter **V** represents the vertical cut, and the numbers **50** and **50** represent the area ratio for the generated views which are taken from the parent view. In the next section, the MVG Grammar is described in detail.

Listing 5.2: BNF description of the MVG

```
<Code> ::= <Expr>  
<Expr> ::= <Op> + " (:␣+␣<Area>␣+␣" ) :  
<Area> ::= <Variable> + ", " + <Variable>  
<Variable> ::= "number" | "number" + <Expr>  
<Op> ::= "v" | "h"
```

5.4 MVG grammar for multiple view layouts

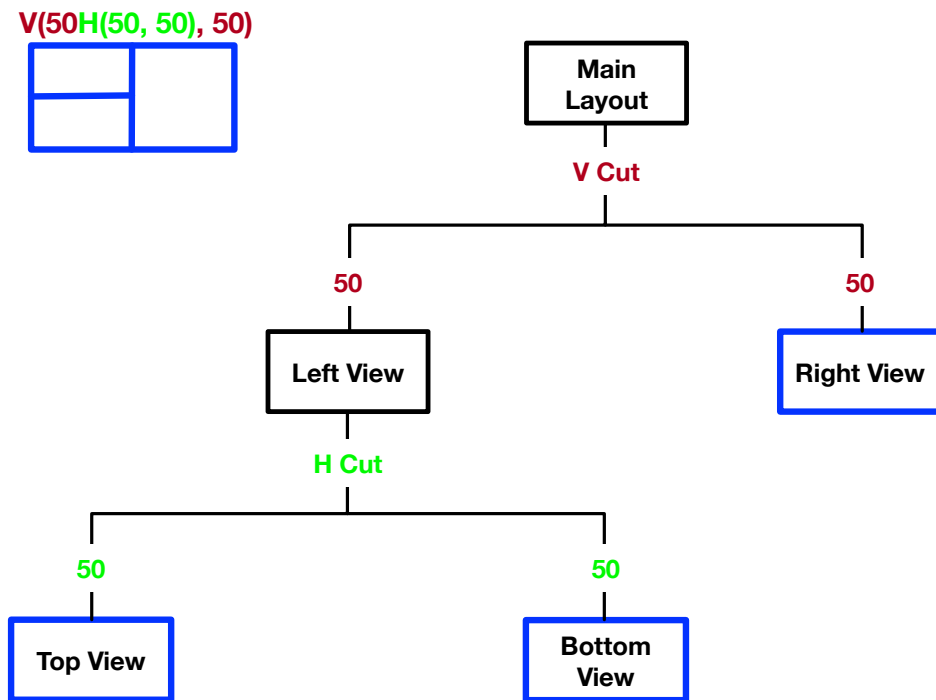



Figure 5.2: The MVG grammar expression “ $V(50H(50,50),50)$ ” describes the layout  in a hierarchy structure.

The same hierarchical method can be used to describe any multiple view layout, where we can change or add more vertical and horizontal cuts to the grammar expressions, in addition to changing the area ratio for the views.

Also, the structure of the MVG grammar expression can be formally written using the Backus normal form (BNF), as shown in Listing 5.2. BNF is a convenient meta-syntax, that is useful for context-free grammars. Non-terminal= { Code, Expr, Variable, Area, Op} and Terminal= { “number”, “V”, “H”, “(”, “)”, “;” } Where:

Expr is the MVG grammar expression.

Area is the layout area or a view area

OP is the operation.

v is the vertical cut.

h is the horizontal cut.

number is the area ratio of a view from the area of the parent view

In addition, there are some view combinations that can be shortened. By making some shorthand descriptions the MVG grammar can be simplified. This simplified version can be used to create well-known layouts structure such as a Grid and the Golden Ratio structures. This will make creating these multiple view layouts much easier. These are referenced as a ‘shortcut’ grammar. E.g., user can use the MVG grammar expression “**grid 3x3**” to create a grid layout which has nine views as shown in Figure 5.3.

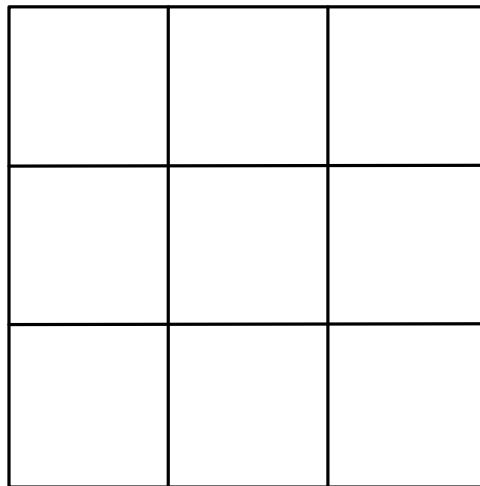


Figure 5.3: The shortcut MVG grammar expression “**grid 3x3**” created a grid layout with nine views.

Moreover, the same technique can be used to create a golden ratio layout using the MVG grammar expression. For example, we can use the expression “**GoldenRatioV6Q4S70%**” to create a layout as shown in Figure 5.4.

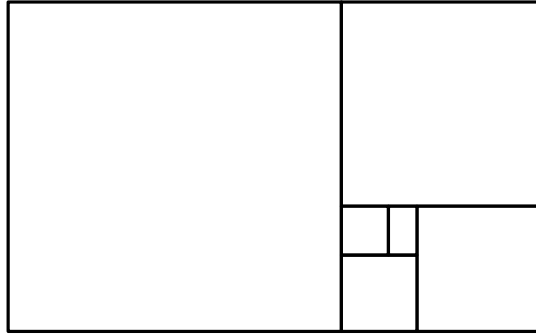


Figure 5.4: The shortcut MVG grammar expression “**GoldenRatioV6Q4S70%**” created a golden ratio layout, where “**V6**” is the number of views in the layout (in this example, we have 6 views), “**Q4**” indicates the position of the golden ratio in the layout (in this example, the golden ratio is placed in the fourth quarter of the layout), and “**S70**” represents the size of the layout which is 70 percent of the multiple view tool panel, and we will give more detail about it in the next chapter.

Section 5.5 “**Examples for the MVG Grammar**”, provides more examples to explain the breath of possibilities with the MVG grammar.

5.5 Examples for the MVG grammar

This section focuses on explaining the functionality of the MVG grammar. Moreover, the following examples will explain the depth of the MVG grammar; where the MVG grammar is used to express various multiple view layouts.

Example 1. The first example starts simple. It provides a 3-view system, of two views on the left hand side, with a larger vertical view on the right, written as **v(50h(50,50),50)**. It can be considered in a two step process: First, the vertical cut **v(50,50)** divides the layout into two views vertically. Second, the expression **h(50,50)** divides the left layout into two views horizontally, as shown in Figure 5.5.

Expr : v(50h(50,50),50)

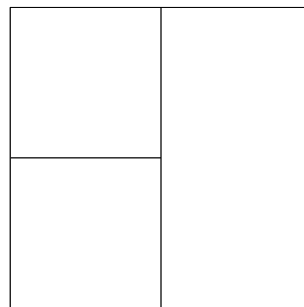


Figure 5.5: On the left, we show the grammar code **v(50h(50,50),50)**. Where, the rendering of the code is shown on the right.

Example 2, demonstrates a more complex layout. The idea is to extend the Example 1 (Figure 5.5) and adds an additional cut. This is achieved by changing the size of the views and adding one more view by using the grammar expression $v(75h(25,75v(50,50)),25)$ to describe the layout, as shown in Figure 5.6.

$v(75h(25,75v(50,50)),25)$

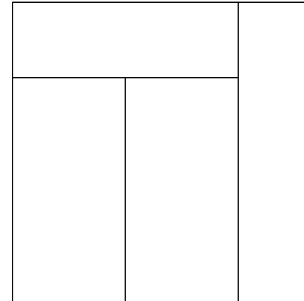


Figure 5.6: On the left we show the grammar code $v(75h(25,75v(50,50)),25)$. Where, the rendering of the code is shown on the right. And, as shown in the picture, we can define the size of the view by changing the view's ratio in the grammar expression

Example 3, demonstrates how the user can define variables to write the MVG grammar, this will simplify the grammar expression when the grammar describes a complicated multiple view layout. The $v(40h(25,75vb),20hb,20hb,20hb)$ expression creates complex layout in a simple way, where “a” is a variable equal to “50” and “b” is a variable equal to “(a,a)”, as shown in Figure 5.7. Obviously, if variables are not used, then the grammar expression will be $v(40h(25,75v(50,50)),20h(50,50),20h(50,50),20h(50,50))$.

a : 50
b : (a , a)
Expr : v (40 h (25 , 75 v b) ,
 20 h b , 20 h b , 20 h b)

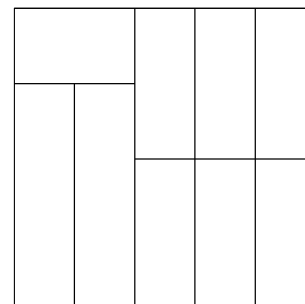


Figure 5.7: This example shows how the user of the MVG grammar can define variables to describe more complex layout.

Example 4, this example shows how more complicated layouts can be created, using more variables. The variables enable values to be changed more effectively, as shown in Figure 5.8.

```
a:25
b:( a , a , a )
c:20hb
Expr:v(40h(25,75vb),c,c,c)
```

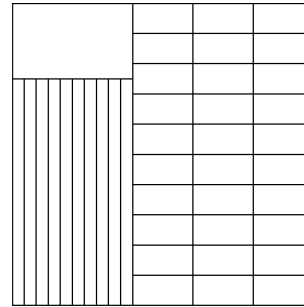


Figure 5.8: The grammar code, on the left, shows how variables can be used. Where, the rendering of the code is shown on the right.

Example 5, in this example we used the MVG grammar expression **v(38.1h(38.46v(62.5, 37.5h(60,40v(33.3,66.7))),61.54),61.9)** to describe a golden-ratio layout, and that expression represents a complex grammar, as shown in Figure 5.9. Alternatively the solution can be achieved using the shortcut grammar **“GoldenRatioV6Q2S100%”** to describe this layout, which will give the same result.

```
Expr:v(38.1h(38.46v(62.5,
37.5h(60,40v(33.3,66.7))),
61.54),61.9)
```

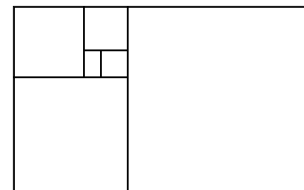


Figure 5.9: This example show how to use a complex grammar expression to describe a complex multiple view layout.

Example 6, this is another example of using a complex MVG grammar expression to describe a multiple view layout, even when we used the variables **“a”** and **“b”** in the expression **“v(25hb,25hb,25hb,25hb)”**, that did not simplify the grammar expression, as shown in Figure 5.10. However, as we mentioned before, an alternative solution is to use the shortcut grammar **“grid 4x4”** to describe this layout.

```

a : 25
b : ( a , a , a , a )
Expr : v ( 25 hb , 25 hb , 25 hb , 25 hb )

```

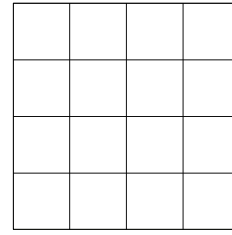


Figure 5.10: On the left we show the grammar code. The rendering of the code is shown on the right. Or, we can use the shortcut grammar “**grid 4x4**” to create the same layout.

5.6 Discussion

The goal was to develop a grammar that describes multiple view layout, and this grammar can be used in multiple view visualisation tools to create a multiple view layout; the MVG grammar created by using the principle of cut a view into two views horizontally or vertically.

This grammar is helpful on many levels because it is simple and matches the slicing and dicing ideas, and it is relatively easy to parse. Moreover, the MVG grammar is reliable because it is working and we used it to code all the layout we want, and this grammar allows the user to save the structure of the layout and load it again. In addition, this grammar allows the user to edit the layout.

When starting this research, about using grammar in our multiple view tool, there were many considerations made: such as using existing grammar such as Vega-lite. However, each of these grammars require a specific parser, and the grammar is designed for specific purposes, so as developers it was not possible to have complete control over the language. It would be difficult to adapt it to the research purpose and change it in an efficient timely manner. It would require change to the tool design for this thesis. Consequently the decision was taken to create this new grammar.

Actually, if Vega lite was chosen, and while it would be able to describe multiple view visualisation layouts, it would be challenging to extend the functionality such as the ability to craft bespoke shorthand layouts such as `GoldenRatioV6Q4S60%`. Using JSON files it is possible to embed it into any type of visualisation system, and subsequently use the grammar with D3.js commands. Furthermore, by creating the bespoke layout

structure it is possible to adapt it for any purpose, as it provides more control over the layout description.

In addition, this study considered how to develop the grammar in a formal and effective way. This done by using BNF notation method, a representation that helped us to cover all the possible of multiple view layouts that can be made by users. Moreover, the MVG grammar expressions can build any juxtaposition multiple view layout, where the views are positioned beside each other. Furthermore, using this definition to represents the layout, it is possible to use it in any multiple view tool, to save and reload the layout and even use the layout with other visualisations or data.

Nevertheless, there are potential limitations. Some layouts may not be possible or may be difficult to create. For instance, creating a complected layout with many views would be difficult, because the grammar expression would be large. This would not be impossible, just tedious to create by hand. However, shorthand definition, do simplify this problem. So for instance, golden ratio with many levels could be possible, but would be very tedious to create without the shortcut grammar version.

There could be other shorthand definition such as to define ‘adjacent’ commands, or packing commands that organised many views in a specific lattice layout. Reconfiguration may also be possible, where the layout that describes percentages of requirements. This is similar to the floating layouts in \LaTeX , which are controlled by values. The user can say that they want the figure to be here [h], or at the top [t] and so on, but \LaTeX may decide that this is not possible and consequently they are floated to the next page, as appropriate. Likewise it may be possible to say ‘put this here, probably, and if it will fit’ otherwise ‘put it over there’, ‘if these positions are not possible then do your best’. To achieve these commands, heuristic algorithms are used. Similar algorithms are used in label-placement visualisation (see for example work by Christensen et al. [33]), or in graph drawing layouts (e.g., see work by Di Battista, Peter Eades and colleagues [38]). Other (non juxtaposition) layout types, such as overlaying layouts, are not covered by the MVG grammar. Indeed, it will be interesting to develop the MVG grammar to describe overlapping multiple view layouts. In addition, even including the visualisation in the MVG grammar as an option will be an excellent addition to the MVG grammar to describe the entire multiple view visualisation systems.

Moreover, it will be interesting to develop the MVG grammar to describe these multiple view layouts. In addition, even including the visualisation in the MVG grammar as an option will be an excellent addition to the MVG grammar to describe the entire multiple view visualisation systems.

Therefore, this study considered to use a simple version of the MVG grammar when we want creating more complex layouts, for that we considered using variables to build complex grammar expressions and also we created a short-cut grammar which is very short expression to create specific types of layouts, as the use of the normal MVG grammar will be hard to tackle these layouts.

5.7 Summary

This chapter focused on describing a new grammar that can be used to describe and create multiple view layout. Consequently, this chapter answered five questions: “Q1/ What is a multiple view grammar?”, “Q2/ Why users need a multiple view grammar, and how is it used?”, “Q3/ Why we did choose to create our multiple view grammar over another grammar?”, “Q4/ How is our multiple view grammar structured?” and “Q5/ What is the advantage of using multiple view grammar, and what is the limitation?”.

This study developed rules for the MVG grammar, that make this grammar more formal, where each grammar expression describes a unique multiple view layout. This chapter determined the specification of the MVG grammar and we explained ability of this grammar, and we specified the goals we want to achieve from this grammar. Next the chapter discussed the design decisions that were made to develop the MVG Grammar and the consideration of alternatives, and included a comparison between different styles of grammar. After that, the work focused to explain how the grammar was created and discussed the different design decisions that were made over the grammar and solution.

Subsequently, the MVG grammar was defined, and how it could be used to create multiple view layouts, the structure of the MVG grammar was explained, and how it works. Principles underpinning the grammar were explained: how the idea of vertical and horizontal cutting can be used to create the layouts, and used to develop a hierarchical structure. Finally, this chapter presented examples for the MVG grammar,

to explain the depth and functionality of the MVG grammar, and demonstrated how the grammar can be used to explain several example layouts.

This chapter focused on, and answered the first part of the research question “RQ8/**What is a multiple view grammar** and how can it be used in multiple view tools to create multiple view layouts?”. The next chapter will use the MVG grammar and the shortcut grammar for the the MVG grammar to create multiple view layouts using the multiple view tool that we present in this thesis.

Chapter 6

Design and implementation of the MV layouts tool

This chapter focuses on developing a tool for multiple view layout. The chapter describes the design and implementation of the “**L**ayouts for **M**ultiple **V**iews” (LayMV) tool, a web-based software. This tool will give developers of multiple view visualisations more flexibility in creating, saving and reloading multiple view layouts.

The chapter concentrates on the following research questions and is likewise structured to answer these challenges:

Q1/ Why we want to develop the LayMV tool.

Q2/ What methodology should be used to design the LayMV tool? And, what alternatives are there?

Q3/ What process should be followed to implement the LayMV tool?

Q4/ What is the role of the MVG grammar in the LayMV tool?

Q5/ What methodologies could be used to evaluate the LayMV tool?

6.1 Introduction

Visualization tools, libraries and systems all help users create visualisations. While there are many ways to create a visualisation, controlling the layout of multiple view systems is still difficult. This is not the case for websites, where there are many design tools to help users layout their websites. Moreover, Java programmers can use methods such as GridBag Layout to easily organise different components, or a BorderLayout to place nodes: top, bottom, left, right, and center. Why can we not have the same idea

in visualisation? Visualisation developers would likewise benefit from a similar system. Templates could help users follow ‘typical’ layout strategies, and methods to graphically design different layouts. While there are similarities between a web page structure and a visualisation multiple view layout, there are differences. Both present information, both often have multiple facets – web pages have the main story, adverts, menus, while multiple view visualisation tools have many faceted views. But, their goals are different. Web pages have a goal to inform and provide predominantly written information, and do not require much control or linkages between facets, whereas multiple view visualisation tools are typically more visual, and many are highly coordinated interfaces. But people do often create visualisation tools on websites, and consequently, we used the a web page as an end-platform for our LayMV tool.

This study presents LayMV (Layouts for Multiple views) tool, the goal of which is to allow users to lay out their visualisations using pre-designed layouts, or easily create their own design configurations. This tool helps users create and re-configure different multiple view visualisations. Yet, it is not focusing on creating the visualisations for multiple view. Instead, it is used for laying out and controlling visualisation views. where, LayMV creating visualisations by using D3 library.

The motivation is to give users the ability to create juxtaposed view layouts in a simple and easy way. Furthermore, the goal is to allow users to create ‘typical’ layouts, and consequently we drew on our quantification work as explained in Chapter 4 and presented in Almaneea and Roberts [81] [82]. The work quantifies and identifies typical and frequent layout strategies as used in the scientific literature and presented by visualisation researchers. This tool utilises the results from these previous quantitative studies, and the tool helps guide users on popular arrangements.

This chapter demonstrates how the LayMV tool was designed, implemented, and presents screenshots of the tool in action. To motivate and guide the research, at the start of this process, several research questions were posed: “How can we develop a system to help visualisation users lay out their views?”, “How can we allow users to quickly layout their views, and then easily change their layout design?” and “How do we map data to a view and easily change the appearance of the layout viewer?”. These questions focus around the development and use of the layout tool. At the start of the research, it

was clear that the goal was to create something that would help people layout views. But how? What would the tool look like? Would it be focused around the grammar, or a drag-and-drop (or similar) interface? Additionally, we asked: how would it be used? Developers would need to use the layouts in their own applications. But how? How can these structures be created, without (potentially) building a whole design system?

In addition, this chapter explains what is this LayMV tool is used for, how it was built, the main components of the tool and what it will look like. This chapter also explains how the MVG grammar is parsed and used to create different multiple view visualisations. Furthermore, this chapter explains the functionality of the LayMV tool and discusses what methodologies could be used to evaluate it.

The tool is designed based on the design guidelines from Chapter 4. We start the design with sketches, using the five design sheet method (FDS) [117, 118], low-fidelity prototype, and developing a prototype in JavaScript, HTML, D3 and JSON (JavaScript Object Notation). Users can create different multiple view layouts, associate specific visualisation types with a view panel, and change the appearance of the final multiple view layout. In addition, they can control the layouts through bespoke grammar in JSON. They can choose default layouts, design their layout, go back-and-forth between grammar and visual interface, and factor and re-factor their view layouts to allow, for example, an “n*m” and swap it to a “m*n”.

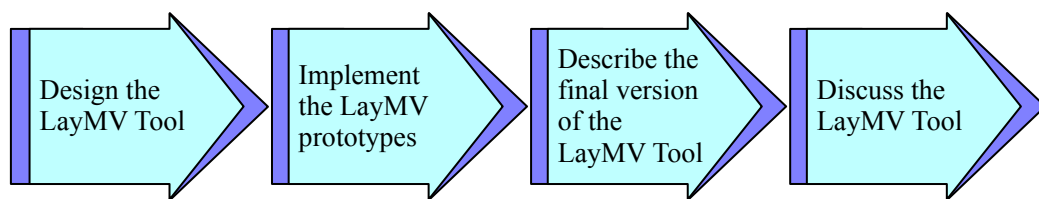


Figure 6.1: This diagram illustrates the stages to build the LayMV tool, started with the design of the tool, the implementation, Describe the LayMV Tool, and finally a discussion about the LayMV tool.

The chapter contains three main sections: the *design*, *implementation* and *results* of developing LayMV, followed by the Discussion and Conclusions. Figure 6.1 shown

different stages of building a multiple view tool. This chapter is structured around these stages, as follows:

- Section 6.2: The **design** of the LayMV Tool section, explaining the design process and the reasons and the purposes from building the LayMV tool. In addition, we explain the design methodology we followed to design the tool and what decisions that we made and the alternatives of each decision.
- Section 6.3: This section on the LayMV Tool's **implementation** discusses how we built the tool, our rationale for developing it, and our consideration of alternatives.
- Section 6.4: The **results** section describes LayMV's layout, clarifies the LayMV's functionality and demonstrates how to use MV's grammar to create multiple view layouts.
- Section 6.5: Finally, the **discussion** section discusses the design process for LayMV tool and the benefits from using the MVG grammar, including the advantages and the limitations of the tool.

6.2 Design of the LayMV tool

This section provides a description of the design methodology. The methodology was to start by drafting the vision, and then refining the concepts through several prototype implementations. Because an incremental approach was used (with three main prototypes) this section focuses on the main design concepts to create the LayMV tool. Then the implementation goes into detail for each prototype. There are several stages to designing a tool that can be used to create multiple view layout. The following steps were followed to design LayMV tool:

- (Step 1) Explore alternative design methods and define a vision for the system.
- (Step 2) Sketch ideas for the LayMV layout.
- (Step 3) Determine the LayMV tool functions and the main components.
- (Step 4) Incrementally improve the LayMV design through analysing the LayMV layout and the LayMV functions to detect design errors, which required building different prototype versions.

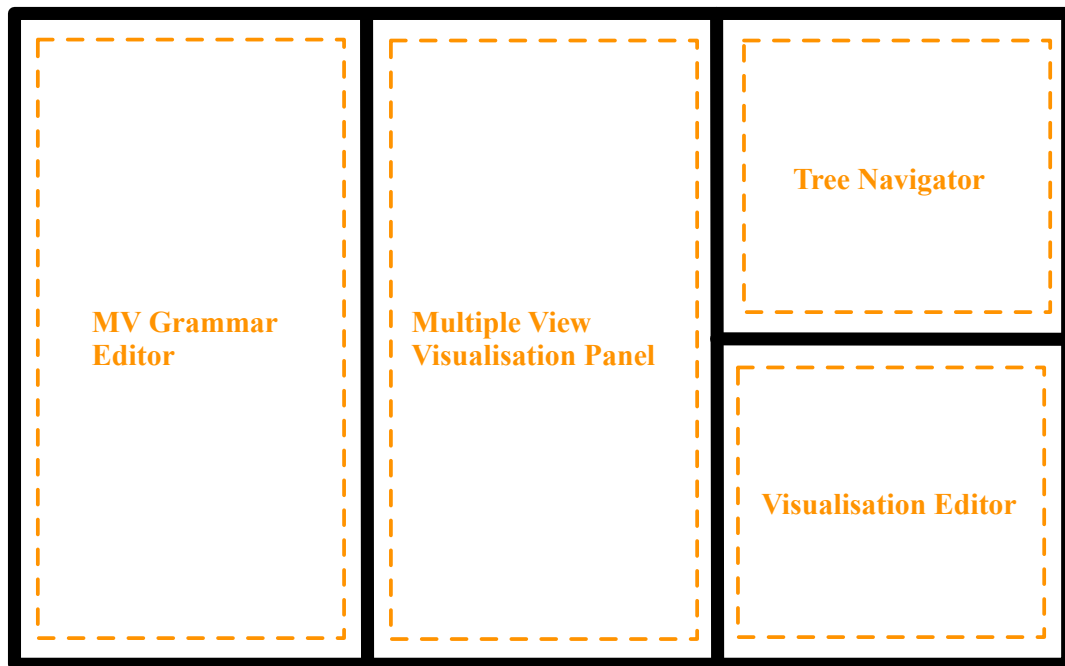


Figure 6.2: The wireframe layout of LayMV tool, showing four main views: the grammar editor, visual panel to control the layout visually, and tree-navigator (to select different parts of the layout code) and the visualisation editor (to allow users to add in their own visualisation code).

The Wilson’s method of heuristic evaluation [153] was followed, which was used to examine each prototype. Three developers were involved (who were separate to this project); two with visualisation expertise and one a generalist software engineer to provide think-aloud comments, we developed three major prototypes.

The design study investigated general ideas of how the system would work and appear. The goal of this design study was to determine the appearance and main functionality of the tool. The Five Design-Sheet (FDS) method was chosen because it was familiar. While there are alternatives, such as using ad hoc sketching [24]. However, the FDS method gives a very defined set of instructions to follow that will create not only lots of the different design ideas, but three focused design concepts and then a final design idea. This process helps us to expand the ideas and consider alternative ideas in great detail. In addition, through the design process the concepts that have been evaluated in Chapter 4 can be integrated with the designs, and we apply the design guidelines from Section 4.9. After the sketches then a wire frames model were created [128].

As shown in Figure 6.2, the design of the LayMV fits with our design guidelines (see Section 4.9) as the LayMV has simple layout structure with four views and a vertical Symmetrical balance.

- (View 1) MVG grammar editor, this view will be the place where the user can write and edit the grammar expression to create multiple view visualisation.
- (View 2) Multiple view visualisation panel, this view will be the place where the user can see and control the created multiple view visualisation.
- (View 3) Tree navigator, this view will be used by the user to choose a specific view from the multiple view layout to add a data and a visualisation technique to it through the fourth view.
- (View 4) Visualisation editor, this view will be the place where the user can add data and visualisation technique to each view, moreover, this view will control the appearance of the created multiple view visualisation.

Using these Low-fidelity designs, we got the experts to reflect on the proposed ideas (**Step 4**). While this was an informal session, we used Wilson's method of heuristic evaluation [153] as a guideline for analysis.

The feedback from the experts was positive. They liked the many-view system that we proposed. They were positive about the idea of a layout grammar. One suggested that we could consider using Vega-lite [127], which extends Leland Wilkinson's Grammar of Graphics [152]. However after our discussion, the expert agreed that Vega-lite was not suitable for our purpose.

One of the main design goals was to create a visual editor (named Multiple View Visualisation component, shown in Figure 6.3). There are different ways this could be achieved and implemented. For instance, the layout of the views themselves could be controlled by the data. This is similar to `ggplot` and `gridBagLayout` in Java, where the developer says that they want a view to the left, to the right, and so on, and the system places the views in their actual position. Some views could be snapped next to others. For instance, there are many examples of tiled views (especially on mobile devices) where the user can drag a widget into place, and it gets snapped close to its neighbour widget. They are automatically aligned. Alternatively, the user can

take complete control. For instance, in a drawing program, the user can control the size of the rectangle, and use split, or merge commands to split this rectangle in two parts, and so on. In this way, we posed the question “who controls the layout of the views – the designer and user, or the system in an automatic way” [82]. The automatic layout mechanisms can be considered as a ‘geometric object packing’ [75]. Where objects (views in our instance) are placed side-by-side in a grid or layout that does not leave any gaps. There are also similarities to other visualisation designs. For example, treemaps [134] pack the data in hierarchies; a slice-and-dice algorithm is used to pack the data hierarchy. Small multiples layout visualisations in a grid [78], spreadsheet visualisations [30] organise the small visualisations in a regular lattice, and the view bracketing concept of Roberts [115] places visualisations in threes.

Following, it was decided to design a system with some automatic snapping. This would allow a designer to align objects suitably. Systems look untidy if the windows are slightly misaligned. However, we also want designers to craft multiple-view systems from their imagination and under their control. Consequently we decided that the system should also allow users to position individual parts by hand. For these reasons, and similar to packing problems, it was realised that four **challenges** need to be solved:

- First, a way to define the **space** of the layouts, to allow users to position views side-by-side, on top, or to the left of other views.
- Second to define the **content** of that *space*. In other words, what is required is a way for a user to define the grammar that will describe the visualisation contained within that space.
- Third, there needs to be a way for the user to be able to define the **appearance** of the windows, frames or spaces that the visualisations sit within. What is required is a way to define: window spacing, background colour, frame colour, whether the frame can be moved, etc. For instance, each multiple view part may be surrounded by a blue frame, or have no visible frame at all.
- Fourth, there needs to be a way for the user to load **data** and visualise it. Furthermore, the data needs to be mapped to each visualisation. Perhaps every visualisation uses the same data, or some windows use one data, and others use another.

These design requirements led to a multi-part solution, with several component parts (step 3). The implementation was divided into components that (A) deal with the visual front end, and (B) those that pertain to the underlying functionality. Through discussion with the experts, it was decided to split the system into eight main components. Figure 6.3 shows a schematic picture of this model. This model was useful in communication and conversation. It meant that intermediate prototypes could be created, which contain some components, get feedback from the experts, and then develop the others in turn.

- 1) **LayMV layout**. This module controls the visual visual components of the LayMV tool.
- 2) **MVG grammar** is the MVG grammar editor, where users can type new grammar and edit existing text.
- 3) **Multiple View Visualisation** is the code that controls the main system.
- 4) **Appearance** allows users to control the appearance of the layouts that are created through the LayMV system.
- 5) **MVG grammar checker** is the checker algorithm that allows the code to be checked and validated. It means that validated code will be used to present the layout in the visual (Multiple View Visualisation) component. This is a two-way link, meaning that changes to the grammar alter the layout view, or changes to the layout view alter the grammar code.
- 6) **MVG Grammar parser**, the algorithm that should be used to convert the grammar expression into a multiple view layout. On loading, or on a change to the layout, the grammar is parsed.
- 7) **Save/ Load** component enables the project to be saved and loaded. On load, grammar is parsed and checked.
- 8) **Multiple view visualisation algorithms**, these are the algorithms that should be used to visualise the data and to create the multiple view visualisation.

LayMV Tool Components

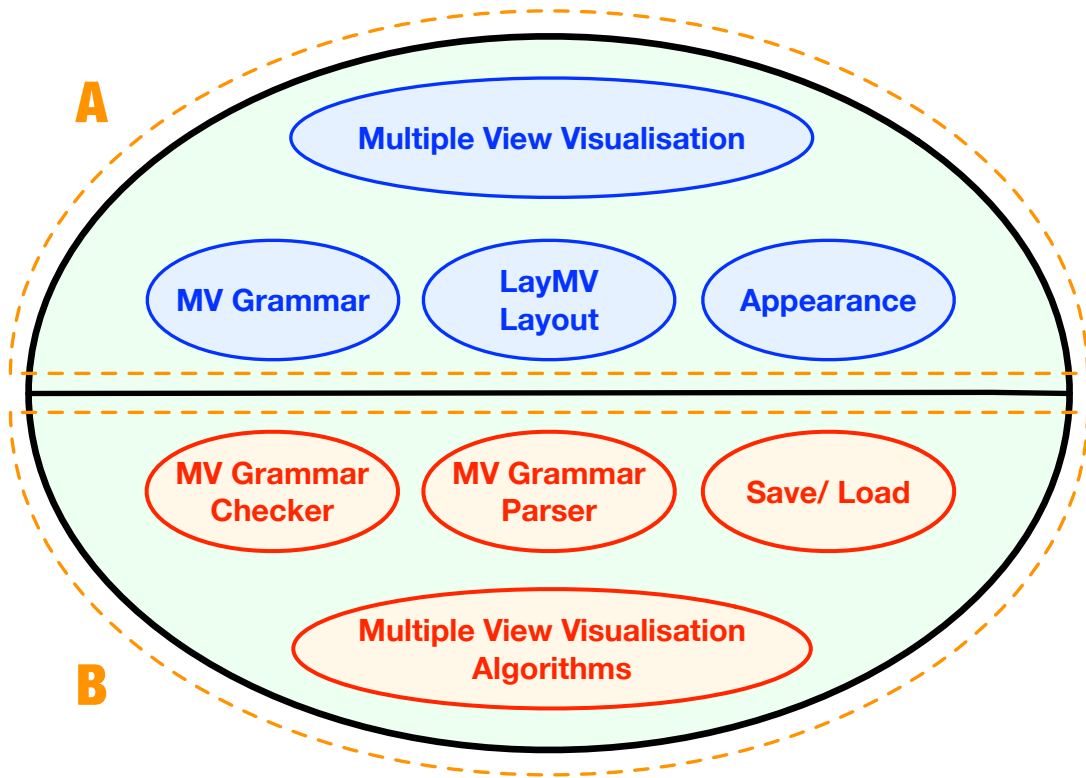


Figure 6.3: This diagram describes the LayMV tool and illustrates the main components and functions, these are the main components of the system, where “A” represented the visual components and “B” underneath the line represents the hidden (algorithms) components.

6.3 Implementation of the LayMV tool

The process of implementing the LayMV tool went through several prototypes. Starting with the components in the design stage (Section 6.2). Second, the first prototype had basic information that was tested with experts. Third, the prototype was improved. During this process the LayMV design was improved through analysing the LayMV layout and the LayMV functionality to detect design errors and problems in each LayMV prototypes, and then redesigned and rebuilt to develop a new enhanced version of the tool based on the evaluation of the previous version. The improvement process repeated until the final design for the LayMV tool. For communication purposes this whole process can be summarised by three principal prototypes. The final step is where users

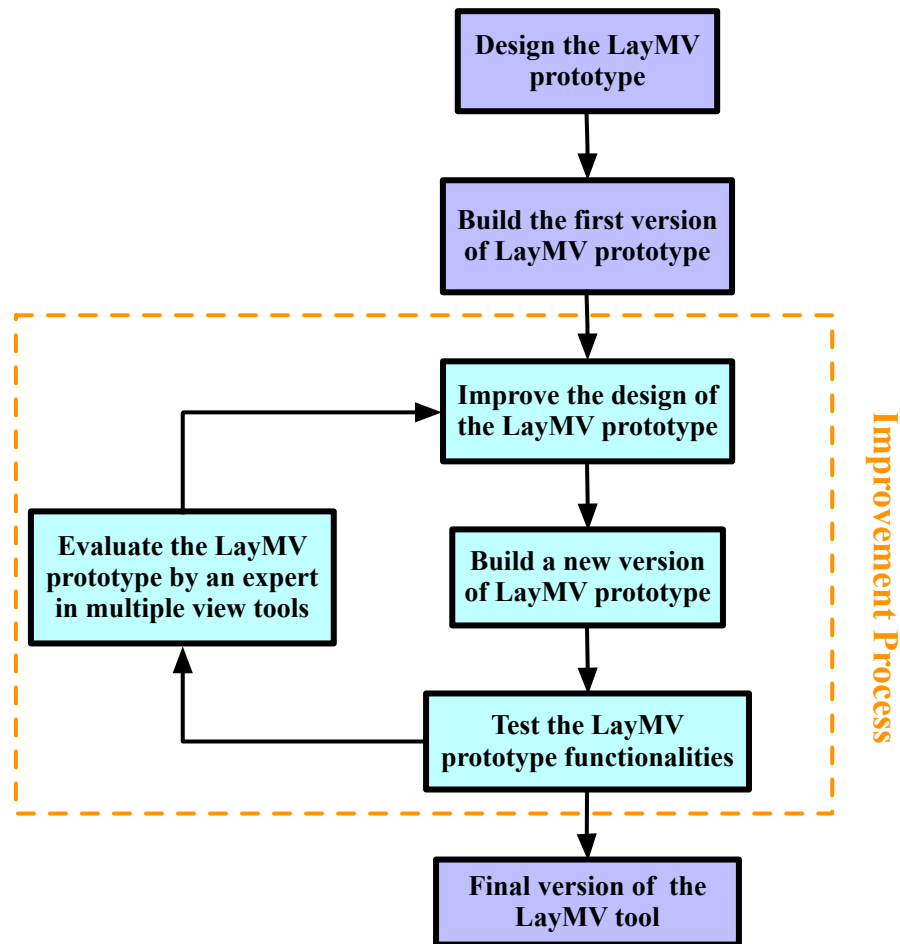


Figure 6.4: This diagram describes the implementation of LayMV tool, we asked experts in multiple view tools to use the LayMV prototypes and give feedback for each version of the tool.

do a end-user-evaluation for the tool. Figure 6.4 illustrates the LayMV implementation process.

For the implementation a Web design environment was chosen. This utilised: HTML, JavaScript and JavaScript Object Notation (JSON) to build the LayMV tool. The motivation for this choice is to make the overall system able to be used remotely, as well as to be visualisation agnostic. Typically visualisation systems are built with single libraries. However, one of the goals of this research is to create a structure (the grammar) which has the potential to be used by other systems. Certainly, the software could have been built using different libraries and tools. Processing.org could be used to create the visualisations, but the goal was to focus on the layout and not develop additional bespoke visualisations. D3.js would be another obvious choice, but through

discussion with the experts, it was decided that while D3 would provide the visualisation components it was less suitable for the task of developing the tool. The decision was therefore taken to use HTML with JavaScript, which would mean that the prototype could work on any browser. Then we could allow visualisation libraries (such as D3) to be integrated with the tool. It would mean that we could create a system that hopefully could be used for D3, Shiny-R and other visualisation libraries.

The first prototype (a two-view system with three panels), shown in Figure 6.5. The left panel is the layout editor, and the right panel shows the rendered output. The middle panel contains a button to “render” the output. We use the slice-and-dice methodology, to draw the view layouts. On the editor view the user can cut a view in half, and subsequently cut each other view in two. These early prototypes helped us understand how we should lay the views to create a layout in one of the LayMV views and how we render the same layout in another view.

The feedback from the expert visualisation designers was positive for this first prototype. They realised that it was in the early phases of the developments, but were very encouraging. But the experts found some configurations tedious to create (such as 5 by 5 view). Certainly, with the experts and supervisor, we had been discussing, different ways to make the design more efficient. Subsequently, from the feedback of this group, methods to make (e.g., 5x5 view) views less tedious were investigated. The expert group also noticed that sometimes they believed it to be easier to craft view layouts by hand, and other times through commands. This exactly was the vision; that some configurations would be easier in code and others to do on screen, and their feedback was positive encouragement that the development was on track.

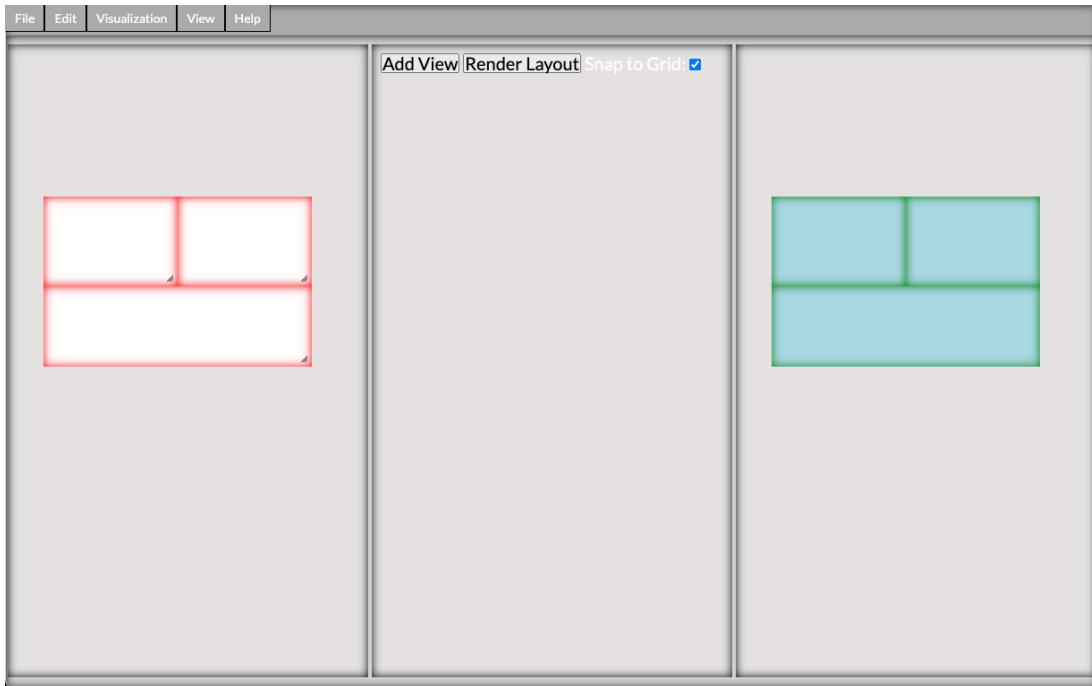


Figure 6.5: A snapshot for an early version of the LayMV tool, the “**Add View**” button adds a view to the first left view of the LayMV tool each time we clicked on it. In addition, we can use the mouse cursor to drag and drop the views to move them around and build a multiple view layout; Then, by clicking on the “**Render Layout**” button, we can render the same layout in the third view of the tool.

The second prototype added more components. The design study (Step 3) described a multiple-window design, and so this development stage started to implement this idea, and make the system more complete, as shown in Figure 6.3. Prototype-2 contained some menus and control buttons (top left), the grammar editor (lower left) and right the layout editor. The first major improvement on the previous versions of the LayMV tool was done through adding the MVG Grammar editor to create the multiple view layout, as shown Picture 6.6; these grammar can be edited to modify the multiple view layout. In addition, this prototype included the implementation of some summary grammar expressions, which allow users to quickly explain views through grammar commands. The control panel enabled the user to match the grammar to the editor, or the other way round.

Again the experts were asked to comment on the design, and feedback on how the LayMV prototypes can be improved. In addition, they were asked to suggest solutions

for problems that were faced during the implementation process. And they gave helpful suggestions of how to succeed in the development of the interactions and lay out of the menus. At this stage they were keen to have the tool completed. They mentioned the need to ‘error check’ the input and output files. Saying that, as developers themselves they would want to edit the file outside the application, and so parsing and error checking was important. They also commented on the fluidity of the commands, suggesting that dynamic editing would be possible, where the grammar is automatically updated from the design, or the other way round. The experts also emphasised the need to keep the three different parts of the design separate. This would allow the user to focus on a specific part of the layout design:

- 1) Grammar Editor, to create and control a multiple view layout.
- 2) Visualisation Editor, to upload data, map the data and create and control the visualisations inside the layout.
- 3) Layout Panel, where the layout will be placed.

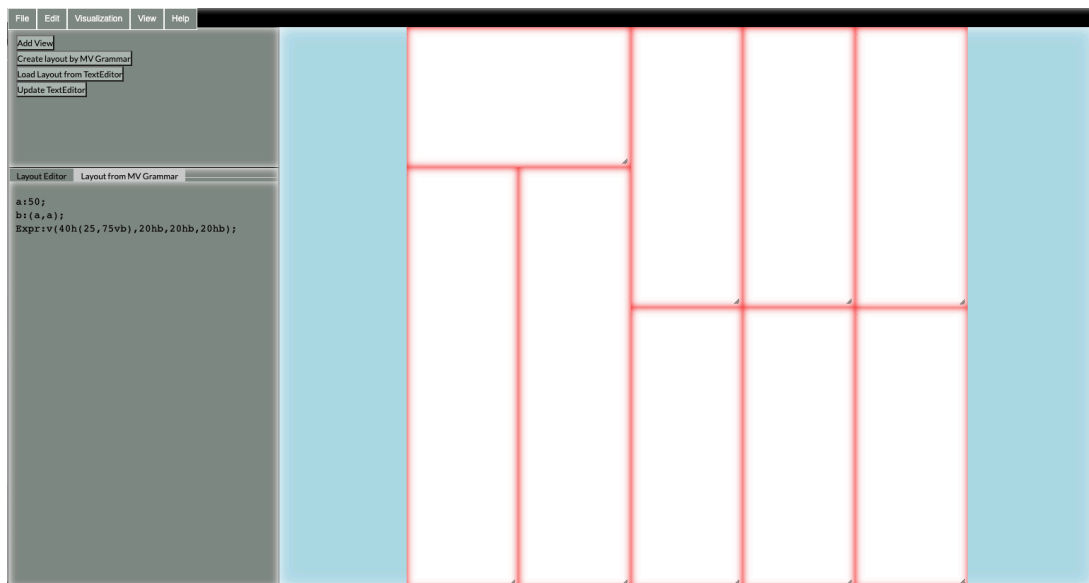


Figure 6.6: A snapshot for an updated version of the LayMV tool, the “**Create layout by MVG Grammar**” button created the multiple view layout by using the MVG grammar “**a:50; b:(a,a); Expr:v(40h(25,75vb),20hb,20hb,20hb);**”.

The final prototype included each of the component parts, along with the parser, file save and load, and grammar checker (as per the components described in Figure 6.3).

The final prototype is shown in Figure 6.7. In addition, a tree navigator was added. This helps the user add data, visualisation and appearance to a specific view in the multiple view layout, and edit appearance of the layout itself. Furthermore, a shortcut grammar editor was added to give the user the ability to quickly create particular layouts efficiently (such as Golden Ratio layout).

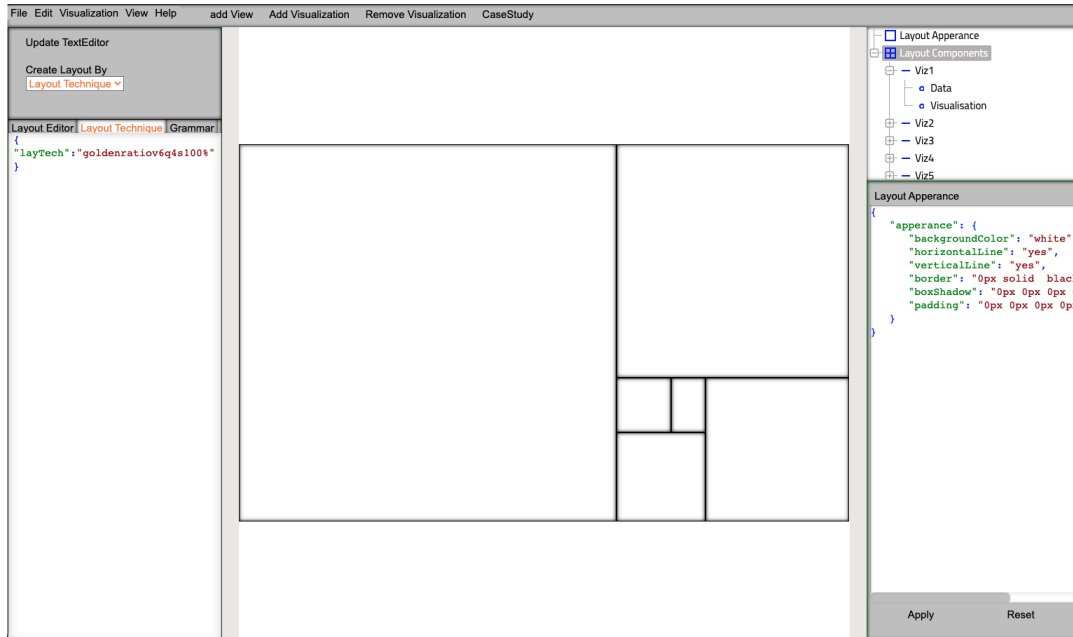


Figure 6.7: A snapshot for the LayMV tool after we added the tree navigator view and the shortcut grammar (Layout technique editor), the expression “goldenratio64s100%” in the Layout technique editor created the above golden ratio layout.

As considered in Section 6.2, the final layout of the LayMV tool has **four views**, which are three main views divided horizontally, where the third view divided into two views vertically. The four views from the left are:

- **First view (V1)** has three tabs; we named them as “Grammar Tabs”. The first tab is the “Layout Editor”, used to find and edit the details of the created multiple view layout, such as the position and the size of each view. The second tab is the “Layout Technique” where the user can use the shortcut grammar to create special multiple view layouts. The third tab is called “Grammar” where the user can use the MVG grammar to create a multiple view layout.
- **Second view (V2)** is used as a multiple view panel where the user can create and

modify the multiple view visualisation visually by using the mouse cursor, where the user can resize the views and change its positions.


- **Third view(V3)** is the tree navigator, which the user can use to navigate the multiple view layout and select a specific view to add visualisation to the chosen view through the fourth view. In addition, the user can choose “Layout Appearance” to edit the overall appearance of the multiple view layout.
- **Fourth view (V4)** is used to add data and visualisation techniques to any view in the multiple view layout; first, where the user can select a view by the “tree navigator”, then the user can add visualisation to the chosen view. In addition, users can change the overall appearance through this view.

In addition, besides the visual components of the LayMV layout, the LayMV tool has underpinning algorithms that responsible for performing the tasks and the functions of the LayMV tool and ensuring its correct functioning. The main algorithms are:

- 1) MVG Grammar checker, this algorithm checks that the MVG grammar expressions are legal and fit with the MVG grammar parser requirement.
- 2) MVG grammar parser, this algorithm converts the MVG grammar expressions into visual multiple view layouts.
- 3) Views synchroniser, this algorithm responsible for the synchronisation between LayMV views.
- 4) State saver, this algorithm allows the user of the LayMV to save all the details of the multiple view layout.
- 5) Visualisation creator, this algorithm adds visualisations to the multiple view layout.

MVG grammar checker algorithm makes sure that the MVG grammar expression is writing accurately. Suppose the user did not write the MVG grammar expression in the correct form. In that case, the checker algorithm will prevent sending the MVG expression to the MVG grammar parser, eliminating errors in the MVG grammar parser. In order to have a successful MVG grammar expressions, the expressions should be written in the following format:

- 1) The expression starts with **Expr:** and end with a semicolon “;”.

- 2) The cut operations divide a view (or the whole layout) into two or more areas (views), and the user should put a comma “,” between each two areas. In away, the vertical cut “v” and the horizontal cut “h” should be followed by the “(” bracket and end with the “)” bracket, and the user should define the percentage of the size of each new view from the parent view and put it between these two bracket, where the size summation of the all new views equal to “100”. In addition, there are spacial ways to define the areas for the new views, such as:
 - A) **h()**, this expression will divided the layout into two horizontal views.
 - B) **v(,,)**, this expression will divided the layout into three vertical views.
 - C) **v(50,,)**, this expression will divided the layout into three vertical views, where the area of the first view will be 50 percent from the parent view and other two views will have share the rest of the area equally.
- 3) The user of the LayMV tool can define variables to simplify the complex expressions; in this case, the grammar expressions will have more than one line. E.g., the first line is **a:(50,50)**; and the second line is **Expr:v(50ha,50)**;; this grammar expression will create the layout . Furthermore, the user should put a semicolon “;” each time he defines a variable.

In addition, the grammar checker examines the shortcut MVG grammar to be sure that the expression is written in the right form. As the shortcut grammar create ether grid layouts or golden ratio layouts, the expression should follow the following structure:

- 1) The shortcut MVG grammar expression should be put between the “{” bracket, and the “}” bracket.
- 2) The expression should start with "**layTech**" then followed by **:** and ending with shortcut expression such as "**grid3*3**" to create 3 by 3 grid. In this case the whole shortcut expression should be written as "**layTech**": "**grid3*3**", see Section 5.4.

Next, if the MVG grammar expression passed the grammar checker successfully, then it will by moved to the **MVG grammar parser** algorithm which will convert the expression to a visual multiple view layout.

The **views synchroniser** algorithm synchronises events between LayMV components. It makes sure that all the views of the LayMV tool are up-to-date, and the information

is harmonious across each view. It also guarantees that all the views of the LayMV tool have non-contradictory details about the multiple view layout, and these views are not conflicting and fight with each other. Furthermore, all these views are connected and any change in the data that belong to a view will affect the data of the rest views. The user can make a manual synchronisation using **Update TextEditor** button, e.g., if the user create a layout using the MVG grammar in the grammar tab, then the user needs click on the **Update TextEditor** button to update the contain of the layout editor tab so that the layout editor will be described the created layout.

Finally, a user can use the tree navigator to add data and associate different visualisations to each view in the multiple view layout, to build the overall multiple-visualisation system. Moreover, the LayMV tool can save the project's detail as a text file this will allows the user to save the state of the multiple view project. This also includes the layout, the visualisation techniques and how views share the same data or not. A possible drag-and-drop operation was discussed and trialed, however this method did not work well. It was difficult to control where the data was going to go, and difficult to control the window placement. The idea could be used to add data or a visualisation to one panel, and then drag and copy it to other parts. However, instead it was decided to use a tree view to control this functionality. This tree view means that the data is explicitly described, that it can readily be saved into the file, and users clearly understand how to add data to specific multiple views.

6.4 Results

The LayMV tool allows users to design, create and modify a multiple view layout visually or by use the MV grammar. In addition, this tool allows users to save, reload and edit the MVG grammar. Also, the LayMV tool do grammar checking, and highlight the grammar that needs to be corrected. The LayMV tool included the ability to choose pre-designed layouts and add visualisations to the views, in addition to a tree navigator that helps the user to add data, visualisation and appearance to a specific view in the multiple view layout.

This section focuses on the final version of the LayMV tool and what the tool looks

like, and its functionality. Screen shots are included, with explanation of the different components, to explain the main functions. This section records the final prototype.

LayMV tool Description: The final prototype has two screens, each addressing a different task, the first screen is the “templates viewer” and the second screen is the “grammar viewer”.

LayMV tool provides layout templates on the first screen, the first screen has one main view which called the templates viewer, as shown in Figure 6.8, and this screen is the default first screen and allows a pre-built design to be selected. The tool provides the ability for users to filter the layouts, where users can search and filter designs. After selecting a starting template, which could be a blank layout, the second screen will be opened.

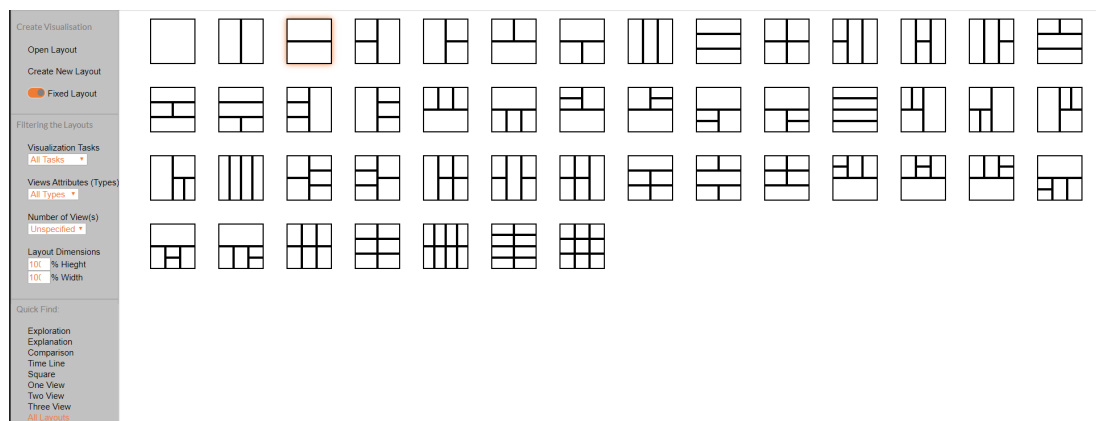


Figure 6.8: Template viewer. Users can choose a starting layout, search for a specific view quantity, various multiple view Layouts can be created with different views number, and the user can later edit the template.

The second screen has four main views, as shown in Figure 6.9, the grammar view (the “Grammar Tabs”), the visualisation editor, the tree navigator view and the property view.

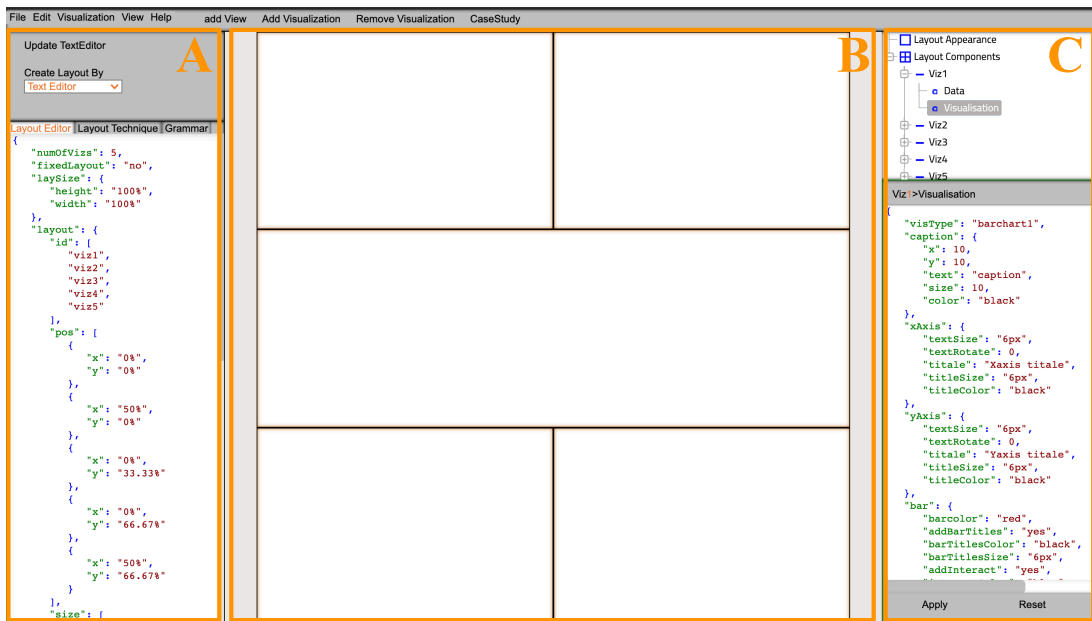


Figure 6.9: LayMV tool, with three linked views. (A) grammar panel to edit the grammar (either shorthand e.g., v(50,50) or full JSON, shown), (B) Visualisation panel of either the wireframe editor or layout editor (shown) and (C) property panel.


The grammar view, as shown in Figure 6.9 A, has three tabbed parts. The first tab allows the “Layout Editor” to be shown. In this window users can edit grammar that describes the position and the size of each view. The second tab shows the “Layout Technique” editor. This allows users to use the MVG shortcut grammar to used to create a multiple view layout. The fourth tab “Grammar” allows users to use the MVG grammar to create a multiple view layout


The visualisation Panel, as shown in Figure 6.9 B, shows either a wireframe layout editor (without visualisations), or the final visualisation view. In this picture, Figure 6.9 B, the final visualisation of two bar charts are shown. The wireframe editor allows users to add, delete and change the size and the position of the views, and snap wireframe views together. The grammar description is dynamically updated on the left panel, allowing users to jump between grammar or layout descriptions.

The property panel, as shown in Figure 6.9 C, has two panels, the top panel is the “Tree navigator” which allows users navigate the layout to add data and visualisation techniques. The bottom panel is the “property view” which allows users to change the appearance of the layout (border width, background colour, etc.) and how the

visualisations and data are mapped to panels. Different visualisation libraries could be embedded, the example in Figure 6.9 uses D3.

LayMV tool functionality: LayMV tool allows user to create and control multiple view visualisations by doing the following:

1. Create a new layout either by choosing a layout template from first screen, or using the mouse cursor by clicking on the “Add View” button in the second screen to add a new view and use the mouse cursor to resize and place the view in the right position.
2. Also, the user can change the parameters in the layout editor or by using the mouse cursor.
3. By using the MVG grammar a user can create a multiple view layout, the basic idea is based on hierarchical cuts. Cut one view horizontally (h) or vertically (v) to produce two views, and so on. E.g., $h(50,50)$ creates an equal sized side-by-side view . The value 50 is representative, $h(20,20)$ would provide the same result.

Moreover, complex cuts can be easily created, e.g., $h(50v(75,25),50)$ creates layout with three views, a long bottom view, with the top split 75% across . We can control quantities, make variables, define prototype layouts in the grammar. E.g., $a:50; b:(a,a); Expr:h(30vb, 40hb, 30vb)$ creates a layout with six views. We can quickly define a nine-grid view (`"laytech": "Grid3*3"`) or place six views in a golden ratio with a centre in the fourth quarter (`"laytech": "GoldenRatioV6Q4"`). A full description of the grammar is not possible here, due to space constraints. The JSON can be saved, and reloaded, and is checked for errors on load.

First, the grammar will produce two horizontal views with the same size percentage from the layout, then the top view (parent view) will be divided into two vertical views, the size for the first vertical view from the left will be 75% of the parent view size and the size for the second vertical view will be 25% of the parent view. With the experts feedback, much thought and effort was made, on how the data will be stored to define the structure of the layout within JSON, and make it easier to control specific view arrangements. Quantities can be controlled, users make variables, define prototype layouts in the grammar. E.g., $a:50; b:(a,a); Expr:h(30vb, 40hb, 30vb)$

creates a layout with six views. For instance, users can quickly define a nine-grid view ("*laytech*":"*Grid3*3*") or place six views in a golden ratio with a centre in the fourth quarter ("*laytech*":"*GoldenRatioV6Q4*"), more description will be in Chapter 7. The JSON can be saved, and reloaded, and is parsed and checked for errors on load.

In order to have a successful MVG grammar expression, if the user of the LayMV tool did not write the MVG grammar expression in the right format, the MVG grammar checker would indicate an error in the expression format and the colour of the expression text will be changed from black to red. In this case, the MVG Grammar checker will not pass the expression to the MVG grammar parser.

Continuing on the functionality, LayMV allows users to:

4. Upload data files to be related with its views.
5. Add and edit visualisation technique for each view by using the visualisation editor.
6. Use the tree map navigator to navigate the visualisation components (the views and its content).
7. Save and reload the visualisation project (the layout, the data sets and the visualisation techniques).

Figure 6.10 shows the LayMV tool created a multiple view layout using the a MVG grammar.

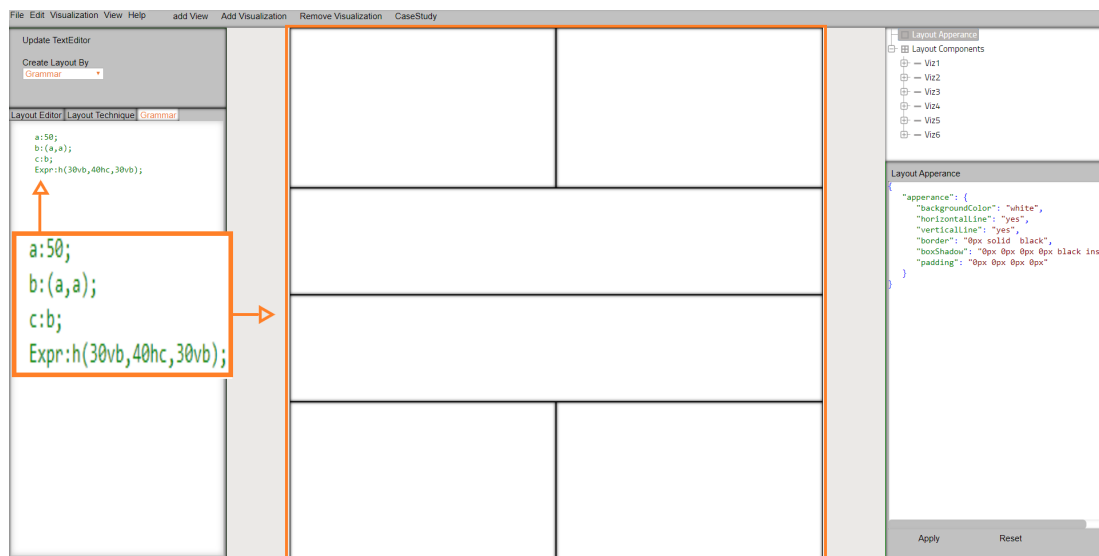


Figure 6.10: Using MVG grammar to create multiple view Layouts.

In addition, Figure 6.11 shows how user can create multiple view Layouts using a the MVG shortcut grammar.

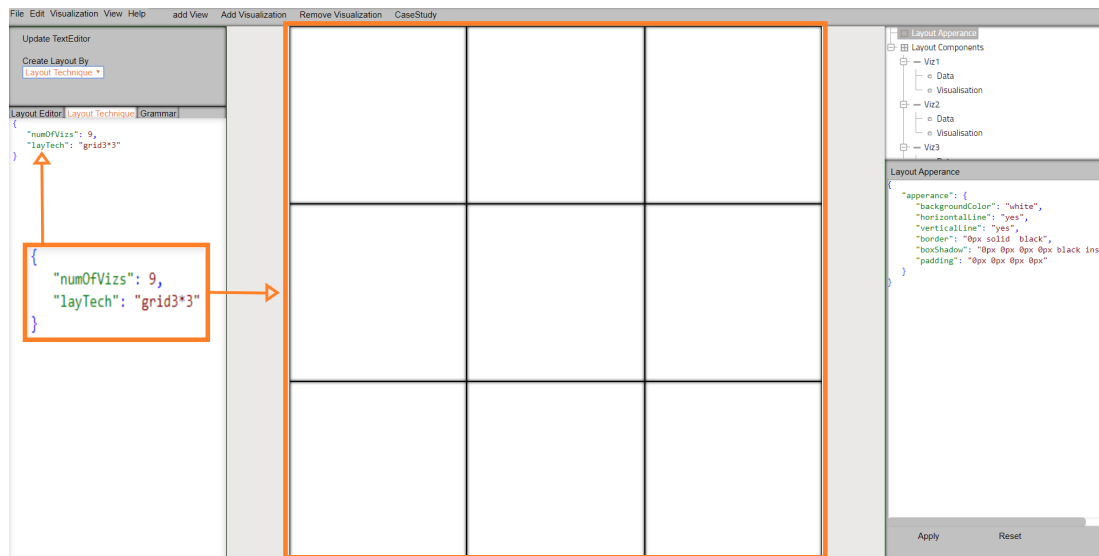


Figure 6.11: Using LayMV Layout Technique to create multiple view Layouts.

6.5 Discussion

There are many foundational parts to the thesis that have led to the development of the LayMV tool. First, the related work chapter described many different multiple view visualisation systems, and the work discussed different multiple view visualisations techniques, such as: design concepts, multiple view systems and tools, and theories and design guidelines for multiple views. Second, the quantification Chapter (Chapter 4) provides insight into how people use multiple views. The analysis of multiple view visualisations helped to develop and create a new set of design guidelines for multiple view visualisations. From this analysis people are using few views but also developers also do create many viewed systems. Consequently there needs to be a way for people to create multi-viewed systems, and create them in a simple way. Third, the grammar chapter (Chapter 5) provides the underpinning principles that lead to the file structure of LayMC. It provides a convenient way to, not only store view layouts, but edit them (outside of the LayMV system), and load them between sessions.

It is clear to see that developers have many layout techniques which can be used for placing the visualization on the screen. However, the user which is used the visualisation tool is forced to use the method specified by the developer without the ability to change the visualization layout, to be able to see the data effectively. This gap can be reduced by

given the users the control to change the layout to improve the display of the visualization. In this case, there will be two challenges:

- Auto layout method which has to be determined at the first appearance for the visualization.
- Give the possibility for the user to save the layout and apply it on another data.

This chapter presented the “Lay multiple view” (LayMV) tool based on an in-depth analysis study, we have done on multiple view layouts, this tool helps users build, control and save multiple view visualisations simply and easily using a bespoke grammar. The tool incorporates template multiple view layout strategies as quantified from prior research on view analysis, and the user can build different layouts by defining the grammar, or through the linked visual interface. In addition, the LayMV tool was designed from the design guidelines, that were introduced in this study, see Section 4.9.

LayMV tool guides the user to (i) design and control the multiple view layout, (ii) add data and allocate a specific visualisation technique for each view, and (iii) to adapt specific appearance properties of the layout.

After that, the user can assign data and a visualisation technique for each view so that he can use JSON file later to develop the multiple view visualisation by changing the layout or using different data sets or different visualisation techniques. In addition, much thought and effort on how the user will store the structure of the layout was given. With discussion with the expert group, the LayMV system was developed. This chapter provided details of the visualisations itself (project), and basic structure capture that within JSON file. In a way the JSON file is like a style sheet, where the LayMV tool allows users to save multiple view visualisations as JSON files, including all the details of the layout and attributes of the visualisations, which can be subsequently loaded and adapted or used to create dashboards.

There are limitations to the current implementation. It would be good to add more interaction and animation to the views. Users currently need to take the code, and extend it to define their own interaction. It would be good to add these commands to the Tree viewer. The data entry can also be fiddly, and this could be adapted by

adding a drag-and-drop function to the data and views, to drop data between views. Furthermore, different types of visualisations could be crafted. It would be good to have a comprehensive set of visualisations that, again, could be dragged and dropped into place. However, as we mentioned early in this thesis, the focus on our research was to develop tools around analysing and controlling layout. What LayMV does demonstrate is that the grammar works, and that it is possible to integrate it into a working multiple view editor.

6.6 Summary

This chapter focused on developing a tool that allows users to create and control multiple view visualisations. Consequently, this chapter answered five research questions: “Q1/ Why we want to develop the LayMV tool?”, “Q2/ What methodology should be used to design the LayMV tool? And, what alternatives are there?”, “Q3/ What process should be followed to implement the LayMV tool?”, “Q4/ What is the role of the MVG grammar in the LayMV tool?” and “Q5/ What methodologies could be used to evaluate the LayMV tool?”.

The chapter explains how the LayMV tool was designed and built. It allows users to lay out the multiple view visualisation. Users can select a template layout, or edit the layout visually or control the layout through the MVG grammar. The grammar is used to save/load view layouts, and each view is linked, such as when the user controls the wireframe editor, the grammar updates. The tool was developed through prototypes, with expert heuristic feedback. Its application was demonstrated with examples using D3, and it has the potential to be used with other visualisation languages and tools. LayMV is still being developed, and the plan is to provide a more in-depth user evaluation. But already the ongoing feedback from the experts, during the prototype development, has helped to create a working and suitable system.

The goals from building the LayMV tool have been to demonstrate the grammar and develop a working system, to help users manage views in multiple view systems. The work started with the five design sheets method, where the design decisions were discussed and alternative possible design solutions were discussed. Then, the tool development was explained, starting with early prototype versions, and from the expert

feedback the tool was developed, and its layout and the functionality of the LayMV tool improved. Subsequent, this chapter presented examples for how the LayMV tool creates multiple view visualisation to explain the tool's depth and functionality and show how the tool is work. Finally, this study described the LayMV tool layout, and the tool's functionality explained the goal and the job of each window and algorithm in the tool.

In this chapter, this study answered the second part of the research question “RQ8/ What is a multiple view grammar and **how can it be used in multiple view tools to create multiple view layouts?**”.

The next chapter will explain how LayMV can be used. Several case studies will be presented, that explain how a user can create different multiple view visualisations, highlighting the functionality of the tool.

Chapter 7

Case studies and discussion

This chapter shows and explains how the LayMV tool works to create multiple view layouts, and allow visualisations to be added to the layouts. This chapter will describe the application of the LayMV system that uses the grammar, as described in Chapter 5 and the design of the system, as described in Chapter 6. The chapter follows three designs, a 2-view system, a 3-view and a 8-view system. It adds D3.js code to the layouts, which demonstrate that visualisations can be added into the defined layouts.

The chapter focuses on demonstrating the LayMV system and the application of the grammar, and demonstrates the following research questions.

Q1/ How can users use the LayMV tool and the MVG grammar to create a multiple view layout?

Q2/ How can users start using the LayMV tool, and what is the process they need to go through?

7.1 Introduction

Developers of the software need to be able to create a multiple view layout. They need to be able to use the grammar, and be able to discover logical errors, bugs and limitations in the grammar. Also users need to create layouts quickly. We follow two case studies, to demonstrate how someone can use the LayMV tool. The first study creates a simple two-view system. The dual view layout is common, and while it is simple to create, the procedures followed in the examples demonstrate the principle ideas of the tool and grammar. The tool and grammar work together to create quick multiple view layouts. The examples herein, also demonstrate the possibilities and

limitations of the process, which will be discussed later in this chapter, and in future work in Chapter 8. Subsequently, we suggest a sequential process for developing views. Figure 7.1 illustrates the case study and creation process of a multiple view visualisation using the LayMV tool.

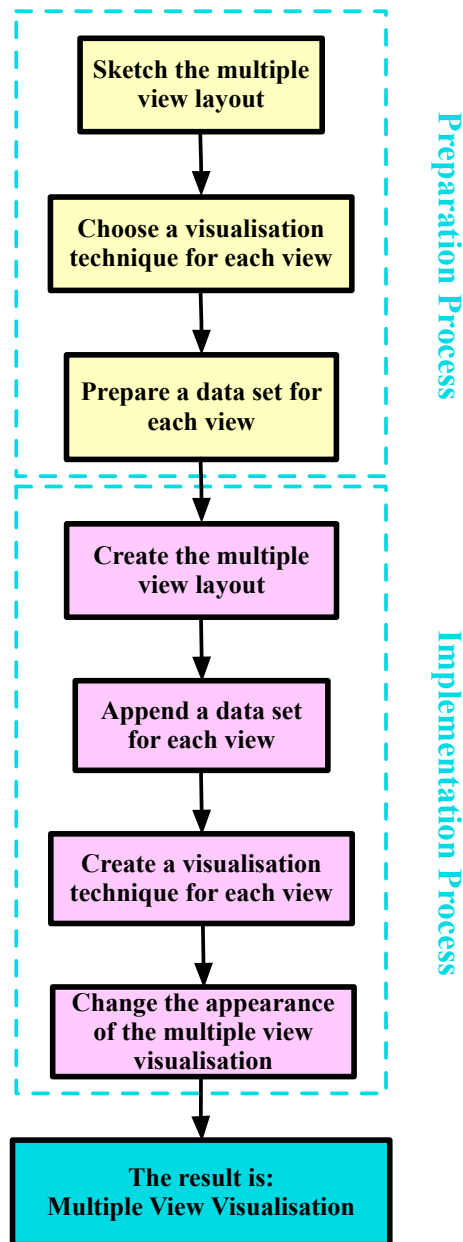


Figure 7.1: This diagram describes the steps that the user of the LayMV tool should follow to create a multiple view visualisation.

This chapter therefore contains three main sections: the *scenario* that required to be followed to create a multiple view visualisation, the *implementation process* to create the first and the second case studies, and followed by the *discussion of the results and Conclusions* of using the LayMV tool to create multiple view visualisation. This chapter is structured around these stages, as follows:

- Section 7.2: This section explains the preparation process and the scenario to create a create multiple view visualisation.
- Section 7.3: This section illustrates the the first case study.
- Section 7.4: This section illustrates the the second case study.
- Section 7.5: This section illustrates the the third case study.
- Section 7.6: this section discusses the limitation the results of case study one and two.

7.2 Preparation process and scenario to create multiple view visualisation

Figure 7.1 summarises the scenario and creation process that the user of the LayMV tool should follow to create a multiple view visualisation, the scenario includes the following steps:

- 1) Sketch a multiple view design.
- 2) Prepare a data set for each view.
- 3) Create a multiple view layout using the LayMV tool.
- 4) Edit the appearance of the multiple view layout.
- 5) Upload and assign the data set for each view.
- 6) Create a visualisation technique for each view in the layout.
- 7) Edit the appearance of the visualisation technique for each view.
- 8) Browse the result (the multiple view visualisation) or take a Screenshot for the multiple view visualisation.
- 9) Repeat any of the above steps to edit the multiple view visualisation.

When a user wants to create a multiple view, they need to have an idea (a vision) of what they want to create. The LayMV tool helps the user in this regard. Especially the first screen, that is shown to the user, is the Template view, where users can quickly choose a template, and then once chosen, can adapt the layout. Also, the user needs to consider what library they will use for their visualisation. They need to prepare their data, and understand the visualisation library. For these examples, D3.js will be used, but it would be possible to use other visualisation libraries.

The first stage is the preparation process, where the developer needs to prepare the required materials and consider their scenario, prepare their data. Then they implement their layout. Using LayMV and the grammar they can design a layout, append their data, create their visualisation in their chosen library, append it to the LayMV tool, and alter the appearance of the multiple views (by changing parameters of the frames of each view, background colours and so on). Finally they present their results. Users can go back to different stages and adapt. For example, they could go back to the layout editor, change the layout and then they will need to alter how the visualisations are loaded in each view.

At any stage, they can save the layout in a JSON file, saving whole project. When a project is reloaded, it is checked for compliance with the MVG grammar, before being loaded. The visual editor is updated appropriately. When the file is saved, everything will be saved including the data and the details of the visualisation types for each view, the layout appearance, along with the grammar.

7.3 First case study

When a user loads the tool, the first screen that is shown is the Template viewer. This is shown in Figure 7.2. The figure demonstrates two parts. In part A, the user can choose from a list of the top 52 layouts. These layouts were chosen to demonstrate the most popular versions. They are ordered by the quantity of views, and also included with normalised scales, in 50/50 proportions. Users can also search for a particular layout, using the search term.

Users can use these ‘templates’ to initiate the setup. The template loads default values into LayMV, places each of the views in the middle, and provides default values for all of the parameters and appearances. The user can change the layouts through the grammar at a later stage, or through the visual layout editor, and they can also change the proportions of the views. One of the main design inspirations for this template viewer was Microsoft WordTM interface. When the Word word-processor is loaded, it offers the user to choose from some templates. In this way a user can define the initial appearance of their document. Similarly, users of LayMV can start choose a template and start with a predetermined layout.

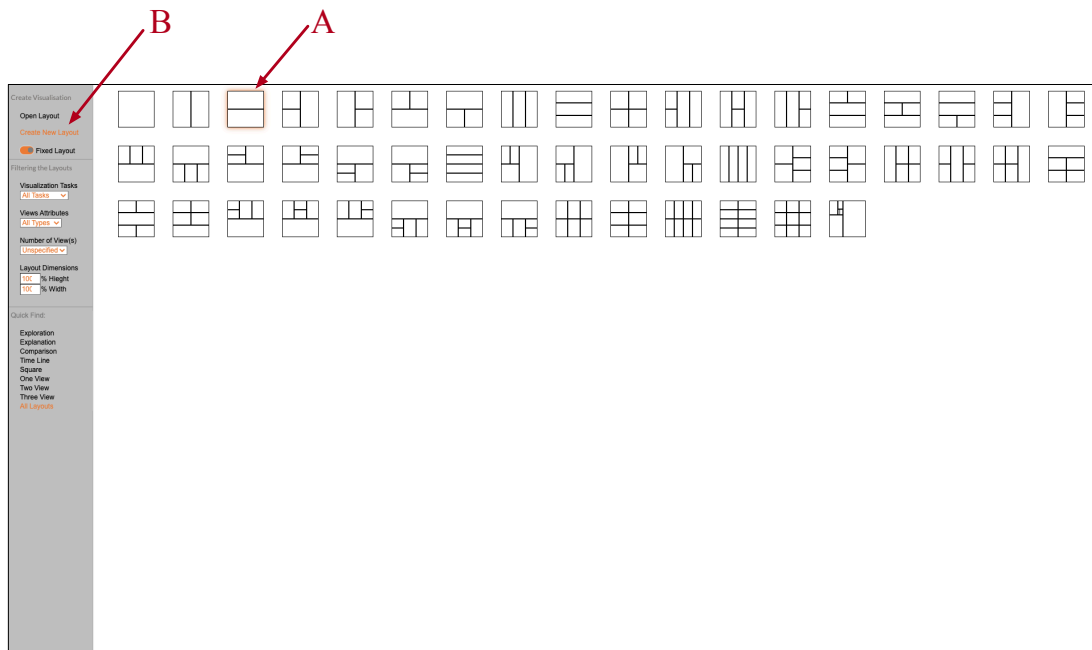


Figure 7.2: Template viewer. Users can choose a starting layout, search for a specific view quantity, various multiple view layouts can be created with different views number, and it can edit later in the second screen. Label “A” shows the multiple view layout that was chosen through a left mouse click, then we clicked on the “**Create New Layout**” button as shown by label “B”.

In this example the user has chosen a 2-view system, with a horizontal split. The result of choosing this template is shown in Figure 7.3. The user can adapt it, choose to edit the grammar, or to edit the shorthand code. They can change the appearance of the layout through defining the appearance of the stokes around each window, or the colour of each multiple view, and so on. Users can also add their own visualisation code. When the user changes the grammar, the editor window updates. Every view can be kept consistent with each other. So when the user changes the appearance of the view, they can update the main view, by selecting ‘update text editor’ and the information populates to the other views. It would be possible to make this automatic, however because the grammar needs to be checked, and that the grammar could come from the shorthand grammar (which needs to be parsed and interpreted) there are many different parts of information that need to be updated. Consequently, in this demonstrate we took the decision to make the updates manually controlled. This has the advantage that users can edit the grammar and when completed they can press ‘update’. On systems, such as Overleaf, that do have a live update mode, sometimes the update creates errors. Say, for instance, someone is writing $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ commands that have an open and closed curly bracket. If the system automatically renders when the user has only typed the open bracket, then an error will occur. Subsequently, we took the decision, in this demonstrator, and to have an ‘update’ button.

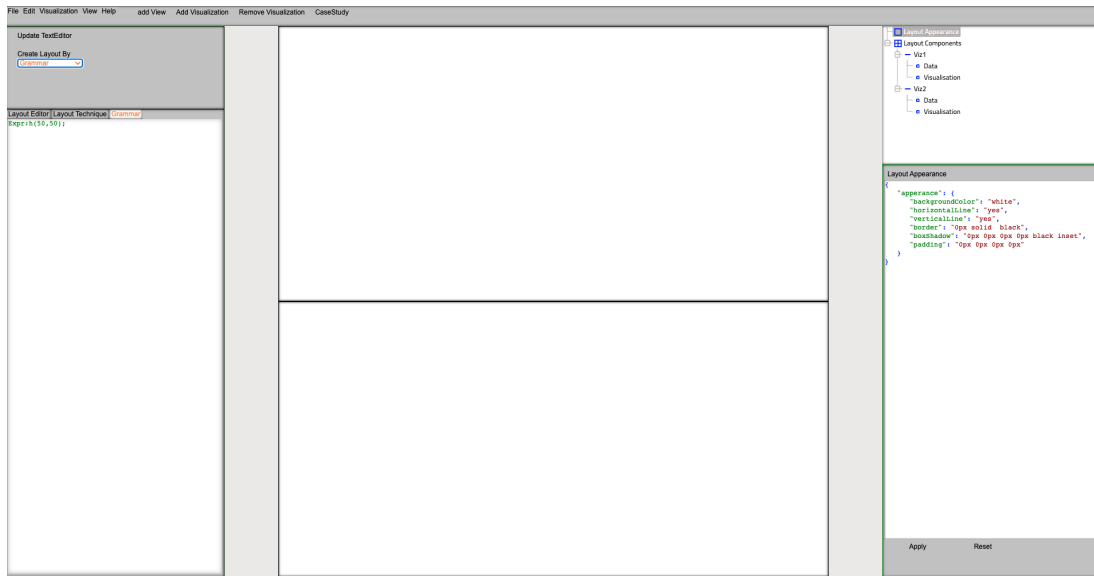


Figure 7.3: Main view. After choosing the template it is shown in the main view. There are three vertical panels. The left most panel controls the grammar, and has three tabs. The middle part is the visual editor, allowing users to ‘draw’ the views. The right most panel shows two views, which can control the appearance and allow users to add in data and visualisation code. Along the top, users can choose to save the project and load it.

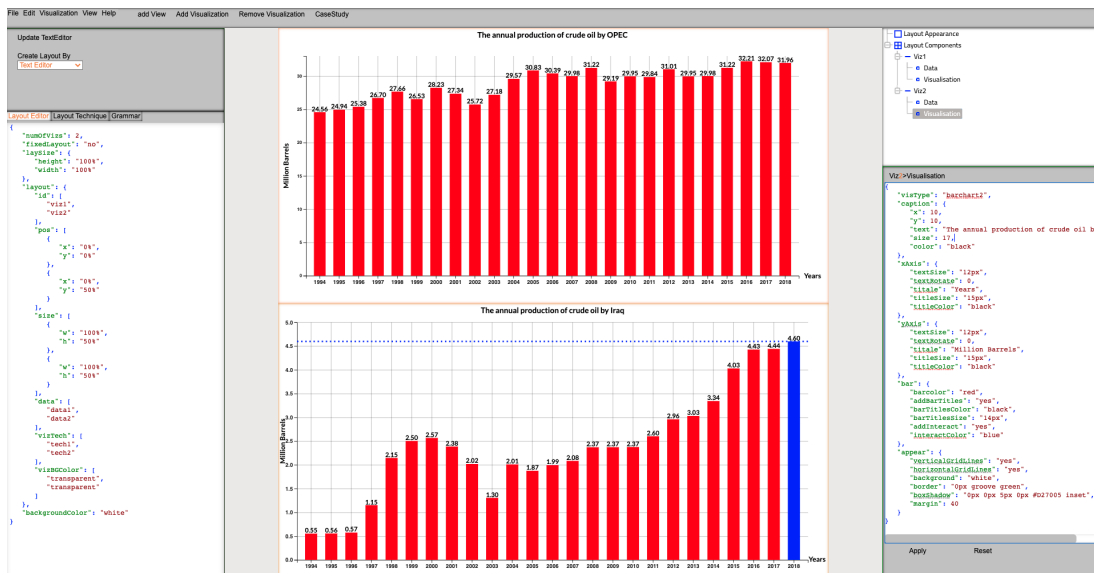


Figure 7.4: Main screen showing two bar chart visualisations. Users can choose a starting layout, search for a specific view quantity, various multiple view Layouts can be created with different views number, and the user can later edit the template.

This example demonstrates how someone can add in D3.js code, to create a 2-view bar chart, as shown in Figure 7.4. The code gets added into the data part (as shown in the panel on the right of the two bar charts, in Figure 7.4. When the code is created then the user can look at the main visualisation, and interact with it (if this is included in the D3.js code). The final visualisation, therefore, is shown in Figure 7.5.

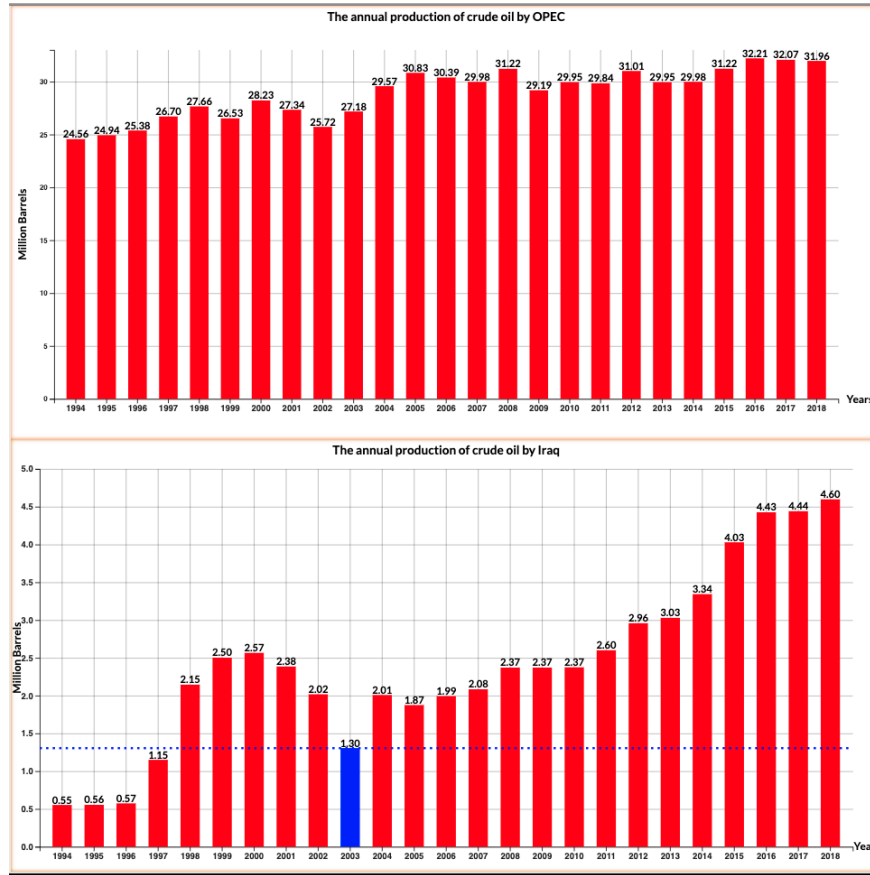


Figure 7.5: Final visualisation. This screenshot demonstrates the final visualisation for the first case study. It shows a simple D3.js side-by-side view of two bar charts.

7.4 Second case study

The second case study demonstrates a more complex visualisation layout. This example shows how a user can edit and adapt the full grammar, or the shorthand grammar. Like the first case study, the user is first shown the Template viewer. Where they can choose the initial layout. Should a user want to start from scratch, they can choose the blank view, with no multiple views. Then they can add the views using the shorthand grammar, or by typing the grammar. However, it is usually going to be easier to start with a layout that is close to what the user requires, and then adapt the layout to fit the need of the user. The template viewer is shown, for the second case study, in Figure 7.6.

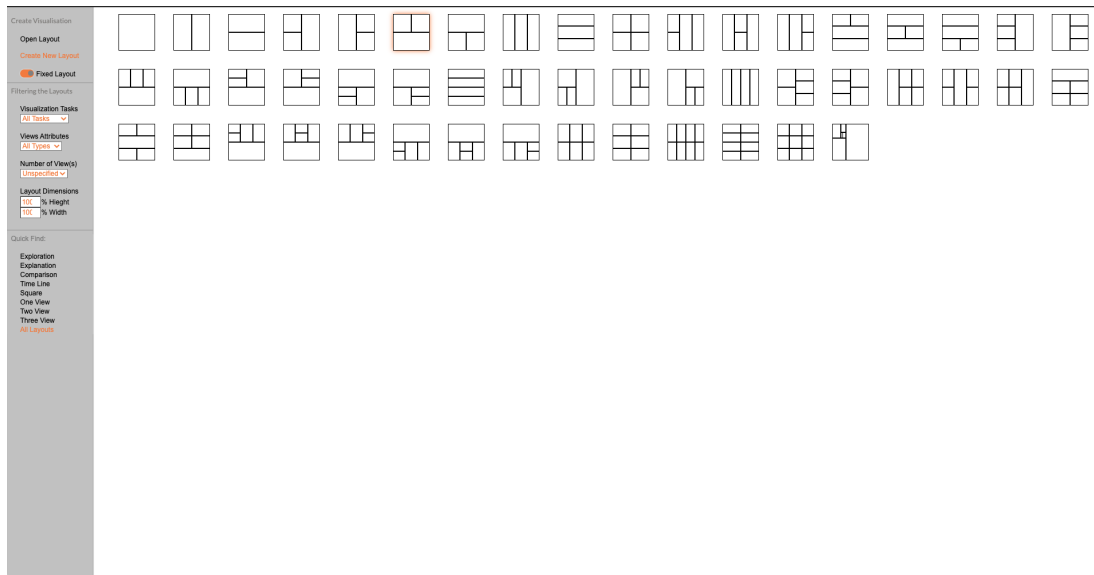


Figure 7.6: Template viewer. We start again, for case study 2, on the template screen. In this case the user selects the 2 by one view layout.

The next goal for the user is to make the layout their own. To adapt it to their needs. This can be done in several ways. Figure 7.7, shows how a user can (A) change the grammar, and (B) change the appearance of the layout. When the user wants to adapt the grammar, they also have a few choices to make. They can choose the Layout Editor, Layout Technique or the Grammar. In other words, they can choose to adapt the layout manually by creating cuts over the graphical output of the views. This is a visual process and users can snap the graphical elements, cut them and divide them to create the desired layout. Or they can choose to edit the grammar itself, or they can edit the shorthand grammar.

The shorthand grammar is shown in Figure 7.8. It demonstrates the split, and the grammar as the template viewer has created. Each of the views are split in half. And there is one horizontal, and one vertical.

The next stage is to consider the appearance. Users can change several aspects of the appearance of the views. They can change different colours, whether there are vertical or horizontal lines the delineate the multiple views, and if there are borders or shadows. Also the user can determine the padding around the views. This is shown in Figure 7.9.

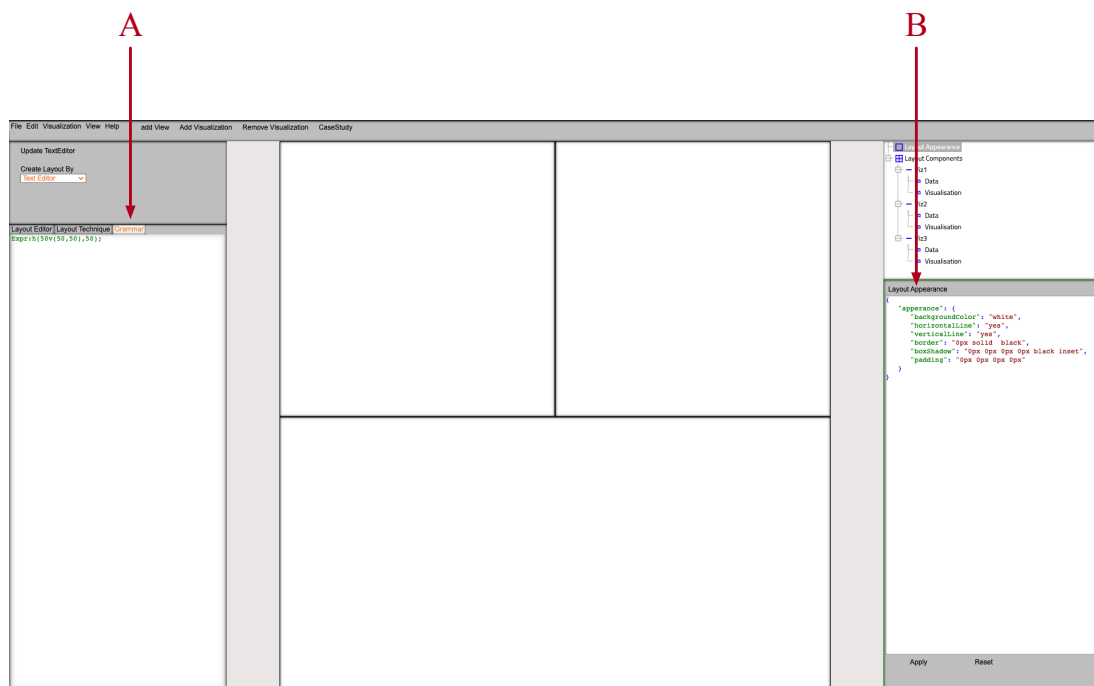


Figure 7.7: Template viewer. Users can choose a starting layout, search for a specific view quantity, various multiple view Layouts can be created with different views number, and the user can later edit the template.



Figure 7.8: Shorthand grammar viewer. Users can write the shorthand grammar in this window, which will display (when updated) in other views. Using the shorthand users can create complex views, using some simple commands.



Figure 7.9: Layout Appearance views. The user can adapt different aspects of the appearance. They are able to choose a specific Component of the layout. The tree is automatically updated from the information in the MVG grammar. Subsequently, users can then select the Viz, add the Data and the Visualisation code. Here the Layout appearance is shown in the lower view. Users can change the parameters to adapt how the views appear.

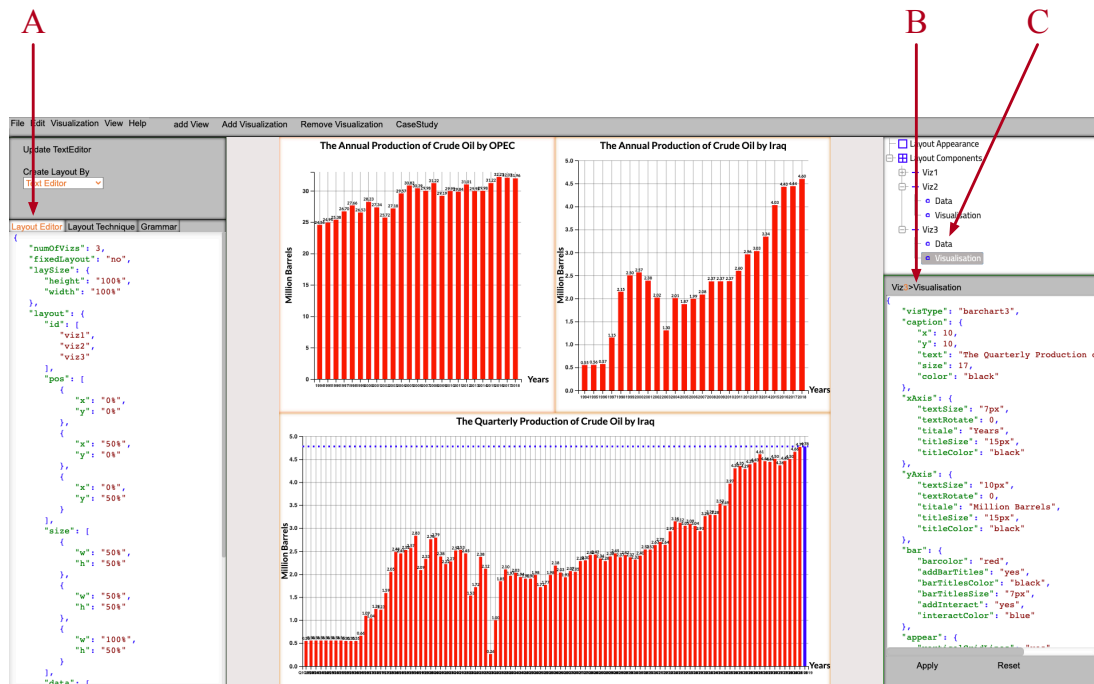


Figure 7.10: This screenshot for the LayMV tool shows the result of the creation process of the multiple view visualisation. Where label “A” points to the layout editor (Figure 7.11 gives a clear picture with more details about this part of the LayMV tool). Moreover, label “B” points to the properties window of the visualisation technique that located in the third view at the bottom of the multiple view visualisation (Figure 7.12 gives a clear picture with more details about this part of the LayMV tool). Finally, label “C” points to the Data section in the tree navigator that belong to the third view (Figure 7.13 shows a screenshot for the Date window that should appear instead of the properties window of the visualisation technique in the user click on the Data part that belongs to the third view).

Figure 7.10 shows the final screenshot, for the second case study. Because it is difficult to understand the individual parts from this picture, each part will be explained separately in the following paragraphs. Different parts of the LayMV system will be highlighted, shown as a large picture and explained. These are shown in the next five pictures and explained below.

Figure 7.11 shows a clear picture of the layout editor where all the details of the multiple views are located, the user of the LayMV tool can change the parameters in this editor to change the position and the size of the views. In addition, the user can assign a data set that belongs to a view to another view.

In addition, users can create and edit the visualisation technique that belongs to a view through the properties window. First, the user should select the visualisation section (the word “visualisation”) in the tree navigator that belongs to a specific view. Then the properties window of the visualisation will display underneath the tree navigator window that relates to the selected view, as shown in Figure 7.12.

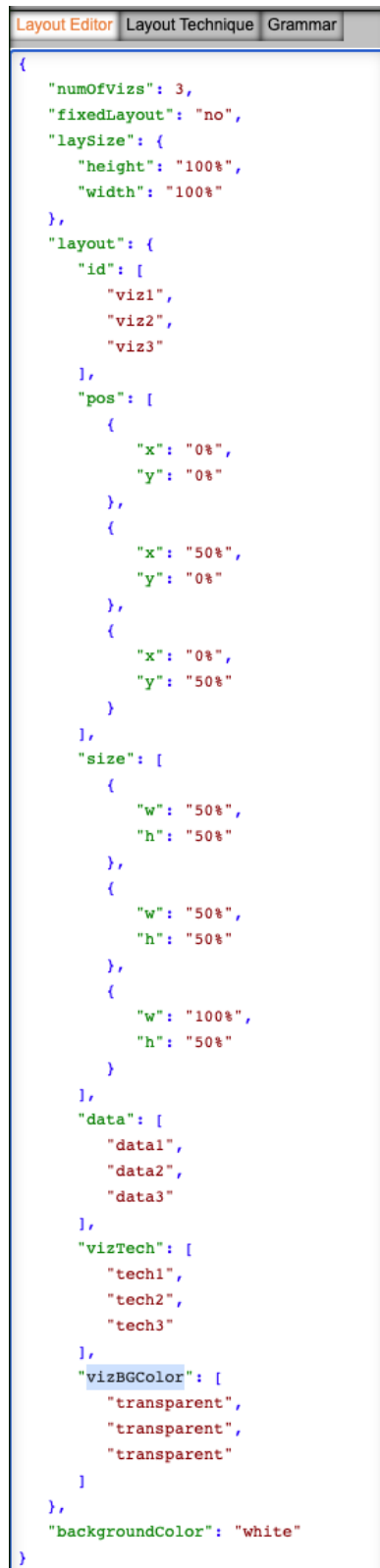


Figure 7.11: This figure shows a clear picture for the layout editor window that was labelled in Figure 7.10 as “A”.



Figure 7.12: This figure shows a clear picture for the properties window of the visualisation technique that was labelled in Figure 7.10 as “B”.

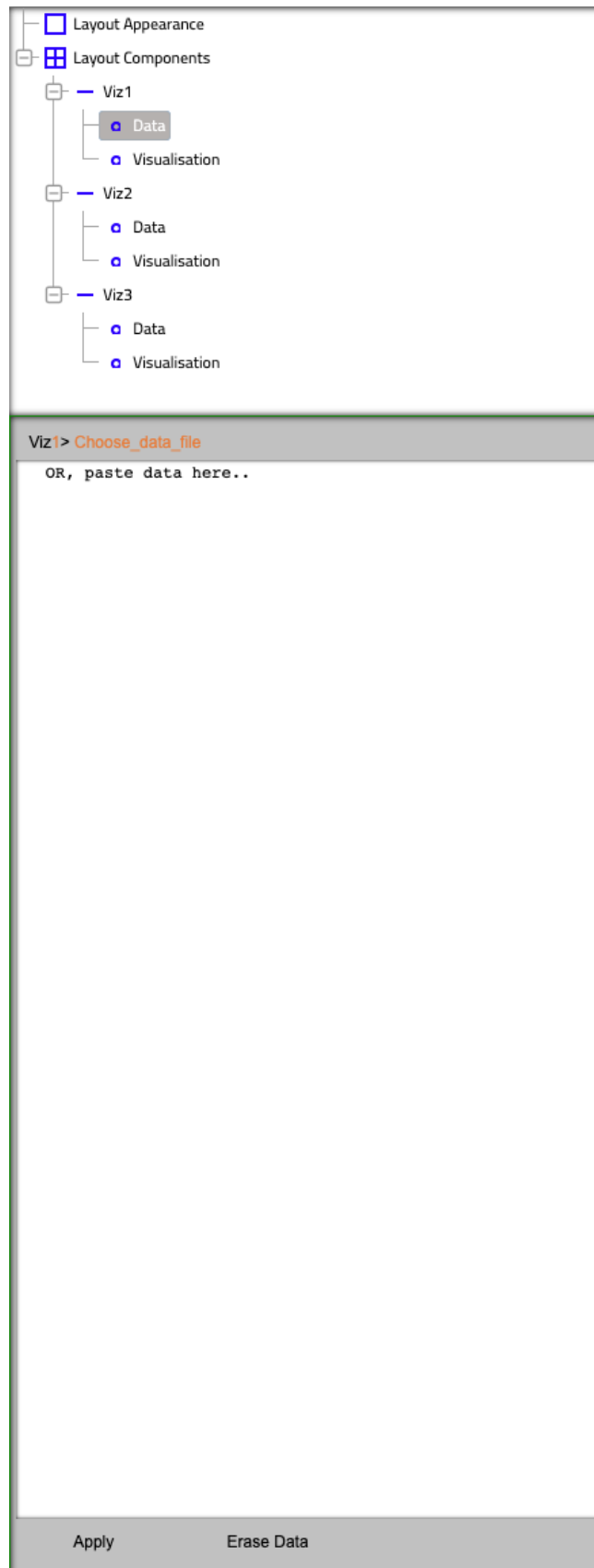


Figure 7.13: This figure shows a clear picture for the Data window that was labelled in Figure 7.10 as “C”.

Furthermore, Figure 7.13 shows the data window where a user can assign data to each view by writing the data directly in this view or copy it from a CSV file and paste it into the window or through upload a data file (file.csv) by clicking on the “choose-data-file” button.

Figure 7.14 shows a screenshot of the multiple view visualisation; if the user want to interact with the multiple view visualisation then he should using the LayMV tool.

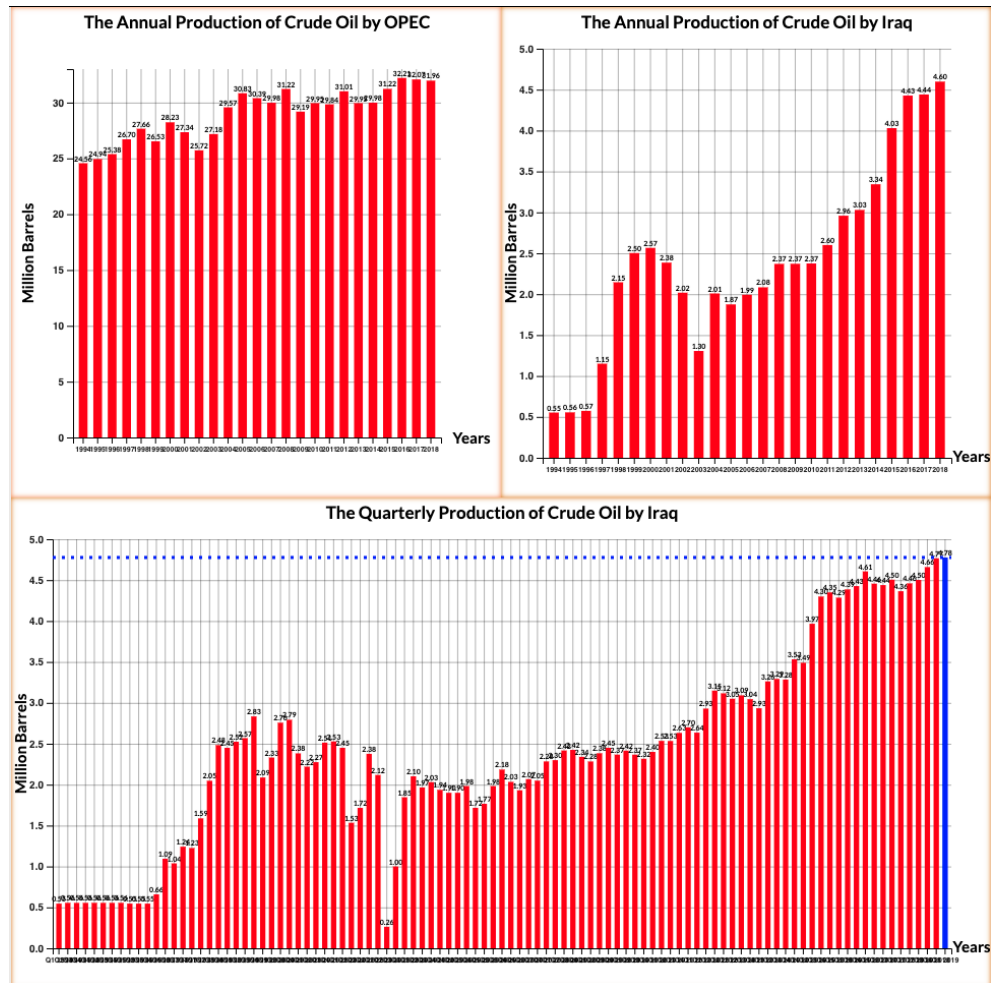


Figure 7.14: Screenshot for the multiple view visualisation that created by the LayMV tool.

The user may modify the layout of the multiple view visualisation by manipulating the MVG grammar, as shown in Figure 7.15.

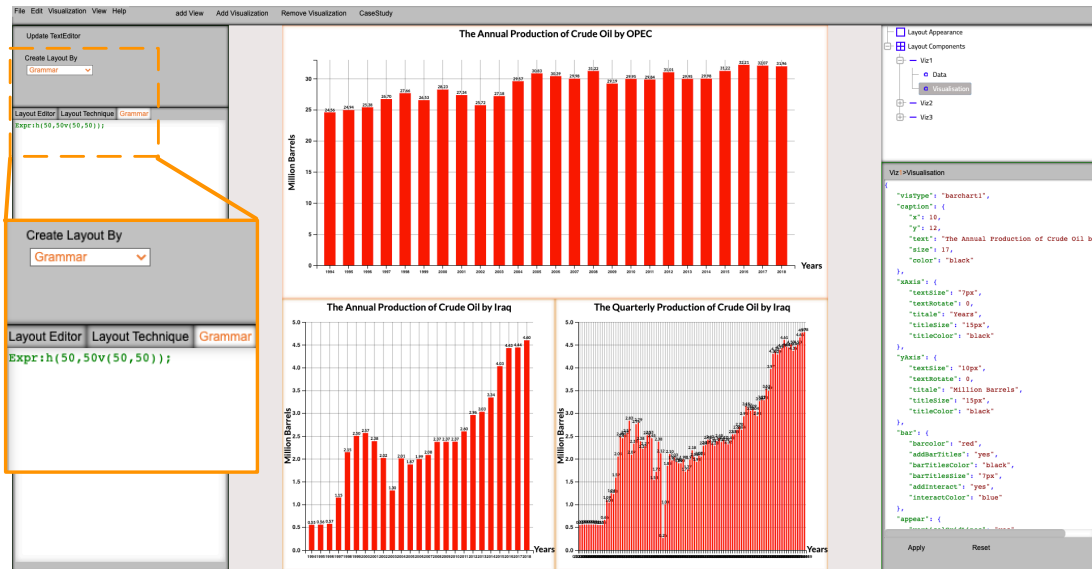


Figure 7.15: This figure shows how the user can change the layout of the multiple view visualisation, that shown in Figure 7.14, using the MVG grammar

7.5 Third case study

The third case study demonstrates that the LayMV tool can create a multiple view visualisation with eight views. What is developed is a layout similar to the example layout developed by Keshif [161]. This layout is achieved as follows:

- First, the layout is created through an MVG grammar expression, as shown in Figure 7.16 below, where the layout is divided into three side-by-side views. Next each view is divided into three, two and three views, respectively.
- Second, the LayMV tool should parse the MVG grammar expression to create the required layout.
- Third, eight datasets are uploaded. And the user needs to associate them with each view.
- Fourth, the LayMV tool will generate a visualisation for each view.

```
a : (34 , 33 , 33);
Expr : v(30ha , 40h(50 , 50) , 30ha );
```

Or ,

```
Expr : v(30h(34 , 33 , 33) ,
        40h(50 , 50) ,
        30h(34 , 33 , 33));
```

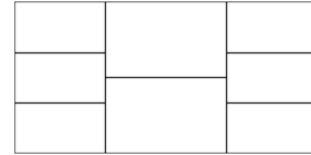


Figure 7.16: The MVG grammar to create a layout with eight views.

The first stage is to use the template-viewer (LayMV tool first-screen) to start the layout creation process. This can be achieved by clicking on the “Create new layout” button” at the top left corner, as shown in Figure 7.17 below, this will then take the user to the next screen.

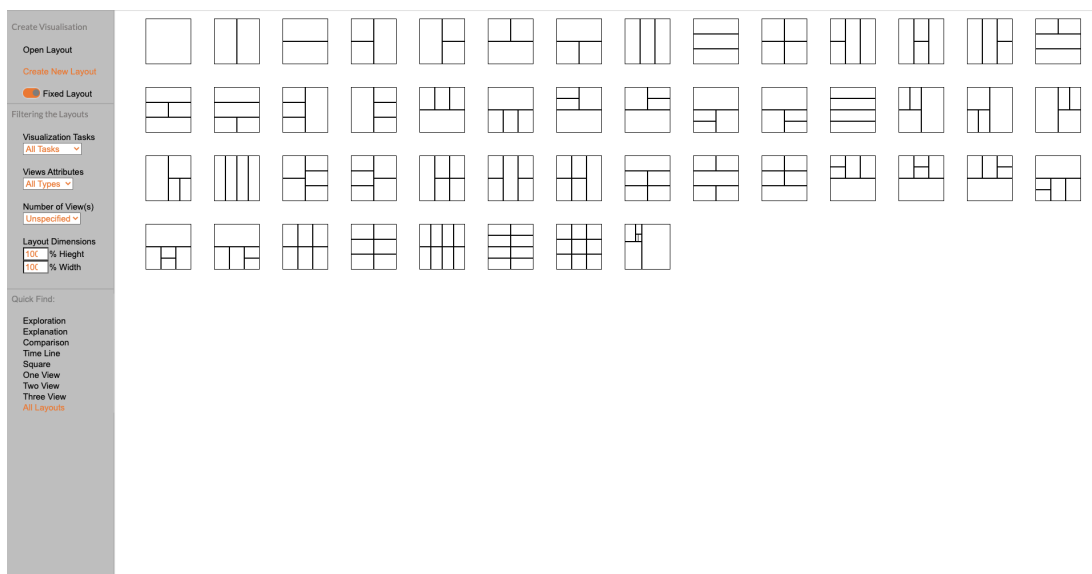


Figure 7.17: Template viewer. Starting again from scratch for case study 2. First use the template screen. In for this case study, the user selects the 2 by one view layout.

In the LayMV tool second-screen, the user can write the MVG grammar expression for the required multiple view, as explained in Figure 7.16. In this window the user writes the expression inside the Grammar editor, located in left side of the tool, as shown in Figure 7.18 below. Then, the user chooses the option to “Create layout by grammar”. and that will create the multiple view layout.

The MVG grammar is shown clearly in Figure 7.19. This demonstrates how users can split the views into sub views. Then, the user can change the layout's dimensions through “Layout editor”, as shown in Figure 7.20 below, which will allow them to control the layout's shape.

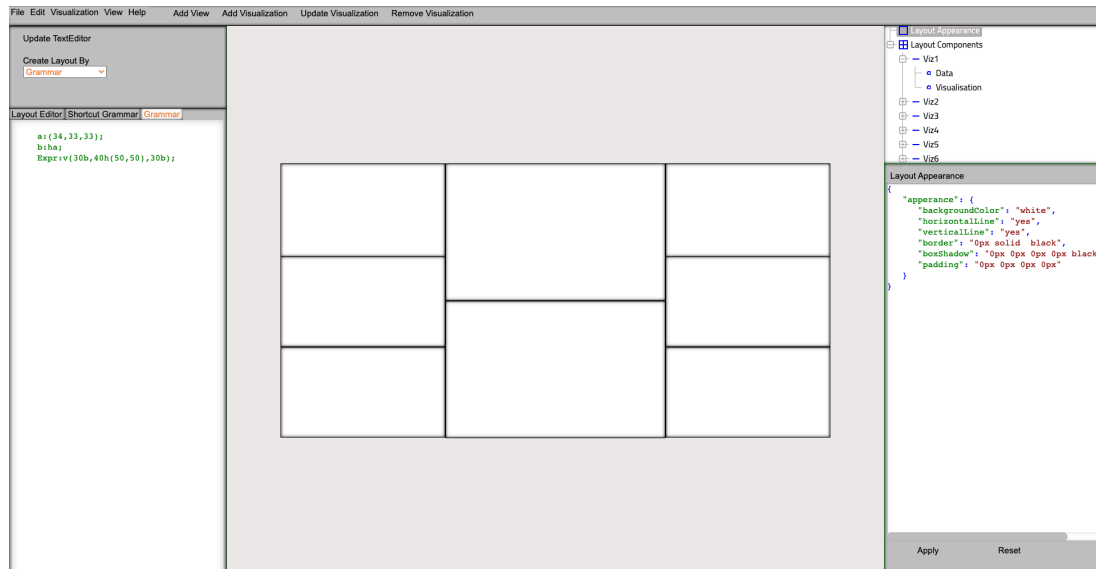


Figure 7.18: Template viewer. Users can choose a starting layout, search for a specific view quantity, various multiple view Layouts can be created with different views number, and the user can later edit the template.

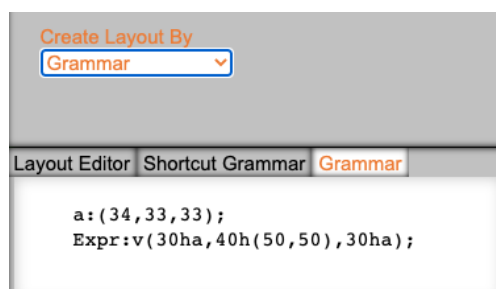


Figure 7.19: We used the grammar viewer users can create the multiple view layout in this window.

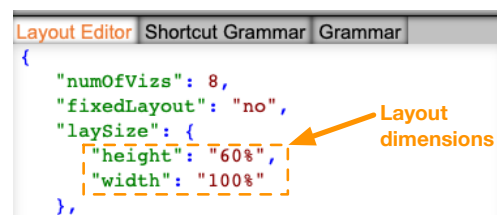


Figure 7.20: From the layout editor we can control layout's dimensions.

The next goal, is for the user to create the actual visualisation for each view based on its dataset. For that, the user looks to the tree-navigator panel, which is located at the top right corner of layMV tool's second-screen. This navigator helps the user to select a specific view name which represent a view in the layout, and then they can select a "Data" node from the tree-navigator to assign a dataset for each view.

Figure 7.21 shows the final screenshot after the user has created the multiple view visualisation for the third case study. As it is difficult to see all the details in this picture, details will be explained in the next set of paragraphs and pictures.

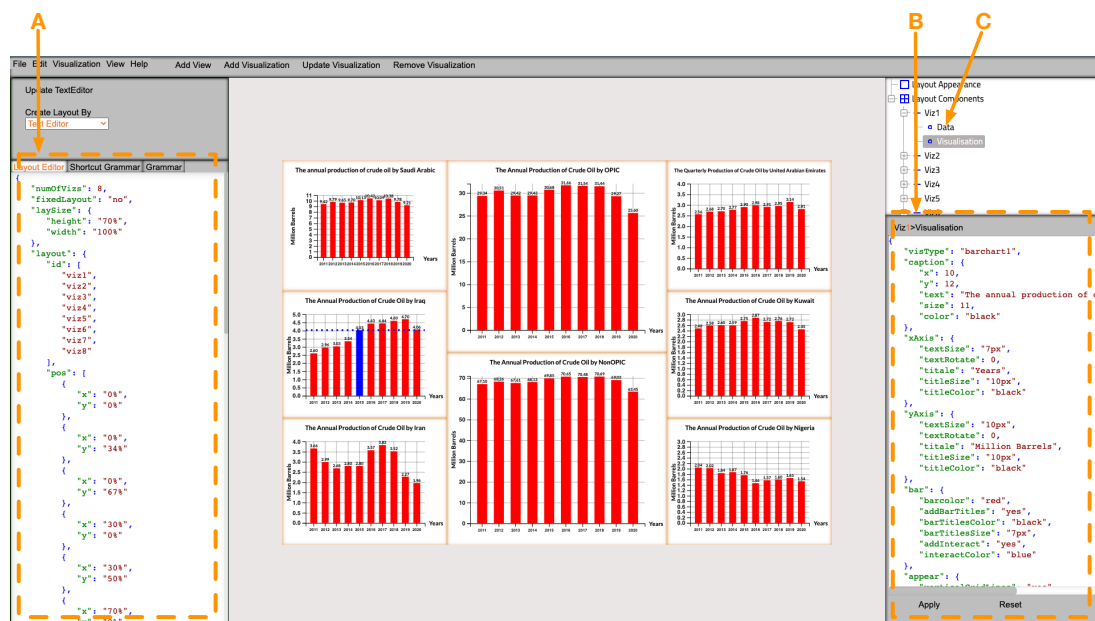


Figure 7.21: This screenshot for the LayMV tool shows the result of the layout creation process of the multiple view visualisation. Label “A” points to the layout editor, where (Figure 7.22 gives a clear picture for this part of the LayMV tool). Moreover, label “B” points to the visualisation properties window that located at the bottom corner of the LayMV tool, where (Figure 7.23 gives more details about this part of the LayMV tool). Finally, label “C” points to the “Data” node in the tree navigator that belong to the third view of the multiple view layout, where (Figure 7.24 shows a screenshot for the Date window that should appear instead of the visualisation properties window when the user click on the “Data” node).

Figure 7.22 below shows the layout editor. This figure explains how we can command the LayMV tool's parser to display the right visualisation-type template in the visualisation properties window for each view, where we can control the appearance of the visualisation, as shown in Figure 7.23 below. For this case study, we chose the bar chart visualisation for each view in the multiple view layout by putting the word **"Barchart"** in the **"vizTech"** section at the **"layout editor"**. Then, we clicked on the **"Create Layout by"** button after choosing the **"Text Editor"** option at the top right corner. This procedure allows us to define the appearance of the bar chart visualisation. We can use the tree-navigator to switch between views to control the visualisation properties for each view in the multiple view layout.

Likewise, we can extend the ability of the LayMV tool to create various visualisations (in addition to bar chart) by adding more D3.js visualisation types to the LayMV's code. Then, we can add a visualisation type, for example, **"line chart"**, to a view in multiple view layout by following the same procedure above to create and control the line chart visualisation.

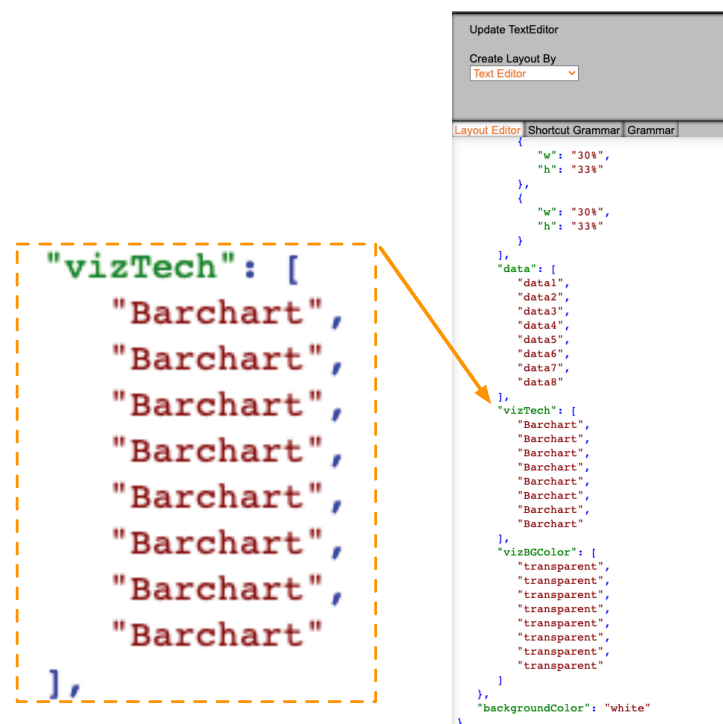


Figure 7.22: This figure shows a clear picture for the layout editor window that was labelled in Figure 7.21 as "A".

In addition, the layout editor can be used to control the multiple view layout as all the details of the multiple view layout are located in the layout editor. The user, of the LayMV tool, can change the parameters in this editor to change the position and the size of the views.



Figure 7.23: This figure shows a clear picture for the properties window of the visualisation technique that was labelled in Figure 7.21 as “B”.



Figure 7.24: This figure shows a clear picture for the Data window that was labelled in Figure 7.21 as “C”.

Finally, Figure 7.24 above shows the data window where a user can upload datasets and correlate these datasets to the multiple view layout.

From all these actions, the user can observe the multiple view visualisation. Figure 7.25

below shows the result. Now, if the user wishes to change anything, they can. For example, users can change the multiple view visualisation either by changing the layout structure through the MVG grammar (or the layout editor), or by changing the data that assigned to the views, as we explained in Section 7.3 and Section 7.4.

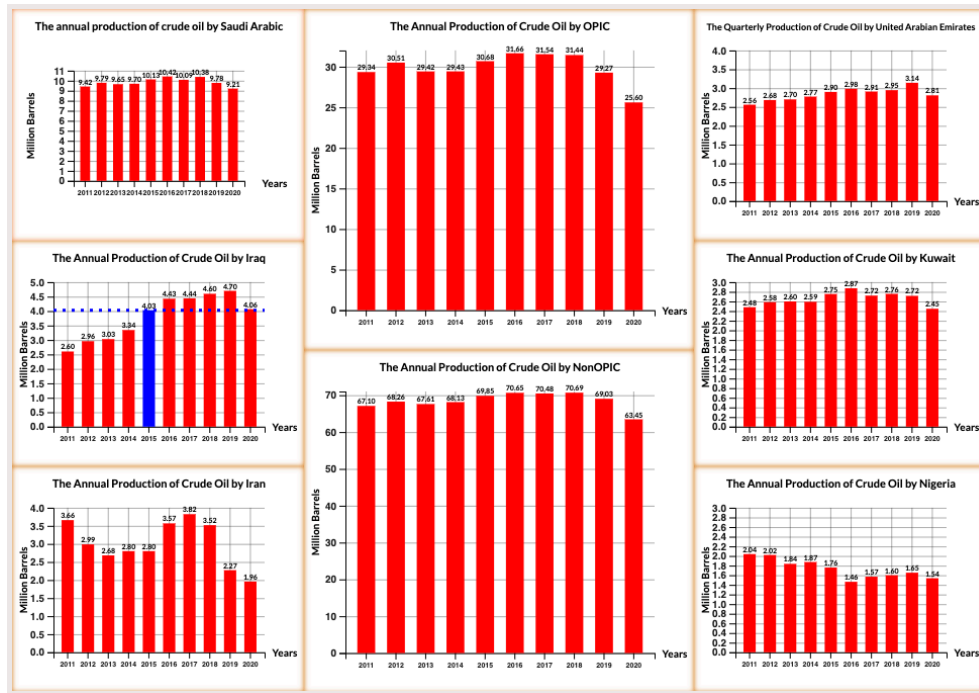


Figure 7.25: Screenshot for the multiple view visualisation that created by the LayMV tool.

7.6 Discussion

This chapter focused on explaining how a user can use the LayMV tool to create different layouts, and develop multiple view visualisations using the MVG grammar. In addition, the chapter explained how someone starts using the tool, and then adapt the view layouts. The case studies demonstrate how the tool can be used, grammars created and saved, and how the grammar can be used to create a visualisation layout.

There are limitations to this study. The case studies started with simple few-view systems moving to an 8-view system. The case studies do not include an example of a larger viewed systems (such as a 20-view system); although the larger viewed systems are merely an extension of the same process. Furthermore, the case studies did not exemplify the full potential of the grammar, such as the full potential of short-hand methods, but these were covered in previous chapters. It is certainly possible to create

different examples, however the grammar chapter, with the implementation and this case study together do explain the full potential of the LayMV tool and grammar. Indeed, it must be emphasised that the focus of the case studies was to demonstrate the principles of the tool, and use the grammar, rather than focus on the usability of the tool. This thesis has substantial evaluation and analysis of quantification, which took much time. Consequently it was decided to focus on a case study approach, use expert heuristics to develop the tool through prototype development, rather than doing another study with other users at the end of the project. While the tool is useful and has much potential, it is important to reiterate that the LayMV tool itself was designed to demonstrate the principles and application of the grammar, and not be a completed commercial product.

While there are limitations, there are successes too. The LayMV tool makes several useful contributions. The “template view” is an idea that can be extended to other visualisation tools. Generating demonstration code, and draft layouts is powerful idea. This idea could be extended to other visualisation systems, for instance, it could be possible to create demonstration (draft) visualisation colour maps, which can then be adapted. This is similar to the ‘show me’ concept in Tableau, where users can select different buttons to see what they can do with the data. Loading and saving the information in the JSON format is useful, and allows users to create visualisation layouts and then save them, load them and adapt them. In one regard, the whole LayMV system, and integration of the grammar, flips the visualisation design on its head. Users first think about the layout of their tool, and then how to add the visualisations. Users should follow the suggested processes, as described in Figure 7.1.

7.7 Summary

This chapter focused to explain how users of the LayMV tool can create multiple view visualisations, this was done by following two case studies. First creating a 2-view system, second a 3-view system and third a 8-view system. In each view a d3.js visualisation is added, and the appearance of the frames are determined. Consequently, this chapter answered two research questions: “Q1/ How can users use the LayMV tool and the MVG grammar to create a multiple view layout?”, and “Q2/ How can users start using the LayMV tool, and what is the process they need to go through?”.

This chapter also included a process by which the user can create multiple views, starting with a sketch of their idea, following the layout, the grammar, and adding the visualisations along with data. In the preparation process of the case studies, the proposed layout was sketched before each case study, data prepared and decisions made over the type of visualisations that would be used in each view. In the implementation process, we described the steps that the user should follow to implement the multiple view visualisation, including how the user can choose a template layout from the first screen or create a custom layout using the MV grammar or create the multiple view layout manually by adding views and using the mouse to arrange the views. Next, we explained how the user of the LayMV could append a data set and create a visualisation technique for each view and change the appearance of the multiple view visualisation.

The goals from creating these case studies were to explain how the LayMV tool works and discover the advantages and limitations of the tool. Finally, this study described how the user of the LayMV tool could manipulate the multiple view visualisation by changing the data or the multiple view layout or the visualisation techniques.

The next chapter reflects on the research questions of the thesis, and work achieved. In addition, general discussions on the successes of the research are made, the limitations and opportunities of future work of this research.

Chapter 8

Discussion and conclusions

The vision of the thesis was to investigate the layout of multiple views. Through the course of this study research six publications have been achieved and two further papers are pending publication. Related work has been investigated, values quantified, new grammar developed and a new tool created. This chapter reflects on the achievements and discusses potential further research.

8.1 Discussion

Multiple views is a broad topic, that has been applied to many disciplines. The topic has certainly attracted and excited a large set of researchers, who have used the technique to a wide range of application domains. From medical applications, volume data, statistical information to virtual reality – see the Related Work chapter (Chapter 2).

Multiple views, as a technique for visualisation, has been around for many years. The idea of seeing information from different perspectives, and placing those views in different windows, has been achieved since early windowing systems were developed. Merging and linking between many windows has been a part of computing interfaces since the early days and development of the graphical user interface, and graphical operating systems. Early Windows – windows, icons, menus, pointer (WIMP) interfaces – interfaces, interactive Unix graphical user interfaces, and other systems, all demonstrate many windows. However the use of multiple views for visualisation started in the mid 1970s, especially with brushing and scatterplot interfaces by Becker and Cleveland [12] and Tukey in the Prim9 system [45]. Subsequently it is now in its third wave of research in the area, as discussed in the Related Work chapter (Chapter 2). In the first wave researchers investigated the technique and explored how brushing was used. In the

second wave people created tools and researchers were using multiple views in a variety of application areas. Now, in the third wave, researchers are starting to understand how people use multiple views, and how they can be applied beyond the desktop and through (for instance) immersive displays [106]. Indeed, nowadays multiple views and multiple coordinated views are expected technologies. Users expect to be able to move information between windows (through at least copy and paste), to see the information displayed in different windows, and to understand the information in different ways. Products like Tableau, Qlik and other commercial visualisation tools all allow multiple views. Software libraries such as R, Matlab, and libraries such as D3.js and other JavaScript visualisation tools all can help to create different multiple view visualisation tools. However there are still challenges and questions over multiple views that remain unanswered today. Indeed, this study has looked at some of the research questions over multiple views, starting with the Introduction and working through each of the chapters. We have investigated and investigated multiple views through many viewpoints and using different strategies.

Chapter 1 provided an introduction, the vision and aims for this body of thesis. The vision of the work was to develop guidelines for visualisation developers and teachers, such that:

If we (as researchers) can understand the multiple view visualisation area better, and place some quantification analysis on multiple coordinated views, then we will be able to develop a set of guidelines which help developers understand how to create appropriate tools.

In order to investigate this vision fully, the objectives of the project were:

Obj 1. Preparing and coding multiple view visualisations.

Obj 2. Quantifying multiple view visualisations.

Obj 3. Analysing the quantification and introducing a set of guidelines to help developers to create multiple view visualisations.

Obj 4 Introducing a tool to create multiple view visualisation based on multiple view grammar.

Chapter 2 described the related work, and presented an overview of the topics in multiple views, things like tools, coordination, grammar and the guidelines. This chapter explained that researchers have moved from treating multiple views as a novel technique, to developing systems that utilise multiple views, to now (in the third wave) looking at how they can be used beyond the windows, icons, menus, pointer (WIMP) interfaces [106]. The chapter also presented a broad background into different words that people use in topic of ‘multiple views’. These include, side-by-side views, juxtaposition, separation of concerns, multiform, and so on.

Chapter 3 illustrated the preparing and coding processes, this helped to answer the research questions RQ1, RQ2 and RQ3. This chapter explained how a database of images was created and how they were analysed. The chapter proposed a new methodology, that can be used by other people, to extract suitable images from research papers. This chapter also helped to develop understanding into the breadth of how people have used multiple views, and the diversity of application areas in multiple view systems. From this work a new coding methodology was developed, that enabled layouts to be classified, and the work started to understand the different layouts ready for the quantification analysis of the next chapter.

Chapter 4 quantified the data. The chapter also presented a set of design guidelines for multiple view visualisation, which helped to answer research questions RQ4, RQ5, RQ6, RQ7 and RQ8. This chapter also explained the results from the view quantification. From this analysis it was clear to see that people preferred to use three-view systems. The work also proposed that there was a long tail in the statistics, demonstrating that there are people who used many views in their visualisation systems. However, over ten-view-systems were used rarely. The work also demonstrated that most multiple view systems (80% of the multiple view systems) were six views or less. Furthermore, the work also demonstrates that one-view systems were useful, and have their place in scientific presentation.

Chapter 5 demonstrated how the a grammar can be created to help people describe

multiple view layouts. The grammar is explained, and its shorthand form, to address complex layout arrangements. In addition, the chapter explained that it was possible to easily create simple layouts from the grammar, as well as more complex layouts, through a human readable grammar. And the grammar is powerful to be able to describe complex layouts.

Chapter 6 described the stages that been used to build the multiple view tool that used multiple view grammar, helping to answer research question RQ10. The chapter also demonstrated that it was possible to incorporate the grammar into a visualisation system. The chapter demonstrated how it was suitable to create a multiple view system to allow users to create different multiple view layouts. Users can use the grammar to create the multiple view layouts, or can use the shorthand, or can design layouts through a graphical editor. Each of these present the user with different ‘viewpoints’ on the information. This, therefore, creates a very powerful, and comprehensive multiple view system for editing different multiple view layouts. The chapter demonstrated that it was possible to integrate visualisations (the examples used D3.js) and change the appearance of the layout of the views.

Chapter 7 presented three case studies, which demonstrate the functionality of the LayMV multiple view tool that was described in the earlier chapter. Finally, the case studies explain in a systematic way, how people can use the different buttons and menus of the tool to create different visualisations, and alternative multiple view layouts.

8.2 Consideration of the work

Developing multiple view systems is difficult. There are many challenges to overcome. Developers, creating any visualisation, need to decide on how to load the data, how to map and associate the data to the visualisation, and choose which visualisation type they will use. It does seem that developers leave the layout of views to the last moment. However, from a design point-of-view perhaps the design layout of the multiple view system, should be the first consideration. While visualisation tools do often provide the user with some functionality of laying out different views it is still difficult for users to layout their designs appropriately. The layout can be better provided, and more

functionally rich expressions can be allowed, through better interfaces and access to grammars.

This thesis has discussed, created and demonstrated both a grammar (MVG) and a system (LayMV) to help people create multiple view layouts. The proposed ‘systems’ allow users to model their multi- layout systems by either controlling the grammar, or through the design environment of the interactive tool (the LayMV tool). The LayMV tool has therefore much potential. And it is hoped that the structures, methods and operation of the tool will be used and taken up by other researchers. The grammar could be readily integrated and used with other libraries. The resultant layouts can be used with different libraries (e.g., D3 demonstrations were given).

These systems, however, are only the start of the journey, to help people create complex multiple view systems more easily. Indeed, they need to be integrated into visualisation systems that help and advice the developer; helping users create better designed and easier-to-control multiple view visualisations. Researchers have started to do this already. As discussed previously, tools like Keshif and Adobe’s Data-illustrator provide the user with ways to create visualisations more easily. However, the LayMV the MVG grammar make an excellent contribution towards these goals. And the next step would be to geth the grammar and the layouts from the LayMV system to be used more widely.

But tool design, is not only (and should not be) about “using tools to create multiple view visualisations” but needs to incorporate ‘best practices’. The research community, in their research papers, obviously are presenting ‘best practices’, and researchers are starting to develop more ‘guidelines for visualisation’. Certainly we (as a community) are only at the start of this journey. Indeed, it does appear that more research papers are being published that propose guidelines, taxonomies, structures, new design methods, and help to increase data-literacy. However, it is important to note that if these ‘ideas’ and recommendations of ‘good practice’ only stay within the academic community then the general public will not improve. This thesis has studied, quantified data, and from the quantification process proposed and demonstrated a suite of recommendations and guidelines over the use of multiple coordinated views. Again these are a great start, and develop from, and improve on guidelines from previous researchers (such as Baldonado [144]). However, unless these guidelines are exposed and championed to

the general public then they will, again, stay in the academic domain and not be used by the general population. What is required now are ways to take these ideas forward and champion them to the general public. Perhaps a potential next step would be to try to get the guidelines taught in Schools and colleges. To promote them to the public through social media, for instance, and present them in a way that they can be consumed by the general public.

8.3 Reflections on the research questions

At the outset of the research, eight research questions were posited, see Chapter 1. Each question has been systematically and comprehensively addressed. The research successes, in answer to these questions, is summarised as follows:

RQ1 What is the strategy to code layout topologies and visualisation types? To answer this question, sketches were first made, by using the figure-ground method. First, all content from the views were removed. Then the layout was coded, based on the layout topologies. These topologies were coded, and the visualisation types for each view stored. This thesis addressed this question in Chapter 3.

RQ2 How many views are used in multiple view systems? The solution was to quantitatively analyse the number of views that developers used, presented in the visualisation literature, by extracted images from visualisation papers presented in the IEEE visualization Conference series, over a seven year period, coded the images and calculated the quantity of each configuration. These results can be used by developers to guide them in decisions over view quantity and design. This thesis addressed this question in Chapter 4.

RQ3 Do developers prefer symmetrical or non-symmetrical layouts for multiple view systems? The solution, to answer this question, was to quantify and analyse the topology of multiple view layouts. This thesis addressed this question in Chapter 4.

RQ4 What layout arrangements are popular in multiple view systems? When answering this question, items were classified their basic topological layouts, and results

discussed to confirm decisions made. The resultant taxonomy (and popularity of each configuration) can be used by users who wish to configure multiple view systems and developers to create suitable systems. This thesis addressed this question in Chapter 4.

RQ5 What visualisation types are used in multiple view systems? What visualisation types are the most frequently used? Are some layout arrangements more likely to hold certain types of visualisations? The solution was to code and quantify the types of visualisations (bar chart, line graph, scatter plot etc.) and their use in different layouts. This information was used to develop and summarise design guidelines and identified best practices in multiple view visualisation. This thesis addressed this question in Chapter 4.

RQ6 What types of visualisations come together in multiple view systems? To answer this question, a basket analysis was used. This provided a set of correlations and results to help answer this question. This thesis addressed this question in Chapter 4.

RQ7 What salient guidelines can be learnt from the analysis, to help users to design and develop robust multiple view visualisations? The analysis of the answers for the above questions assisted to provide a set of guidelines to help the learners in the visualisation field and new developers to build multiple view visualisations. This thesis addressed this question in Chapter 4.

RQ8 What is a multiple view grammar and how can it be used in multiple view tools to create multiple view layouts? A new grammar has been defined (that can be used to store and edit view layouts). The grammar was also used to develop a new multiple view visualisation system, to help people create different view layouts. These ideas were also demonstrated in several case studies. This thesis addressed this question in Chapter 5, Chapter 6 and Chapter 7.

8.4 Limitations and future work

There is much future work that can be achieved such as minor improvements to the research. For instance, the LayMV tool could be adapted to include more coordination between views. While the tabbed views is useful, it may be better to separate these tabs into individual windows. Also, while the work demonstrates that D3 visualisations can be integrated into the tool, it would be good to explore how other libraries could be included. Furthermore, additional demonstrations could be included to explore a variety of types of visualisation (bar charts, line graphs, scatter plots, geographical maps and so on). While there will always be minor things to improve, there are several major ideas that could be investigated in the future.

First, it would be good to allow LayMV to generate code that can be included into other software. For instance, it can be used to create code that can be incorporated on websites, or mobile phones. There are some software development libraries, and interface creation tools, such as Adobe XD, InVision, or Proto.io that help users create interfaces quickly. They allow users to drag and drop images, and link them from one part of the screen to another, to make hot-zones that (when run) can be clicked to load the next image. In this way the user can rapidly prototype a demonstration system. One idea to develop LayMV towards this direction, whereby the grammar could be created and the tool would output code that could be incorporated by other systems. Another idea could be to adapt towards a suite of tools, and create a tool-chain of individual tools that can be used together to exchange the output into a different form. For instance, like in Unix, many functions can be piped together. Or like markdown is written by the user, and then these tools are used to convert it to an output form, such as HTML or PDF. The grammar could be extended to be used in a similar tool-chain. For instance, it could be possible to create the grammar, store it in a file, run the multiple view extractor function, incorporate multiple-view style sheets, which add code to describe the appearance of the layouts, and then output working code that can be incorporated into other systems. This code could then be used on small mobile phone interfaces or large infrastructure systems (e.g., power grid systems, air-traffic control systems, nuclear power stations, train systems).

Another idea would be to research how to integrate aspects of coordination into the

grammar, and to develop techniques of how to markup coordination. To describe ways to allow the information to be coordinated by the viewers. When this had been created, then it would be interesting to investigate how LayMV could be used to create multiple view systems across devices. How can these be created? How can they be managed? How can individual views be connected? How would it be possible to describe general aspects of coordination between specific parts of the data? Also, our quantitative study did not investigate coordination methods. Further work could be achieved to quantify coordination (such as brushed highlight, and not zooming, window focus etc.). Much like we investigated and quantified the quantity of views used, it would be useful to extend the analysis to how coordination is used by these developers. To achieve this, a way to ‘code’ the types of interaction would be required. Perhaps following from the different types of coordination that North and Shneiderman classify [94] or by classifications and rudiments by Boukhelifa and Roberts [17] or Weaver [148]. In addition, the MVG grammar could be integrated into other systems, and extended for other uses (e.g., in windowing systems such as Windows10 or IOS).

Third, another whole set of research is required to look beyond juxtaposition. Throughout the thesis, and especially in the quantification (chapter 4) and the chapter on grammar (Chapter 5), the chapter on the LayMV tool (Chapter 6) the work focused on juxtaposed views. But how can non-juxtaposed systems be created? Is it possible to define overlaid or enhanced views through the grammar? Then how does interaction get included into the grammar, to allow users to interact with overlaid views? For that matter, how is the information layered? Would there be an order to the layers, and perhaps would some layers be more transparent than others? Layering is often used in geographical visualisation (e.g., see Plumlee and Ware’s work in geo-visualisation [99] or work by Andrienko and Andrienko [6] in time and space). But how can this be done in information visualisation? How can users allow for and control overlapping visualisations?

8.5 Conclusion

This research has comprehensively examined views as presented in the visualisation literature, and have answered eight research questions: presented the strategy to select multiple view visualisation, the strategy to identify views and the strategy to code layouts; the quantity of views, count layout types and count visualisation types; which is help us answered question number seven by provided the design guidelines set from our analysis. Subsequently, a new grammar (MVG) has been developed. This grammar can be used to express view layouts, and designed and built a tool ‘Layouts for Multiple View’ (LMV) to help users lay out multiple view systems. Users can select a template-layout, or edit the layout visually or control the layout through a grammar. The grammar is used to save/load view layouts, and each view is linked, such when the user controls the wireframe editor the grammar updates. Several prototypes have been developed and three core instances of these prototypes were explained. Each prototype was improved through expert heuristic feedback. The LayMV tool was explained and D3 was used to create the visualisations and examples. These case studies demonstrate how the grammar has potential, and could be used with other visualisation languages and tools. The following results were achieved throughout the duration of this project:

- Obj 1. Developed the strategies for the preparing process and the coding process which helped us analysed multiple view visualisations.
- Obj 2. Quantified multiple view visualisations, as seen through research publications.
- Obj 3. Analysed the quantification and introduced a set of guidelines to help developers to create multiple view visualisations.
- Obj 4 Introduced the multiple view tool to create multiple view visualisation based on multiple view grammar.

Bibliography

- [1] Christopher Ahlberg and Ben Shneiderman. ‘Visual information seeking using the FilmFinder’. In: *CHI '94: Conference companion on Human factors in computing systems*. Boston, Massachusetts, United States: ACM Press, 1994, pp. 433–434. ISBN: 0-89791-651-4. DOI: 10.1145/259963.260431.
- [2] W. Aigner and S. Miksch. ‘Supporting protocol-based care in medicine via multiple coordinated views’. In: *Proceedings. Second International Conference on Coordinated and Multiple Views in Exploratory Visualization, 2004*. July 2004, pp. 118–129. DOI: 10.1109/CMV.2004.1319532.
- [3] J. Allaire. ‘RStudio: integrated development environment for R’. In: *Boston, MA 770* (2012), p. 394.
- [4] Robert Amar, James Eagan and John Stasko. ‘Low-level components of analytic activity in information visualization’. In: *IEEE Symposium on Information Visualization, 2005. INFOVIS 2005*. IEEE. 2005, pp. 111–117.
- [5] Robert Amar and John Stasko. ‘BEST PAPER: A knowledge task-based framework for design and evaluation of information visualizations’. In: *IEEE Symposium on Information Visualization*. IEEE. 2004, pp. 143–150.
- [6] N. Andrienko and G. Andrienko. ‘Coordinated views for informed spatial decision making’. In: *Proceedings International Conference on Coordinated and Multiple Views in Exploratory Visualization - CMV 2003* -. July 2003, pp. 44–54. DOI: 10.1109/CMV.2003.1215002.
- [7] Massimo Aria and Corrado Cuccurullo. ‘bibliometrix: An R-tool for comprehensive science mapping analysis’. In: *Journal of informetrics* 11.4 (2017), pp. 959–975.
- [8] Rudolf Arnheim. *Art and visual perception: A psychology of the creative eye*. Univ of California Press, 1965.
- [9] B. Bach et al. ‘The Hologram in My Hand: How Effective is Interactive Exploration of 3D Visualizations in Immersive Tangible Augmented Reality?’

- In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (Jan. 2018), pp. 457–467. ISSN: 1077-2626. DOI: 10.1109/TVCG.2017.2745941.
- [10] Evan Barba et al. ‘Integrating Visual Exploration into Traditional Scientific Research Methodology’. In: *Posters presented at the IEEE Conference on Visualization (IEEE VIS 2016), Baltimore, Maryland, USA*. Oct. 2016.
 - [11] Michael F Barnsley, John H Elton and Douglas P Hardin. ‘Recurrent iterated function systems’. In: *Constructive approximation* 5.1 (1989), pp. 3–31.
 - [12] Richard A. Becker and William S. Cleveland. ‘Brushing Scatterplots’. In: *Technometrics* 29.2 (1987), pp. 127–142.
 - [13] A.L. Berkin and A.S. Jacobson. ‘LinkWinds, a visual data analysis system and its application to remote sensed data’. In: *Proceedings of IGARSS '94 - 1994 IEEE International Geoscience and Remote Sensing Symposium*. Vol. 3. 1994, 1799–1801 vol.3. DOI: 10.1109/IGARSS.1994.399569.
 - [14] Jacques Bertin. *Graphics and Graphic Information Processing*. Trans. by W.J.Berg and P.Scott. Walter de Gruyter Inc, Jan. 1981. ISBN: 3110088681.
 - [15] Jacques Bertin. *Graphics and graphic information processing*. Walter de Gruyter, 1981.
 - [16] Michael Bostock, Vadim Ogievetsky and Jeffrey Heer. ‘D3 Data-Driven Documents’. In: *IEEE Transactions on Visualization and Computer Graphics* 17.12 (2011), pp. 2301–2309. DOI: 10.1109/TVCG.2011.185.
 - [17] N. Boukhelifa, J. C. Roberts and P. J. Rodgers. ‘A coordination model for exploratory multiview visualization’. In: *Proceedings International Conference on Coordinated and Multiple Views in Exploratory Visualization - CMV 2003*. July 2003, pp. 76–85. DOI: 10.1109/CMV.2003.1215005.
 - [18] Richard Brath and David Jonker. *Graph analysis and visualization: discovering business opportunity in linked data*. John Wiley & Sons, Inc, 2015. ISBN: 978-1-118-8-84584.
 - [19] M. Brehmer et al. ‘Matches, Mismatches, and Methods: Multiple-View Workflows for Energy Portfolio Analysis’. In: *IEEE Transactions on Visualization and Computer Graphics* 22.1 (Jan. 2016), pp. 449–458. ISSN: 1077-2626. DOI: 10.1109/TVCG.2015.2466971.
 - [20] D. Brodbeck and L. Girardin. ‘Design study: using multiple coordinated views to analyze geo-referenced high-dimensional datasets’. In: *Proceedings International Conference on Coordinated and Multiple Views in Exploratory*

- Visualization - CMV 2003* -. July 2003, pp. 104–111. DOI: 10.1109/CMV.2003.1215008.
- [21] Ken W Brodlie and NF Mohd Noor. ‘Visualization notations, models and taxonomies’. In: *Theory and Practice of Computer Graphics 2007, Eurographics UK Chapter Proceedings*. Eurographics Association. 2007, pp. 207–212.
 - [22] Eli T Brown et al. ‘Dis-function: Learning distance functions interactively’. In: *2012 IEEE conference on visual analytics science and technology (VAST)*. IEEE. 2012, pp. 83–92.
 - [23] Andreas Buja et al. ‘Interactive Data Visualization Using Focusing and Linking’. In: *Proceedings of the 2nd Conference on Visualization ’91. VIS ’91*. San Diego, California: IEEE Computer Society Press, 1991, pp. 156–163. ISBN: 0-8186-2245-8.
 - [24] Bill Buxton. *Sketching user experiences: getting the design right and the right design: getting the design right and the right design*. Morgan Kaufmann, 2010.
 - [25] Michael D Byrne et al. ‘The tangled web we wove: A taskonomy of WWW use’. In: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. 1999, pp. 544–551.
 - [26] D. B. Carr et al. ‘Scatterplot Matrix Techniques for Large N’. In: *Journal of the American Statistical Association* 82.398 (1987), pp. 424–436. DOI: 10.1080/01621459.1987.10478445.
 - [27] Hong Chen. ‘Compound brushing explained’. In: *Information Visualization 3.2* (2004), pp. 96–108. ISSN: 1473-8716. DOI: <http://dx.doi.org/10.1057/palgrave.ivs.9500068>.
 - [28] Xi Chen et al. ‘Composition and configuration patterns in multiple-view visualizations’. In: *IEEE Transactions on Visualization and Computer Graphics* (2020).
 - [29] E. H. Chi. ‘A taxonomy of visualization techniques using the data state reference model’. In: *IEEE Symposium on Information Visualization 2000. INFOVIS 2000. Proceedings*. 2000, pp. 69–75. DOI: 10.1109/INFVIS.2000.885092.
 - [30] Ed Huai-hsin Chi et al. ‘A Spreadsheet Approach to Information Visualization’. In: *Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology*. UIST ’97. Banff, Alberta, Canada: ACM, 1997, pp. 79–80. ISBN: 0-89791-881-9. DOI: 10.1145/263407.263513.

- [31] EH-H Chi et al. 'A spreadsheet approach to information visualization'. In: *Symposium on Information Visualization*. IEEE. 1997, pp. 17–24.
- [32] Imran Chowdhury et al. 'MIVA: Multimodal interactions for facilitating visual analysis with multiple coordinated views'. In: *2020 24th International Conference Information Visualisation (IV)*. IEEE. 2020, pp. 714–717.
- [33] Jon Christensen, Joe Marks and Stuart Shieber. 'An empirical study of algorithms for point-feature label placement'. In: *ACM Transactions on Graphics (TOG)* 14.3 (1995), pp. 203–232.
- [34] Ding Chu et al. 'Visualizing Hidden Themes of Trajectories with Semantic Transformation'. In: *Posters presented at the IEEE Conference on Visualization (IEEE VIS 2013), Atlanta, Georgia*. Oct. 2013.
- [35] G. Convertino et al. 'Exploring context switching and cognition in dual-view coordinated visualizations'. In: *Proceedings International Conference on Coordinated and Multiple Views in Exploratory Visualization - CMV 2003 -*. July 2003, pp. 55–62. DOI: 10.1109/CMV.2003.1215003.
- [36] P. Craig, J. Kennedy and A. Gunning. 'Coordinated parallel views for the exploratory analysis of microarray time-course data'. In: *Coordinated and Multiple Views in Exploratory Visualization (CMV'05)*. July 2005, pp. 3–14. DOI: 10.1109/CMV.2005.5.
- [37] Tarik Crnovrsanin, Chris Muelder and Kwan-Liu Ma. 'A system for visual analysis of radio signal data'. In: *2014 IEEE Conference on Visual Analytics Science and Technology (VAST)*. 2014, pp. 33–42. DOI: 10.1109/VAST.2014.7042479.
- [38] Giuseppe Di Battista et al. 'Algorithms for drawing graphs: an annotated bibliography'. In: *Computational Geometry* 4.5 (1994), pp. 235–282.
- [39] Helmut Doleisch. 'Smooth Brushing for Focus+Context Visualization of Simulation Data in 3D'. In: *Proceedings of The 10-th International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media 2002 (WSCG 2002)*. Plzen, Czech Republic, 2002, pp. 147–154.
- [40] Jason A. Dykes. 'Exploring spatial data representation with dynamic graphics'. In: *Computers & Geosciences* 23.4 (1997). Exploratory Cartographic Visualisation, pp. 345–370. ISSN: 0098-3004.
- [41] Robert F. Erbacher. 'Visualization Design for Immediate High-Level Situational Assessment'. In: *Proceedings of the Ninth International Symposium on*

- Visualization for Cyber Security*. VizSec '12. Seattle, Washington, USA: Association for Computing Machinery, 2012, pp. 17–24. ISBN: 9781450314138. DOI: 10.1145/2379690.2379693.
- [42] Guillaume Erétéo et al. ‘Analysis of a real online social network using semantic web frameworks’. In: *International semantic web conference*. Springer. 2009, pp. 180–195.
 - [43] Charles W. Eriksen and Derek W. Schultz. ‘Information processing in visual search: A continuous flow conception and experimental results’. In: *Perception & Psychophysics* 25.4 (1979), pp. 249–263.
 - [44] Jennifer Fereday and Eimear Muir-Cochrane. ‘Demonstrating Rigor Using Thematic Analysis: A Hybrid Approach of Inductive and Deductive Coding and Theme Development’. In: *International Journal of Qualitative Methods* 5.1 (2006), pp. 80–92. DOI: 10.1177/160940690600500107.
 - [45] Mary A. Fisherkeller, Jerome H. Friedman and John W. Tukey. ‘PRIM-9: An Interactive Multidimensional Data Display and Analysis System’. In: *Dynamic Graphics for Statistics* (1975), pp. 91–109.
 - [46] S. Fu et al. ‘How Do Ancestral Traits Shape Family Trees Over Generations?’ In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (Jan. 2018), pp. 205–214. ISSN: 1077-2626. DOI: 10.1109/TVCG.2017.2744080.
 - [47] R. George et al. ‘Interactive Visual Analytics of Coastal Oceanographic Simulation data’. In: *Posters presented at IEEE VIS 2012*. Oct. 2012.
 - [48] M. Gleicher. ‘Considerations for Visualizing Comparison’. In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (Jan. 2018), pp. 413–423. ISSN: 1077-2626. DOI: 10.1109/TVCG.2017.2744199.
 - [49] Michael Gleicher et al. ‘Visual comparison for information visualization’. In: *Information Visualization* 10.4 (2011), pp. 289–309. DOI: 10.1177/1473871611416549.
 - [50] Michael Gleicher et al. ‘Visual comparison for information visualization’. In: *Information Visualization* 10.4 (2011), pp. 289–309. DOI: 10.1177/1473871611416549.
 - [51] M. Graham and J. Kennedy. ‘Multiform Views of Multiple Trees’. In: *2008 12th International Conference Information Visualisation*. July 2008, pp. 252–257. DOI: 10.1109/IV.2008.21.

- [52] Robert B Haber and David A McNabb. ‘Visualization Idioms : A Conceptual Model Visualization for Scientific Systems’. In: *Visualization in scientific computing* 74 (1990), p. 93.
- [53] Eric Hall et al. ‘TellFinder: Discovering Related Content in Big Data’. In: *Posters presented at the IEEE Conference on Visualization (IEEE VIS 2015), Chicago, Illinois, USA*. July 2015.
- [54] H. Hauser, F. Ledermann and H. Doleisch. ‘Angular brushing for extended parallel coordinates’. In: *Proceedings of the IEEE Symposium on Information Visualization*. 2002, pp. 127–130.
- [55] N. Henry and J. d. Fekete. ‘MatrixExplorer: a Dual-Representation System to Explore Social Networks’. In: *IEEE Transactions on Visualization and Computer Graphics* 12.5 (Sept. 2006), pp. 677–684. ISSN: 1077-2626. DOI: 10.1109/TVCG.2006.160.
- [56] Fan Hong et al. ‘FLDA: Latent Dirichlet Allocation Based Unsteady Flow Analysis’. In: *Visualization and Computer Graphics, IEEE Transactions on* 20 (Dec. 2014), pp. 2545–2554. DOI: 10.1109/TVCG.2014.2346416.
- [57] Clare E. Horne. *Geometric symmetry in patterns and tilings*. Woodhead Publishing Ltd, 2000. ISBN: 1855734923.
- [58] Eve Ignatius, Hikmet Senay and Jean Favre. ‘An intelligent system for task-specific visualization assistance’. In: *Journal of Visual Languages & Computing* 5.4 (1994), pp. 321–338.
- [59] P. Isenberg et al. ‘Vispubdata.org: A Metadata Collection About IEEE Visualization (VIS) Publications’. In: *IEEE Transactions on Visualization and Computer Graphics* 23.9 (Sept. 2017), pp. 2199–2206. ISSN: 1077-2626. DOI: 10.1109/TVCG.2016.2615308.
- [60] P. Isenberg et al. ‘Visualization as Seen through its Research Paper Keywords’. In: *IEEE Transactions on Visualization and Computer Graphics* 23.1 (Jan. 2017), pp. 771–780. ISSN: 1077-2626. DOI: 10.1109/TVCG.2016.2598827.
- [61] Y. Rogers J. Preece and H. Sharp. *Interaction design: Beyond human-computer interaction*. Springer, 2002.
- [62] Y. Jansen and P. Dragicevic. ‘An Interaction Model for Visualizations Beyond The Desktop’. In: *IEEE Transactions on Visualization and Computer Graphics* 19.12 (Dec. 2013), pp. 2396–2405. ISSN: 1077-2626. DOI: 10.1109/TVCG.2013.134.

- [63] Gaetano Kanizsa. *Organization in vision: Essays on Gestalt perception*. Praeger Publishers, 1979.
- [64] N. Kerracher and J. Kennedy. ‘Constructing and Evaluating Visualisation Task Classifications: Process and Considerations’. In: *Computer Graphics Forum* 36.3 (2017), pp. 47–59. DOI: 10.1111/cgf.13167.
- [65] Adam Kilgarriff et al. ‘The Sketch Engine: ten years on’. In: *Lexicography* (2014), pp. 7–36. DOI: 10.1007/s40607-014-0009-9.
- [66] P. Koytek et al. ‘MyBrush: Brushing and Linking with Personal Agency’. In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (Jan. 2018), pp. 605–615. ISSN: 1077-2626. DOI: 10.1109/TVCG.2017.2743859.
- [67] K. Kucher et al. ‘Visual analysis of stance markers in online social media’. In: *2014 IEEE Conference on Visual Analytics Science and Technology (VAST)*. 2014, pp. 259–260.
- [68] Kostiantyn Kucher and Andreas Kerren. ‘Text visualization techniques: Taxonomy, visual survey, and community insights’. In: *2015 IEEE Pacific Visualization Symposium (PacificVis)*. IEEE. 2015, pp. 117–121.
- [69] Ricardo Langner, Ulrike Kister and Raimund Dachsel. ‘Multiple Coordinated Views at Large Displays for Multiple Users: Empirical Findings on User Behavior, Movements, and Distances’. In: *IEEE Transactions on Visualization and Computer Graphics* 25 (1 Jan. 2019), pp. 608–618. DOI: 10.1109/TVCG.2018.2865235.
- [70] M. Lawrence et al. ‘exploRase: Exploratory Data Analysis of Systems Biology Data’. In: *Coordinated and Multiple Views in Exploratory Visualization, 2006. Proceedings. International Conference on*. July 2006, pp. 14–20. DOI: 10.1109/CMV.2006.7.
- [71] D. J. Lehmann and H. Theisel. ‘Orthographic Star Coordinates’. In: *IEEE Transactions on Visualization and Computer Graphics* 19.12 (2013), pp. 2615–2624.
- [72] Ralph Lengler and Martin J. Eppler. ‘Towards a Periodic Table of Visualization Methods of Management’. In: *Proceedings of the IASTED International Conference on Graphics and Visualization in Engineering*. GVE ’07. Clearwater, Florida: ACTA Press, 2007, pp. 83–88. ISBN: 978-0-88986-627-0.

- [73] Qing Li et al. 'Dynamic query sliders vs. brushing histograms'. In: *CHI '03: CHI '03 extended abstracts on Human factors in computing systems*. Ft. Lauderdale, Florida, USA: ACM Press, 2003, pp. 834–835. ISBN: 1-58113-637-4.
- [74] Zhicheng Liu et al. 'Data Illustrator: Augmenting Vector Design Tools with Lazy Data Binding for Expressive Visualization Authoring'. In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. CHI'18. Montreal QC, Canada: Association for Computing Machinery, 2018, pp. 1–13. ISBN: 9781450356206. DOI: 10.1145/3173574.3173697.
- [75] Andrea Lodi, Silvano Martello and Michele Monaci. 'Two-dimensional packing problems: A survey'. In: *European Journal of Operational Research* 141.2 (2002), pp. 241–252. ISSN: 0377-2217. DOI: 10.1016/S0377-2217(02)00123-6.
- [76] Jerry Lohse et al. 'Classifying visual knowledge representations: a foundation for visualization research'. In: *Proceedings of the First IEEE Conference on Visualization*. IEEE. 1990, pp. 131–138.
- [77] Doug Lowe. *PowerPoint 2019 For Dummies*. Dummies, 2019. ISBN: 1119514223.
- [78] A. MacEachren et al. 'Exploring high-D spaces with multiform matrices and small multiples'. In: *IEEE Symposium on Information Visualization 2003*. Oct. 2003, pp. 31–38. DOI: 10.1109/INFVIS.2003.1249006.
- [79] K. Madhavan et al. 'DIA2: Web-based Cyberinfrastructure for Visual Analysis of Funding Portfolios'. In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (Dec. 2014), pp. 1823–1832. ISSN: 1077-2626. DOI: 10.1109/TVCG.2014.2346747.
- [80] Tahir Mahmood et al. 'Building multiple coordinated spaces for effective immersive analytics through distributed cognition'. In: *2018 International Symposium on Big Data Visual and Immersive Analytics (BDVA)*. IEEE. 2018, pp. 1–11.
- [81] Hayder M. Al-manee and Jonathan C. Roberts. 'Study of Multiple View Layout Strategies in Visualisation'. In: *Posters presented at the IEEE Conference on Visualization (IEEE VIS 2018), Berlin, Germany*. Oct. 2018.
- [82] Hayder M. Al-manee and Jonathan C. Roberts. 'Towards Quantifying Multiple View Layouts in Visualisation as Seen from Research Publications'. In: *2019*

- IEEE Visualization Conference (VIS)*. Oct. 2019, pp. 121–121. DOI: 10.1109/VISUAL.2019.8933655.
- [83] Allen R. Martin and Matthew O. Ward. ‘High Dimensional Brushing for Interactive Exploration of Multivariate Data’. In: *VIS ’95: Proceedings Visualization ’95*. IEEE Computer Society, 1995, p. 271. ISBN: 0-8186-7187-4.
 - [84] W. N. Martin and J. K. Aggarwal. ‘Volumetric Descriptions of Objects from Multiple Views’. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-5.2 (Mar. 1983), pp. 150–158. ISSN: 0162-8828. DOI: 10.1109/TPAMI.1983.4767367.
 - [85] K. Matkovic et al. ‘ComVis: A Coordinated Multiple Views System for Prototyping New Visualization Technology’. In: *2008 12th International Conference Information Visualisation*. July 2008, pp. 215–220. DOI: 10.1109/IV.2008.87.
 - [86] John Alan McDonald. ‘Interactive graphics for data analysis’. PhD thesis. Citeseer, 1982.
 - [87] John Alan McDonald, Werner Stuetzle and Andreas Buja. *Painting multiple views of complex objects*. 1990.
 - [88] Emile Morse, Michael Lewis and Kai A Olsen. ‘Evaluating visualizations: using a taxonomic guide’. In: *International Journal of Human-Computer Studies* 53.5 (2000), pp. 637–662.
 - [89] Galileo Mark Namata et al. ‘A Dual-view Approach to Interactive Network Visualization’. In: *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*. CIKM ’07. Lisbon, Portugal: ACM, 2007, pp. 939–942. ISBN: 978-1-59593-803-9. DOI: 10.1145/1321440.1321580.
 - [90] Deon Nel, Leyland Pitt and Trevor Webb. ‘Using Chernoff faces to portray service quality data’. In: *Journal of Marketing Management* 10.1-3 (1994), pp. 247–255.
 - [91] Jakob Nielsen and Kara Pernice. *Eyetracking Web Usability*. Thousand Oaks, CA, USA: New Riders Publishing, 2009. ISBN: 9780321498366.
 - [92] Donald A Norman. *User centered system design: New perspectives on human-computer interaction*. CRC Press, 1986.

- [93] Donald A. Norman. ‘Design Rules Based on Analyses of Human Error’. In: *Commun. ACM* 26.4 (Apr. 1983), pp. 254–258. ISSN: 0001-0782. DOI: 10.1145/2163.358092.
- [94] Chris North and Ben Shneiderman. ‘Snap-together Visualization: A User Interface for Coordinating Visualizations via Relational Schemata’. In: *Proceedings of the Working Conference on Advanced Visual Interfaces. AVI ’00*. Palermo, Italy: ACM, 2000, pp. 128–135. ISBN: 1-58113-252-2. DOI: 10.1145/345513.345282.
- [95] Ruben Olsen. *OmniGraffle 5 Diagramming Essentials: Create Better Diagrams with Less Effort Using OmniGraffle*. Packt Publishing Ltd, 2010.
- [96] Xufang Pang et al. ‘Directing user attention via visual flow on web designs’. In: *ACM Transactions on Graphics (TOG)* 35.6 (2016), p. 240.
- [97] J. H. Park et al. ‘C2A: Crowd consensus analytics for virtual colonoscopy’. In: *2016 IEEE Conference on Visual Analytics Science and Technology (VAST)*. Oct. 2016, pp. 21–30. DOI: 10.1109/VAST.2016.7883508.
- [98] H. Piringer, R. Kosara and H. Hauser. ‘Interactive focus+context visualization with linked 2D/3D scatterplots’. In: *Proceedings. Second International Conference on Coordinated and Multiple Views in Exploratory Visualization, 2004*. July 2004, pp. 49–60. DOI: 10.1109/CMV.2004.1319526.
- [99] M. Plumlee and C. Ware. ‘Integrating multiple 3D views through frame-of-reference interaction’. In: *Proceedings International Conference on Coordinated and Multiple Views in Exploratory Visualization - CMV 2003* -. July 2003, pp. 34–43. DOI: 10.1109/CMV.2003.1215001.
- [100] Z. Qu and J. Hullman. ‘Keeping Multiple Views Consistent: Constraints, Validations, and Exceptions in Visualization Authoring’. In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (Jan. 2018), pp. 468–477. ISSN: 1077-2626. DOI: 10.1109/TVCG.2017.2744198.
- [101] François Quesnay. *Tableau oeconomique*. Macmillan, 1894.
- [102] D. Ren, T. Höllerer and X. Yuan. ‘iVisDesigner: Expressive Interactive Design of Information Visualizations’. In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (Dec. 2014), pp. 2092–2101. ISSN: 1077-2626. DOI: 10.1109/TVCG.2014.2346291.
- [103] Gaëlle Richer, Romain Bourqui and David Auber. ‘CorFish: Coordinating Emphasis Across Multiple Views Using Spatial Distortion’. In: *2019 IEEE*

- Pacific Visualization Symposium (PacificVis)*. 2019, pp. 1–10. DOI: 10.1109/PacificVis.2019.00009.
- [104] J. Roberts, N. Boukhelifa and P. Rodgers. ‘Multiform glyph based web search result visualization’. In: *Proceedings Sixth International Conference on Information Visualisation*. 2002, pp. 549–554. DOI: 10.1109/IV.2002.1028828.
 - [105] J. C. Roberts. ‘On encouraging multiple views for visualization’. In: *Proceedings IEEE Conference on Information Visualization*. July 1998, pp. 8–14. DOI: 10.1109/IV.1998.694193.
 - [106] J. C. Roberts et al. ‘Visualization beyond the Desktop—the Next Big Thing’. In: *IEEE Computer Graphics and Applications* 34.6 (Nov. 2014), pp. 26–34. ISSN: 0272-1716. DOI: 10.1109/MCG.2014.82.
 - [107] J. C. Roberts et al. ‘The Explanatory Visualization Framework: An Active Learning Framework for Teaching Creative Computing Using Explanatory Visualizations’. In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (Jan. 2018), pp. 791–801. ISSN: 1077-2626. DOI: 10.1109/TVCG.2017.2745878.
 - [108] J. C. Roberts et al. ‘Visualisation Approaches for Corpus Linguistics: Towards Visual Integration of Data-Driven Learning’. In: *3rd Workshop on Visualization for the Digital Humanities, at IEEE VIS, Berlin, Germany*. Oct. 2018.
 - [109] J. C. Roberts et al. ‘Multiple Views: different meanings and collocated words’. In: *Computer Graphics Forum (EuroVis2019, Porto, Portugal)* (2019). Accepted for publication.
 - [110] Jonathan C Roberts. ‘Exploratory visualization with multiple linked views’. In: *Exploring geovisualization*. Elsevier, 2005, pp. 159–180.
 - [111] Jonathan C. Roberts. ‘Waltz - An exploratory visualization tool for volume data, using multiform abstract displays’. In: *Visual Data Exploration and Analysis V, Proceedings of SPIE*. Ed. by Robert F. Erbacher and Alex Pang. Vol. 3298. Bellingham, Washington, USA, 1998, pp. 112–122. DOI: 10.1117/12.309533.
 - [112] Jonathan C. Roberts. ‘Multiple-View and Multiform Visualization’. In: *Visual Data Exploration and Analysis VII, Proceedings of SPIE*. Ed. by Robert Erbacher et al. Vol. 3960. IS&T and SPIE. Jan. 2000, pp. 182–196. DOI: 10.1117/12.378894.

- [113] Jonathan C. Roberts. ‘Visualization display models – ways to classify visual representations’. In: *International Journal of Computer Integrated Design and Construction* 2.4 (Dec. 2000), pp. 241–250.
- [114] Jonathan C. Roberts. ‘Visualization display models-ways to classify visual representations’. In: *International Journal of Computer Integrated Design and Construction* 2.4 (Dec. 2000), pp. 241–250.
- [115] Jonathan C. Roberts. ‘Exploratory Visualization Using Bracketing’. In: *Proceedings of the Working Conference on Advanced Visual Interfaces*. AVI ’04. Gallipoli, Italy: ACM, 2004, pp. 188–192. ISBN: 1581138679. DOI: 10.1145/989863.989893.
- [116] Jonathan C. Roberts. ‘State of the Art: Coordinated Multiple Views in Exploratory Visualization’. In: *Fifth International Conference on Coordinated and Multiple Views in Exploratory Visualization (CMV 2007)*. July 2007, pp. 61–71. DOI: 10.1109/CMV.2007.20.
- [117] Jonathan C. Roberts, Christopher Headleand and Panagiotis D. Ritsos. ‘Sketching Designs Using the Five Design-Sheet Methodology’. In: *IEEE Transactions on Visualization and Computer Graphics* (Jan. 2016). ISSN: 1077-2626. DOI: 10.1109/TVCG.2015.2467271.
- [118] Jonathan C. Roberts, Christopher J. Headleand and Panagiotis D. Ritsos. *Five Design-Sheets – Creative design and sketching in Computing and Visualization*. SpringerNature, 2017. ISBN: 978-3-319-55626-0. DOI: 10.1007/978-3-319-55627-7.
- [119] Jonathan C. Roberts et al. ‘Visualization beyond the Desktop–the Next Big Thing’. In: *Computer Graphics and Applications, IEEE* 34.6 (Nov. 2014), pp. 26–34. ISSN: 0272-1716. DOI: 10.1109/MCG.2014.82.
- [120] Jonathan C. Roberts et al. ‘Multiple Views: different meanings and collocated words’. English. In: *Computer Graphics Forum* (Mar. 2019). ISSN: 0167-7055.
- [121] M.A. Rosenman and J.S. Gero. ‘Modelling multiple views of design objects in a collaborative cad environment’. In: *Computer-Aided Design* 28.3 (1996). Artificial Intelligence in Computer-Aided Design, pp. 193–205. ISSN: 0010-4485. DOI: 10.1016/0010-4485(96)86822-9.
- [122] Greg Ross and Matthew Chalmers. ‘A Visual Workspace for Hybrid Multidimensional Scaling Algorithms’. In: *Proceedings of the Ninth Annual*

- IEEE Conference on Information Visualization. INFOVIS'03. Seattle, Washington: IEEE Computer Society, 2003, pp. 91–96. ISBN: 0-7803-8154-8.*
- [123] Grzegorz Rozenberg and Arto Salomaa. *The book of L*. Springer Science & Business Media, 2012.
 - [124] A. Sarikaya and M. Gleicher. ‘Scatterplots: Tasks, Data, and Designs’. In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (Jan. 2018), pp. 402–412. ISSN: 1077-2626. DOI: 10.1109/TVCG.2017.2744184.
 - [125] Arvind Satyanarayan and Jeffrey Heer. ‘Lyra: An interactive visualization design environment’. In: *Computer Graphics Forum*. Vol. 33. 3. Wiley Online Library, 2014, pp. 351–360.
 - [126] Arvind Satyanarayan et al. ‘Vega-lite: A grammar of interactive graphics’. In: *IEEE transactions on visualization and computer graphics* 23.1 (2016), pp. 341–350.
 - [127] Arvind Satyanarayan et al. ‘Vega-Lite: A Grammar of Interactive Graphics’. In: *IEEE Transactions on Visualization and Computer Graphics* 23.1 (Jan. 2017), pp. 341–350. ISSN: 1077-2626. DOI: 10.1109/TVCG.2016.2599030.
 - [128] M. Sedlmair, M. D. Meyer and T. Munzner. ‘Design Study Methodology: Reflections from the Trenches and the Stacks’. In: *IEEE Transactions on Visualization and Computer Graphics* 18.12 (2012), pp. 2431–2440.
 - [129] E. Segel and J. Heer. ‘Narrative Visualization: Telling Stories with Data’. In: *IEEE Transactions on Visualization and Computer Graphics* 16.6 (Nov. 2010), pp. 1139–1148. ISSN: 1077-2626. DOI: 10.1109/TVCG.2010.179.
 - [130] Q. Shen et al. ‘StreetVizor: Visual Exploration of Human-Scale Urban Forms Based on Street Views’. In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (Jan. 2018), pp. 1004–1013. ISSN: 1077-2626. DOI: 10.1109/TVCG.2017.2744159.
 - [131] Sergei Andreevich Shershakov. ‘VTMine for Visio: Graphical Tool for Modeling in Process Mining’. In: *information systems* 27.2 (2020), pp. 194–217.
 - [132] B. Shneiderman. ‘The eyes have it: a task by data type taxonomy for information visualizations’. In: *Proceedings 1996 IEEE Symposium on Visual Languages*. 1996, pp. 336–343. DOI: 10.1109/VL.1996.545307.
 - [133] B. Shneiderman and C. Plaisant. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Pearson Addison-Wesley, 2017. ISBN: 9781292153926.

- [134] B. Shneiderman and M. Wattenberg. 'Ordered treemap layouts'. In: *IEEE Symposium on Information Visualization, 2001. INFOVIS 2001*. 2001, pp. 73–78. DOI: 10.1109/INFVIS.2001.963283.
- [135] Ben Shneiderman. 'Tree Visualization with Tree-Maps: 2-d Space-Filling Approach'. In: *ACM Trans. Graph.* 11.1 (Jan. 1992), pp. 92–99. ISSN: 0730-0301. DOI: 10.1145/102377.115768.
- [136] H. Siirtola. 'Combining parallel coordinates with the reorderable matrix'. In: *Proceedings International Conference on Coordinated and Multiple Views in Exploratory Visualization - CMV 2003* -. July 2003, pp. 63–74. DOI: 10.1109/CMV.2003.1215004.
- [137] Harri Siirtola and Erkki Mäkinen. 'Constructing and Reconstructing the Reorderable Matrix'. In: *Information Visualization* 4.1 (Mar. 2005), pp. 32–48. ISSN: 1473-8716. DOI: 10.1057/palgrave.ivs.9500086.
- [138] John Stasko, Carsten Görg and Zhicheng Liu. 'Jigsaw: supporting investigative analysis through interactive visualization'. In: *Information visualization* 7.2 (2008), pp. 118–132.
- [139] C. Stolte, D. Tang and P. Hanrahan. 'Polaris: a system for query, analysis, and visualization of multidimensional relational databases'. In: *IEEE Transactions on Visualization and Computer Graphics* 8.1 (Jan. 2002), pp. 52–65. ISSN: 1077-2626. DOI: 10.1109/2945.981851.
- [140] Martin Theus. 'Interactive data visualization using Mondrian'. In: *Journal of Statistical Software* 7.11 (2002), pp. 1–9.
- [141] C. Upson et al. 'The application visualization system: a computational environment for scientific visualization'. In: *IEEE Computer Graphics and Applications* 9.4 (1989). ISSN: 0272-1716. DOI: 10.1109/38.31462.
- [142] J. J. Van Wijk and H. Van de Wetering. 'Cushion treemaps: visualization of hierarchical information'. In: *Proceedings 1999 IEEE Symposium on Information Visualization (InfoVis'99)*. 1999, pp. 73–78. DOI: 10.1109/INFVIS.1999.801860.
- [143] F. B. Viegas et al. 'ManyEyes: a Site for Visualization at Internet Scale'. In: *IEEE Transactions on Visualization and Computer Graphics* 13.6 (Nov. 2007), pp. 1121–1128. ISSN: 1077-2626. DOI: 10.1109/TVCG.2007.70577.
- [144] Michelle Q. Wang Baldonado, Allison Woodruff and Allan Kuchinsky. 'Guidelines for Using Multiple Views in Information Visualization'. In:

- Proceedings of the Working Conference on Advanced Visual Interfaces. AVI '00*. Palermo, Italy: ACM, 2000, pp. 110–119. ISBN: 1-58113-252-2. DOI: 10.1145/345513.345271.
- [145] Matthew O. Ward. 'XmdvTool: integrating multiple methods for visualizing multivariate data'. In: *VIS '94: Proceedings Visualization '94*. Washinton, D.C.: IEEE Computer Society Press, 1994, pp. 326–333. ISBN: 0-7803-2521-4.
 - [146] Matthew O. Ward. 'Creating and Manipulating N-Dimensional Brushes'. In: *Proceedings of Joint Statistical Meeting*. Baltimore, USA, 1997, pp. 6–14.
 - [147] Colin Ware. *Information Visualization: Perception for Design*. Amsterdam: Elsevier (Morgan Kaufmann), 2012.
 - [148] C. Weaver. 'Building Highly-Coordinated Visualizations in Improvise'. In: *IEEE Symposium on Information Visualization*. 2004, pp. 159–166. DOI: 10.1109/INFVIS.2004.12.
 - [149] C. Weaver. 'Multidimensional visual analysis using cross-filtered views'. In: *2008 IEEE Symposium on Visual Analytics Science and Technology*. Oct. 2008, pp. 163–170. DOI: 10.1109/VAST.2008.4677370.
 - [150] C. Weaver. 'Cross-Filtered Views for Multidimensional Visual Analysis'. In: *IEEE Transactions on Visualization and Computer Graphics* 16.2 (Mar. 2010), pp. 192–204. ISSN: 1077-2626. DOI: 10.1109/TVCG.2009.94.
 - [151] Stephen Wehrend and Clayton Lewis. 'A problem-oriented classification of visualization techniques'. In: *Proceedings of the First IEEE Conference on Visualization: Visualization90*. IEEE. 1990, pp. 139–143.
 - [152] Leland Wilkinson. *The Grammar of Graphics (Statistics and Computing)*. Berlin, Heidelberg: Springer-Verlag, 2005. ISBN: 0387245448. DOI: 10.5555/1088896.
 - [153] Chauncey Wilson. *User Interface Inspection Methods. A User-Centered Design Method*. Elsevier (Morgan Kaufmann), 2014. ISBN: 9780124103917.
 - [154] Marco A Winckler, Philippe Palanque and Carla MDS Freitas. 'Tasks and scenario-based evaluation of information visualization techniques'. In: *Proceedings of the 3rd annual conference on Task models and diagrams*. 2004, pp. 165–172.
 - [155] Jessica Wojciechowski, Ashley M. Hopkins and Richard Neil Upton. 'Interactive pharmacometric applications using R and the shiny package'. In: *CPT: pharmacometrics & systems pharmacology* 4.3 (2015), pp. 146–159.

- [156] Pak Chung Wong and R. Daniel Bergeron. 'Multiresolution multidimensional wavelet brushing'. In: *VIS '96: Proceedings Visualization '96*. San Francisco, California, United States: IEEE Computer Society Press, 1996, pp. 141–148. ISBN: 0-89791-864-9.
- [157] Pak Chung Wong and R. Daniel Bergeron. 'Brushing Techniques for Exploring Volume Datasets'. In: *IEEE Visualization '97*. Ed. by Roni Yagel and Hans Hagen. 1997, pp. 429–432.
- [158] K. Wongsuphasawat et al. 'Voyager: Exploratory Analysis via Faceted Browsing of Visualization Recommendations'. In: *IEEE Transactions on Visualization and Computer Graphics* 22.1 (Jan. 2016), pp. 649–658. ISSN: 1077-2626. DOI: 10.1109/TVCG.2015.2467191.
- [159] Y. Wu et al. 'OpinionFlow: Visual Analysis of Opinion Diffusion on Social Media'. In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (Dec. 2014), pp. 1763–1772. ISSN: 1077-2626. DOI: 10.1109/TVCG.2014.2346920.
- [160] M. Adil Yalcin, Niklas Elmqvist and Benjamin B. Bederson. 'Keshif: Rapid and Expressive Tabular Data Exploration for Novices'. In: *IEEE Transactions on Visualization & Computer Graphics* (19th May 2017).
- [161] Mehmet Adil Yalçın, Niklas Elmqvist and Benjamin B. Bederson. 'Keshif: Rapid and Expressive Tabular Data Exploration for Novices'. In: *IEEE Transactions on Visualization and Computer Graphics* 24.8 (2018), pp. 2339–2352. DOI: 10.1109/TVCG.2017.2723393.
- [162] J. Zhang et al. 'Vis4Heritage: Visual Analytics Approach on Grotto Wall Painting Degradations'. In: *IEEE Transactions on Visualization and Computer Graphics* 19.12 (Dec. 2013), pp. 1982–1991. ISSN: 1077-2626. DOI: 10.1109/TVCG.2013.219.
- [163] Nannan Zhang. 'Application of Computer Graphics and Image Software in Marine Graphic Design'. In: *Journal of Coastal Research* 106.SI (2020), pp. 600–604.
- [164] Michelle X Zhou and Steven K Feiner. 'Visual task characterization for automated visual discourse synthesis'. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. 1998, pp. 392–399.
- [165] John Zukowski. *Java AWT reference*. O'Reilly Media, 1997. ISBN: 978-1565922402.