Concepts and Techniques for 3D-Embedded Treemaps and their Application to Software Visualization

Dissertation

in partial fulfillment for the academic degree
"doctor rerum naturalium" (Dr. rer. nat.)

submitted by **Daniel Limberger**

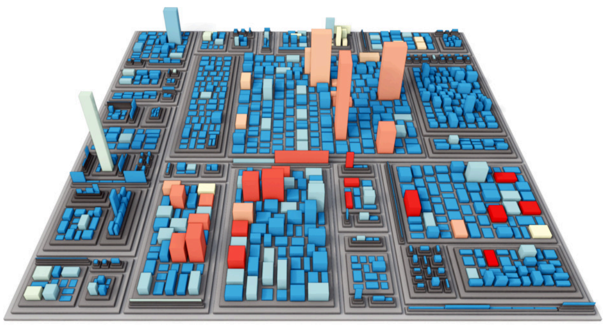
Potsdam, Germany , 22. March 2023

supervised by Prof. Dr. Jürgen Döllner
of the Computer Graphics Systems Group at the

Hasso Plattner Institute | Faculty of Digital Engineering | University of Potsdam

"By visualizing information, we turn it into a landscape that you can explore with your eyes: [...] a sort of information map. And when you're lost in information, an information map is kind of useful."

David McCandless [167]



Abstract

This thesis addresses concepts and techniques for interactive visualization of hierarchical data using treemaps. It explores (1) how treemaps can be embedded in 3D space to improve their information content and expressiveness, (2) how the readability of treemaps can be improved using level-of-detail and degree-of-interest techniques, and (3) how to design and implement a software framework for the real-time web-based rendering of treemaps embedded in 3D. With a particular emphasis on their application, use cases from software analytics are taken to test and evaluate the presented concepts and techniques.

Concerning the first challenge, this thesis shows that a 3D attribute space offers enhanced possibilities for the visual mapping of data compared to classical 2D treemaps. In particular, embedding in 3D allows for improved implementation of visual variables (e.g., by sketchiness and color weaving), provision of new visual variables (e.g., by physically based materials and in situ templates), and integration of visual metaphors (e.g., by reference surfaces and renderings of natural phenomena) into the three-dimensional representation of treemaps.

For the second challenge—the readability of an information visualization—the work shows that the generally higher visual clutter and increased cognitive load typically associated with three-dimensional information representations can be kept low in treemap-based representations of both small and large hierarchical datasets. By introducing an adaptive level-of-detail technique, we cannot only declutter the visualization results, thereby reducing cognitive load and mitigating occlusion problems, but also summarize and highlight relevant data. Furthermore, this approach facilitates automatic labeling, supports the emphasis on data outliers, and allows visual variables to be adjusted via degree-of-interest measures.

The third challenge is addressed by developing a real-time rendering framework with WebGL and accumulative multi-frame rendering. The framework removes hardware constraints and graphics API requirements, reduces interaction response times, and simplifies high-quality rendering. At the same time, the implementation effort for a web-based deployment of treemaps is kept reasonable.

The presented visualization concepts and techniques are applied and evaluated for use cases in software analysis. In this domain, data about software systems, especially about the state and evolution of the source code, does not have a descriptive appearance or natural geometric mapping, making information visualization a key technology here. In particular, software source code can be visualized with treemap-based approaches because of its inherently hierarchical structure. With treemaps embedded in 3D, we can create interactive software maps that visually map, software metrics, software developer activities, or information about the evolution of software systems alongside their hierarchical module structure.

Discussions on remaining challenges and opportunities for future research for 3D-embedded treemaps and their applications conclude the thesis.

Zusammenfassung

Diese Doktorarbeit behandelt Konzepte und Techniken zur interaktiven Visualisierung hierarchischer Daten mit Hilfe von Treemaps. Sie untersucht (1), wie Treemaps im 3D-Raum eingebettet werden können, um ihre Informationsinhalte und Ausdrucksfähigkeit zu verbessern, (2) wie die Lesbarkeit von Treemaps durch Techniken wie Level-of-Detail und Degree-of-Interest verbessert werden kann, und (3) wie man ein Software-Framework für das Echtzeit-Rendering von Treemaps im 3D-Raum entwirft und implementiert. Dabei werden Anwendungsfälle aus der Software-Analyse besonders betont und zur Verprobung und Bewertung der Konzepte und Techniken verwendet.

Hinsichtlich der ersten Herausforderung zeigt diese Arbeit, dass ein 3D-Attributraum im Vergleich zu klassischen 2D-Treemaps verbesserte Möglichkeiten für die visuelle Kartierung von Daten bietet. Insbesondere ermöglicht die Einbettung in 3D eine verbesserte Umsetzung von visuellen Variablen (z. B. durch Skizzenhaftigkeit und Farbverwebungen), die Bereitstellung neuer visueller Variablen (z. B. durch physikalisch basierte Materialien und In-situ-Vorlagen) und die Integration visueller Metaphern (z. B. durch Referenzflächen und Darstellungen natürlicher Phänomene) in die dreidimensionale Darstellung von Treemaps.

Für die zweite Herausforderung – die Lesbarkeit von Informationsvisualisierungen – zeigt die Arbeit, dass die allgemein höhere visuelle Unübersichtlichkeit und die damit einhergehende, erhöhte kognitive Belastung, die typischerweise mit dreidimensionalen Informationsdarstellungen verbunden sind, in Treemap-basierten Darstellungen sowohl kleiner als auch großer hierarchischer Datensätze niedrig gehalten werden können. Durch die Einführung eines adaptiven Level-of-Detail-Verfahrens lassen sich nicht nur die Visualisierungsergebnisse übersichtlicher gestalten, die kognitive Belastung reduzieren und Verdeckungsprobleme verringern, sondern auch relevante Daten zusammenfassen und hervorheben. Darüber hinaus erleichtert dieser Ansatz eine automatische Beschriftung, unterstützt die Hervorhebung von Daten-Ausreißern und ermöglicht die Anpassung von visuellen Variablen über Degree-of-Interest-Maße.

Die dritte Herausforderung wird durch die Entwicklung eines Echtzeit-Rendering-Frameworks mit WebGL und akkumulativem Multi-Frame-Rendering angegangen. Das Framework hebt mehrere Hardwarebeschränkungen und Anforderungen an die Grafik-API auf, verkürzt die Reaktionszeiten auf Interaktionen und vereinfacht qualitativ hochwertiges Rendering. Gleichzeitig wird der Implementierungsaufwand für einen webbasierten Einsatz von Treemaps geringgehalten.

Die vorgestellten Visualisierungskonzepte und -techniken werden für Anwendungsfälle in der Softwareanalyse eingesetzt und evaluiert. In diesem Bereich haben Daten über Softwaresysteme, insbesondere über den Zustand und die Evolution des Quellcodes, keine anschauliche Erscheinung oder natürliche geometrische Zuordnung, so dass die Informationsvisualisierung hier eine Schlüsseltechnologie darstellt. Insbesondere Softwarequellcode kann aufgrund seiner inhärenten hierarchischen Struktur mit Hilfe von Treemap-basierten Ansätzen visualisiert werden. Mit in 3D-eingebetteten Treemaps können wir interaktive Softwarelagekarten erstellen, die z. B. Softwaremetriken, Aktivitäten von Softwareentwickler*innen und Informationen über die Evolution von Softwaresystemen in ihrer hierarchischen Modulstruktur abbilden und veranschaulichen.

Diskussionen über verbleibende Herausforderungen und Möglichkeiten für zukünftige Forschung zu 3D-eingebetteten Treemaps und deren Anwendungen schließen die Arbeit ab.

Table of Contents

1 **Introduction** 1

1.1 Visualization of Hierarchical Data 1
1.2 Charting Software Systems 2
1.3 Thesis Statements and Objectives 3

2 **Fundamentals of 3D-Embedded Treemaps** 5

2.1 Data Characteristics 7
2.1.1 Tree-Structured Data 7
2.1.2 Temporal Data 8
2.1.3 Multivariate Data 9

2.2 Treemap Visualization Pipeline 9
2.2.1 Taxonomy of Treemaps 10
2.2.2 Treemap Layout 11
2.2.3 Attribute and Reference Space 12
2.2.4 Visual Variables 12
2.2.5 Variateness and Expressiveness of Visualizations . 13
2.2.6 Designations of Treemaps in $\mathcal{A}^3 \oplus \mathcal{R}^2$ 14
2.2.7 Alternatives to 2D Image Synthesis 15

2.3 Visualization Process 15
2.4 Software Cartography using Treemaps 18
2.4.1 Specialization of the Term Software Map 18
2.4.2 General Delimitation of Research Contributions . . 19

3 **Visual Variables for 3D-Embedded Treemaps** 21

3.1 Visual Variables of $\mathcal{A}^2$ and $\mathcal{A}^3$ 22
3.1.1 Visualization Mapping and Rendering 23

3.1.2 Visual Variables in $\mathcal{A}^3$ 23

3.2 Sketchy Contours and Surface Hatching 26
3.2.1 Sketchy Outlines for Interactive Visualization 26
3.2.2 Surface Hatching for Interactive Visualization 28
3.2.3 Sketchiness as a Visual Variable in $\mathcal{A}^3$ 29

3.3 Physically-based Materials and Phenomena 30
3.3.1 Physically-based Materials for Visualization 31
3.3.2 Weather Phenomena for Visualization 33

3.4 In-Situ Templates 33
3.4.1 Two-State and Multi-State Mappings 34
3.4.2 Two-State Height Mapping 35
3.4.3 Two-State Color Mapping 36
3.4.4 Two-State Height and Color Mapping 37

3.5 Animated Procedural Textures 38
3.5.1 Transition, Animation, and Change 39
3.5.2 Procedural Patterns for Animated Change Display . 40
3.5.3 Pattern Characteristics 42
3.5.4 Pattern Composition 44
3.5.5 Animation Control 44

3.6 Value-added Adaptations for 3D-Embeddings 45
3.6.1 Height Mapping for Inner-Nodes 45
3.6.2 Height-based Filtering using Reference Surfaces . . 46
3.6.3 Resolution of Visual Data Encodings 47
3.6.4 Addressing Non-Visual Challenges 47

4 **Level-of-Detail and Labeling for 3D-Embedded Treemaps** 49

4.1 Dynamic Aggregation of Nodes 50
4.1.1 Degree-of-Interest Scoring of Nodes 50
4.1.2 Score Propagation and Processing 51

4.2 Visual Display of Aggregates 53
4.2.1 Aggregation Operators for Color and Height 54
4.2.2 Nesting Level Contouring 55
4.2.3 Animated State Transitions for Aggregates 56
4.2.4 Color Weaving 56

4.2.5 Evaluation & Discussion 57

4.3 Dynamic Labeling in 3D-Embedded Treemaps . . . 59
4.3.1 Labeling and Text Rendering 60
4.3.2 Descriptive Label-Placement using OpenLL 61
4.3.3 3D-Embedded Map Legends 63

4.4 Partial 3D-Embedding 63
4.4.1 Node-local Tilt Operator 64
4.4.2 Parameterization of Node-local Tilt 65

5 **Web-based Provisioning of 3D-Embedded Treemaps** 67

5.1 Rendering of Rectangles and Cuboids 68
5.1.1 WebGL-based Rendering of 3D-Embedded Treemaps 69
5.1.2 On the Use of 3rd-Party Renderers 70

5.2 Responsive Accumulative Rendering 71
5.2.1 Composition of a Progressive Frame 72
5.2.2 Progressive Frame Accumulation 74

5.3 Progressive Sampling Strategies 75
5.3.1 Multi-frame Anti-Aliasing 75
5.3.2 Multi-frame Transparency 77
5.3.3 Multi-frame Screen-Space Ambient Occlusion . . . 79
5.3.4 Multi-frame Depth of Field 80

5.4 3D-embedded Treemaps using treemap.ts 81

6 **Software Visualization using 3D-Embedded Treemaps** 85

6.1 Static Source Code Metrics 86
6.2 Map Themes for Visual Software Analytics 87
6.3 Assembling Map Themes 88
6.4 Rendering Performance Evaluation 91

7 **Conclusions and Future Work** 93

7.1 Conclusions 93
7.2 Outlook and Challenges 94
7.3 Closing Remarks 96

1 Introduction

Information visualization is the art and science of presenting complex, abstract data in a clear and effective manner. By leveraging visual representations such as graphs, charts, and maps, information visualization enables users to explore, understand, and communicate information that would otherwise be difficult or impossible to grasp. It faces various challenges, such as how to effectively handle large amounts of data, how to represent complex relationships and hierarchies, and how to balance simplicity and detail. Powerful tools have emerged that can "transform data into information and information into insights"[1.1] and have the potential to help us to make better decisions, discover new patterns and relationships, and communicate information more effectively.

1.1 Visualization of Hierarchical Data

"Humans arrange information hierarchically and use hierarchical methods for reasoning. 'Hierarchization' is one of the major conceptual mechanism to model the world." [236] It should not surprise that hierarchical data is ubiquitous and fundamental to almost all systems and application domains. Our need to explore this data is emphasized by the sheer amount of different ways that have been created to visualize tree-structured data [209]. These techniques help to share, shape, and re-use mental images of facets of data, aiding communication and exploration. Given the wide range of existing techniques, we focus our efforts on refining and extending the capabilities of treemaps. They offer unique advantages for exploring and understanding complex data sets, making them a natural choice for our research goals. By building on the strengths of treemaps, we aim to increase their versatility and help researchers and practitioners make sense of large, tree-structured, multivariate data.

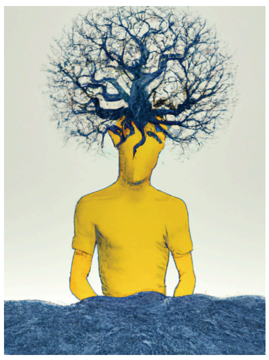


Figure 1.1: A drawing of a person having used hierarchical thinking to cope with complexity, created using DALL·E.

Treemaps use nested rectangles to represent hierarchical data; each rectangle represents a specific portion of the data, with the area of the rectangle corresponding to the magnitude of the data it represents. This allows users to identify trends and patterns that might be difficult to discern from raw data alone. Treemaps leverage our ability to create cognitive maps [7, 86, 153, 171], which are non-egocentric, coherent representations of our

[1.1] Carly Fiorina. *Information: The Currency of the Digital Age.* hp.com/hpinfo/execteam /speeches/fiorina/04openworld.html. Dec. 2004.

surroundings that we create in our minds (Figure 1.1). Such maps help us navigate complex environments using visual cues and spatial relationships. The added data encoding employed by treemaps, e.g., using size, color, or texture, allows users to tap into complex data intuitively and gain insights into trends, patterns, outliers, anomalies, and similarities, all while preserving the relationships among data elements.

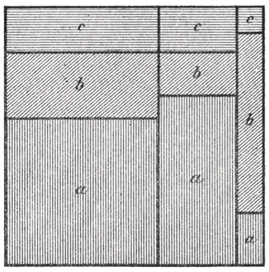


Figure 1.2: "Flächen-Diagramm mit zweimaliger Untertheilung", an area diagram with twofold subdivision from 1877 [259]. The area is split from left to right, then, per column, from bottom to top. For example, the *b* rectangle on the right represents 70 of its column's 100 of the whole square's 1 000 units.

Although the space-filling approach of treemaps is known to information visualization for at least a century (Figure 1.2); it was famously reintroduced in 1991 [125] to visualize the disk space occupied by files on a computer. Since then, space-filling treemaps have been applied to many domains, and their versatility and algorithmic accessibility have made them an appealing choice for visualizing data across various domains. For example, the entire footprint of a rectangular treemap is a single rectangle representing the *root node* of the depicted tree-structured data. When splitting this rectangle recursively according to the number of child nodes, eventually, only leaf nodes (nodes with no children) are left, each represented by a single rectangle. Data values associated to these nodes are mapped to the rectangles' appearance, e.g., area, color, and texture [25, 44]. By leveraging existing user experiences with popular map services, treemaps enable users to interact and navigate as well as locate points of interest within the data in a way that is intuitive and familiar.

1.2 Charting Software Systems

Visual software analytics was chosen as application domain to test and evaluate this thesis's findings. It focuses on the development and use of visualizations to help understand and analyze software systems. It involves the application of data visualization techniques to software engineering tasks (*software visualization*), such as software testing, debugging, and maintenance, as well as the exploration of software data for the purpose of improving software quality and performance. The goal of visual software analytics and software visualization in particular, is to provide developers, testers, and other stakeholders with intuitive and effective ways to gain insights into the behavior, structure, and evolution of software systems.

Sourcing and mining large amounts of test data are inherently easy in software engineering processes. Moreover, *software system and software engineering data* (SWSE data) caters directly to our needs in that it is typically large, tree-structured, multivariate, and has no intrinsic gestalt, i.e., primarily abstract. Software and its associated data are ubiquitous in society, permeating virtually all technological innovations and social activities. Jensen Huang, co-founder and CEO of Nvidia, predicted that "software is eating the world, but AI is going to eat software."[1.2] Software development and maintenance have been disrupted by cloud-based artificial intelligence tools such as GitHub's Copilot [184] and, more recently, OpenAI's ChatGPT [183]. Having the means to facilitate both actionable awareness and

[1.2]Jensen Huang as interviewed by Tom Simonite. "Nvidia CEO: Software Is Eating the World, but AI Is Going to Eat Software". In: *Technology Review* (May 2017). technologyreview.com/2017/05/12/151722.

in-depth comprehension of software systems and software engineering efforts has become essential to keep up with AI-based assistants.

Charting software systems using treemaps can help to grasp the abstract nature of SWSE data. It can shape our understanding of its complex structures and enable us to explore, manipulate, and accurately communicate the underlying data in more detail. In the spirit of David McCandless, creating software landscapes that we can explore with our eyes (Figure 1.3) seems kind of useful when lost in SWSE data.

Figure 1.3: Conceptual rendering of a software system and coding efforts depicted as 3D city scape.

1.3 Thesis Statements and Objectives

This work originates from the following idea: Suppose there is large tree-structured data, e.g., from software analysis. Furthermore, assume that multiple attributes are associated with each data node, such as software metrics computed for each of the software's source files. Thus, the starting point is large, tree-structured, multivariate data. Now instead of employing different visualizations or linking multiple views, the question arises, to what extent can an interactive visualization method enable the display, exploration, and analysis of such data all by itself?

We chose treemaps as 'vehicle' because they are designed for large, tree-structured, multivariate data at its core [72]. By exploring novel ways of using treemaps with a 3D attribute space, we present an expressive, scalable, and interactive interface for exploring general large, tree-structured, multivariate data, including software system and software engineering data. With this in mind, this thesis has the following objectives concerned with *expressiveness* (E), *scalability* (S), and *responsiveness* (R):

- Extend and explore the data mapping capabilities and offer versatile visual variables by embedding 2D treemaps in 3D.
- Decrease visual clutter and complexity and aid interactive exploration of large data using dynamic aggregation and labeling.
- Make 3D-embedded treemaps available for interactive visualization in common application scenarios by developing a scalable, web-based rendering strategy.

Based on these ideas and objectives, we outline the means and formally derive three hypotheses, $\mathcal{H}_E$, $\mathcal{H}_S$, and $\mathcal{H}_R$.

Expressiveness is approached by transferring selected, existing visual variables of 2D treemaps and introducing novel visual variables to the three-dimensional attribute space of 3D-embedded treemaps. We evaluate the capabilities of selected visual variables and assess their suitability for a meaningful, simultaneous, and unambiguous visual display of data:

Thesis Statement $\mathcal{H}_E$

Embedding 2D treemaps in a 3D attribute space offers additional expressiveness over 2D treemaps by the increased set of visual variables available for the meaningful, simultaneous, unambiguous display of data.

Scalability is achieved by combining two complementary techniques: dynamic level-of-detail and labeling. The former is used to circumvent the common 1:1 mapping of nodes to cuboids, which can lead to visual clutter [199] and increase visual complexity [103]. The latter addresses a "major limiting factor to the widespread use of information visualization[, i.e.,] the difficulty of labeling information abundant displays." [71]. It enables high-quality rendering and dynamic placement of text in 3D, taking, e.g., location, size, and orientation of nodes into account:

Thesis Statement $\mathcal{H}_S$

A level-of-detail technique using node-based scoring allows to adapt the number of graphical elements, and thus to increase the readability of 3D-embedded treemaps by means of labeling, data summary, and emphasis.

Responsiveness is accomplished by employing a progressive rendering strategy to implement the techniques presented in this thesis. The technical requirements and challenges introduced to the image synthesis of 3D visualizations compared to 2D visualizations are fundamentally challenging. Nonetheless, we demonstrate how interactive, dynamic 3D-embedded treemaps can be provisioned in web-based workflows using WebGL:

Thesis Statement $\mathcal{H}_R$

Accumulative, multi-frame rendering enables high-quality and responsive rendering of 3D-embedded treemaps using a web-based graphics API.

The remainder of this work is structured as follows. Chapter 2 provides the theoretical foundation, introduces all necessary terminology, covers relevant basics on data, and introduces a visualization pipeline and process for 3D-embedded treemaps. In chapters 3 to 5, the peer-reviewed and internationally published contributions on which this work is based are highlighted in the context of the respective hypotheses $\mathcal{H}_E$, $\mathcal{H}_S$, and $\mathcal{H}_R$. Applications of selected concepts and techniques are illustrated for visual software analytics in chapter 6. In addition to chapter 2, chapters 3 to 6 reference more specific related work, including results and discussions. Finally, chapter 7 summarizes this thesis and provides an outlook on opportunities for future work.

2 Fundamentals of 3D-Embedded Treemaps

The contents of this chapter are based on the following original publications:

W. Scheibel, **D. Limberger**, and J. Döllner. "Survey of Treemap Layout Algorithms". In: *Proc. ACM VINCI*. 2020 [L19]

W. Scheibel, M. Trapp, **D. Limberger**, and J. Döllner. "A Taxonomy of Treemap Visualization Techniques". In: *Proc. SciTePress IVAPP*. 2020 [L20]

D. Limberger, W. Scheibel, J. Döllner, and M. Trapp. "Visual Variables and Configuration of Software Maps". In: *Journal of Visualization* (2022) [L23]

Treemaps represent a class of fundamental charting techniques that capture data and hierarchy, and transform them into visual representations by partitioning a given space into individual shapes. They are "a device for visualizing statistical facts" [192], well-suited to depict data distributions in different categories. The area of the shapes typically represents a ratio, and the shapes' appearance by means of color, shading, hatching, and texture can depict additional associated data. This idea of mapping data to the area of shapes as part of a diagram dates back to the 1780s [78]. The partitioning is obtained by recursive application of a splitting algorithm and allows for well-controlled nesting of the shapes (*containment*). The "containment property" [125] enables treemaps to represent area-mapped data in hierarchically structured categories.

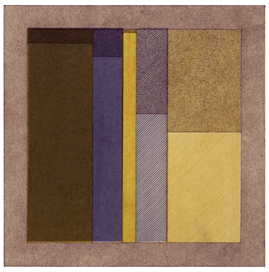


Figure 2.1: A "chart showing the United States [. . .], with distinction of sex [male (non-shaded) and female (shaded)], the ratio between the total population over 10 years of age and the number of persons reported as engaged in each principal class of gainful occupations [agriculture (brown), manufactures and mining (blue), trade and transportation (yellow), personal and professional services (blue hatched),] and also as attending school [(yellow hatched)]." [250]. The proportion of the population not accounted for is shaded in gray.

Early occurrences of treemaps had denotations such as *area charts* [259], *mosaic displays* [78], *hundred-per-cent squares* [131], and *rectangular statistical cartograms* [193]. Pertinent uses of rectangular area charts can be found in the first statistical atlas of the United States from 1874 [250], following the 1870 Census, and subsequent statistical atlases [248, 249]. The chart shown in Figure 2.1 for example, visualizes Census data in a 4-level hierarchy; (1) all citizens accounted for, (2) occupied or not, (3) occupied within one of five occupations, and (4) male or female. It provides easy access to study specific aspects of the 1870 Census data spread over 942 printed pages and its 105 full- and multi-page tables.

So far, the containment of shapes was more of a side-effect not intended to emphasize the data's hierarchy itself. In the 1970s, deliberate uses of containment have become more prominent, e.g., for nested diagrams such as *contour models* [126, 127] and *Nassi-Shneiderman Diagrams* [177] to illustrate block-structured algorithms and programs. Similar to the hierarchical structure inherent in or associated with data, algorithms and programs are (with few exceptions) also inherently hierarchical. Eventually, this led to the deliberate combination of mapping data to area and hierarchy to a space-filling, rectangular layout, studied and aptly named *treemaps* by Johnson and Shneiderman in 1990 (published in 1991 [125] and 1992 [216]).

Since then, many diverse uses in various domains and therewith variations, refinements, and extensions have been published. The thorough visual bibliography of tree visualizations, *treevis.net*, by Hans-Jörg Schulz [209] captures most of these; about a third of the 333 unique visualization techniques (at the time of writing) is directly based on treemaps. This highlights that treemaps provide an expressive and versatile basis for advanced information visualization. This is easily substantiated further by the many applications, e.g., in *biology and medicine* for gene ontologies [10], expressions [23], and microarrays [266], brain tumor data [243], biodiversity [114], RNA-sequence expressions [158], and diseases [261, 277]; in *demographics* for election analysis [83, 109], regional statistics [123], traffic analysis [219], drug adverse effects [92], and infant deaths [221]; in *businesses* for sales time series [101], motary office cases [258], property transactions [218], and callcenter phone calls [196]; in *finance* for stock markets [128, 217, 267], funding [57, 152], patent classification [143], and income [285]; in *multimedia and social media* for photos [27, 91], search results [53], and sports [46, 244, 246]; and in *other areas* for distributed processes [280], CPU utilization [102], rainfall data [137], 5G burst mapping [253], and coffee consumption [73].

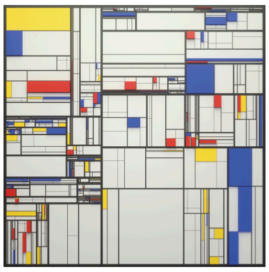


Figure 2.2: A rendering from the project "Algorithm 01" created using a treemap with a random layout and a Piet Mondrian like color palette, by Dimitris Ladopoulos, 2017.

Similar to the fundamental impact of Ken Perlin's Noise algorithm [188] in computer graphics, *computer-aided design* (CAD), and *digital content creation* (DCC), space partitioning algorithms based on treemaps have become increasingly prevalent in these domains. Especially in architecture (e.g., floor planning [163, 170, 282]), generative design (e.g., chip layout [45, 228]), and generative art. Figure 2.2 shows an example of artistic use; one of the aesthetically well-designed, procedurally generated renderings of Dimitris Ladopoulos.[2.1]

The primary use of treemaps, however, has been consistently in the visual display of file systems [72, 125, 216, 264] as well as SWSE data [16, 77, 186, 223, 225, 234, 270, 273]. *Visual software analytics*—combining findings and methods from analytics, visualization, software engineering, and data engineering to extract, visualize, explore, and analyze SWSE data—is the primary domain to which this thesis is motivated and dedicated. Treemaps can give gestalt to otherwise abstract data, thus allowing the synthesis of visual communication artifacts and supporting visual software analytics.

The remainder of this chapter introduces terminology and related work relevant to the following chapters; this comprises characteristics of tree-

[2.1]Dimitris Ladopoulos. *Algorithm 01: Exploration of an algorithm to produce treemap style diagrams.* behance.net/gallery/59061121/Algorithm-01. Nov. 2017.

structured, temporal, and multivariate data, the *treemap visualization pipeline*, the *visualization process*, and, finally, *software visualization*.

2.1 Data Characteristics

The terminology used to denote data characteristics, particularly its hierarchical structure, time reference, and variateness, is fundamental to this thesis. Three brief remarks beforehand, however: First, the term *data* will be used as a collective noun and treated with singular sense [164]. It denotes the body of multiple discrete observations comprised of individual data points, referred to as *data values*. Second, a distinction between input, raw, primary, and secondary data is not necessary in terms of content. Any *input data that is immediate subject to visualization or analytics* will be preferably denoted as *source data*. Some references, including original publications of the authors themselves, might refer to such data as *input data* or *raw data* equivalently. Third, we will sporadically mention the *abstractness* of data and thereby refer to the comprehensibility of its values. More abstract data is more complicated to interpret, less tangible, and less imaginable. Visualization typically aims at giving abstract data a gestalt to make it less abstract and more comprehensible.

How we perceive and think about bits stored in memory depends on how they were obtained, how they are accessed, what they encode, and what we want to do with them. To avoid mixing terminologies of different domains and contexts, we adhere to the rather domain-agnostic terminology used by Tominski and Schumann [238] for interactive visual data analysis:

Data Value A *data value* denotes an atomic or individual datum.

Data Domain The *data domain* denotes "the set of values that can potentially appear in the data." [238] The scale of a domain is either *qualitative* (or categorical) or *quantitative* (or numerical) and implies what operations are permitted (Table 2.1). Qualitative data is either *nominal* or *ordinal*, and quantitative data is *discrete* or *continuous*.

Operation	Nominal	Ordinal	Discrete	Continuous
Equality	✓			
Order	✓	✓		
Distance	✓	✓	✓	
Interpolation	✓	✓	✓	✓

Table 2.1: "Operations possible in different data domains" adapted from Tominski and Schumann 2020 (Table 2.2, p 19) [238].

Data Table A *data table* consists of columns representing *data variables* and rows representing *data tuples* ("depending on the context of use, [. . .] also called observation, record, item, or object" [238]).

Data Element A *data element* denotes a data tuple (a row in the data table).

Dimensions and Attributes "*Independent variables* correspond to the *dimensions* of the [data] space [. . .] the *dependent variables* describe the attributes" [238] of the data. A synonym for dimension is *key attribute*, synonyms for attribute are *value attribute* and *measure* [176].

2.1.1 Tree-Structured Data

For our purpose of visualization, we are not inherently interested in how data is stored, organized, and manipulated (database model) and assume a table-based, flat model. We are interested, however, in data elements that are related to one another in a hierarchical, tree-like fashion, modeled

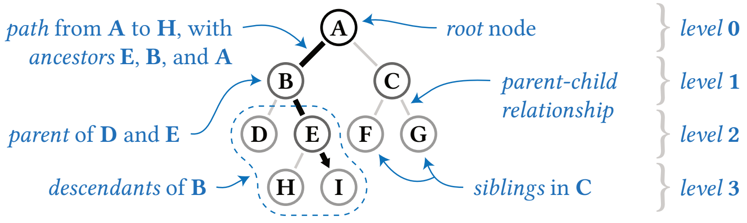


Figure 2.3: An annotated (blue), 4-level tree structure of nine nodes. **A**, **B**, **C**, and **E** are *parent*/*inner nodes*, all but **A** are *child nodes*, and **D**, **F**, **G**, **H**, and **I** are *leaf nodes*.

using *nodes* and *links*. A tree is a hierarchy that does not have cycles, i.e., "each child node has only one parent node pointing to it." [176] Figure 2.3 illustrates a small tree, annotated with the terminology of importance for subsequent chapters. For a comprehensive list of definitions of all related termini, we refer to Scheibel et al. [L19]. Even though most data is structured somehow, it is not always inherently hierarchical. As with the depiction of the census data in Figure 2.1, a tree structure can be superimposed on the data. This data view serves the purpose of processing, analysis, and visualization. SWSE data, however, is usually inherently tree-structured in multiple ways and, similar to time, on different scales. The most apparent structure comes from attributes depending on source code files that are part of a hierarchical file system. Additionally, the file structure of a software project itself tends to be aligned with the overall design and architecture of the software. Complementary hierarchical views of a project can be derived from the structure of engineering teams, integration aspects, testing configurations, budgets and goals, and more.

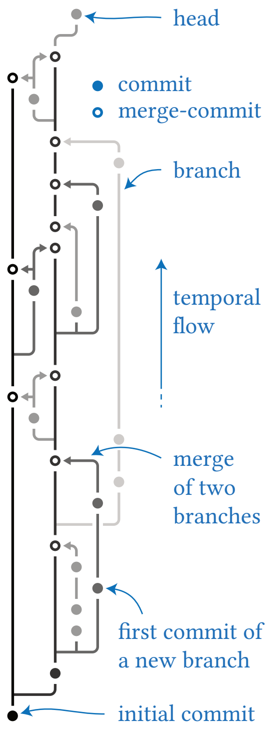


Figure 2.4: The development of a software project illustrated in revisions as seen by git. The figure is based on an example of git-graph by Martin Lange (github.com/mlange-42/git-graph).

2.1.2 Temporal Data

Whenever one of the dimensions (not attributes) of the data space is either time or a reference to time (*temporal reference*), we characterize this data as *temporal*. A value of a time dimension would represent a discrete point in time encoded, e.g., in UTC or any other time frame. In software engineering, a software project is typically represented by a file system and developed over time. Development is primarily done in chunks and scattered over *branches*, e.g., separate code lines for in-progress work and well-tested, stable code. Each chunk, regardless of its branch, contains a mostly manageable set of changes defined by the developers and committed to the project at a particular time.

In the case of git—a distributed version control system for managing source code—each *commit* is given an identity in the form of a 160-bit (SHA-1) or 256-bit (SHA-256) hash.[2.2] This hash provides a distinct reference to the project *revision* at the time of the commit (Figure 2.4). We assume SWSE data to be temporal. Note, however, that temporal references other than commits are also of interest; individual developers intuitively perceive revisions on a much smaller basis during development, such as working on a piece of code for several hours or days, evaluating different implementations locally, and using undo/redo workflows. For the software

[2.2] A free and open source version control system available at git-scm.com. In 2017, it was suggested to migrate from SHA-1 to a stronger hash function. In late 2018 the project picked SHA-256 as its successor hash (git-scm.com/docs/hash-function-transition).

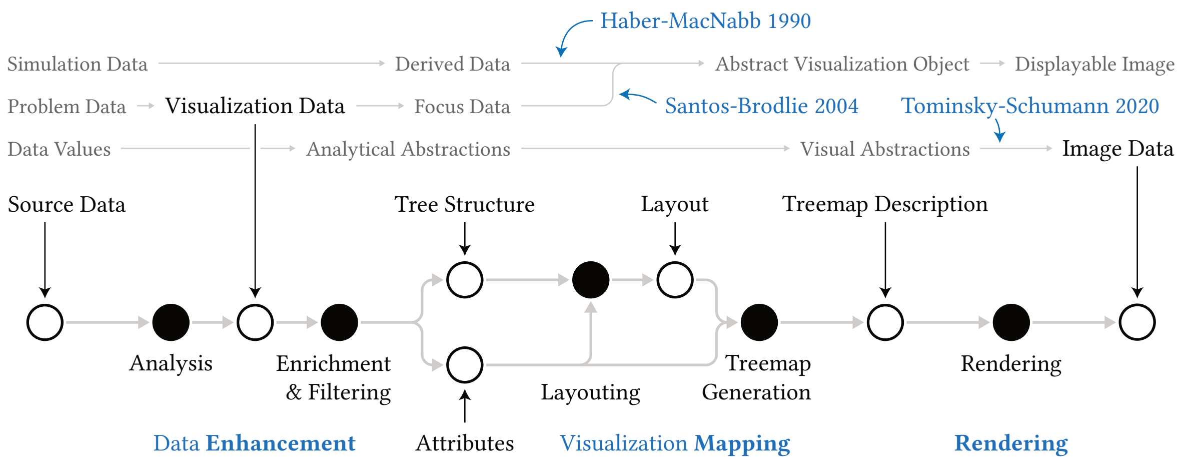


Figure 2.5: A diagram of a visualization pipeline adapted for treemap synthesis. The data-flow shows how data (outlined circles) are processed through steps (filled circles) in three phases (dashed blue). Our preferred terminology is emphasized (black or bold), and the terminology of three well-known visualization models is referenced accordingly.

development life cycle, the focus shifts, in contrast, to collections of commits in the form of minor and major revisions or weekly, quarterly, or yearly accumulated changes.

2.1.3 Multivariate Data

"General multidimensional, multivariate data typically span a frame of reference across multiple abstract dimensions. Along these dimensions multiple quantitative or qualitative variables are measured." [237] Software system and software engineering data is commonly characterized as multidimensional and multivariate. The former simply means that every data element is subject to at least two dimensions. If we now consider one, two, or more attributes of the data, it is denoted as *univariate*, *bivariate*, or *multivariate* respectively. Every static measurement of a file containing source code (*source file*), for example, can be referenced by two dimensions, the file path (a unique identifier or location) and the revision (a temporal reference). Measurements such as file size, line of code added, modified, or removed in that revision, or the number of includes, are all dependent on both dimensions and, thus, attributes within the data.

2.2 Treemap Visualization Pipeline

The term visualization is defined as (1) "the formation of mental visual images" and as (2) "the act or process of [...] putting into visible form." [2.3] In our more technical context, it denotes the process of transforming data into a displayable representation, i.e., an *image*. Today's understanding of this process was described first in 1990 [94] and has remained substantially unchanged despite some adaptions in the literature [48, 202, 238]. The most comprehensive and broadly applicable definition by Helwig Hauser and Heidrun Schumann [105] is not limited to but focuses on three major

[2.3] Merriam-Webster.com Dictionary, s.v. "visualization", accessed 22. March 2023, merriam-webster.com/dictionary/visualization.

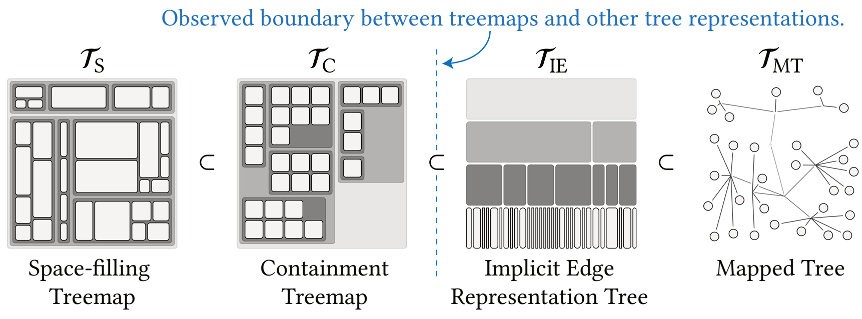


Figure 2.6: Illustration of our hyponymy of four classes of treemap visualizations adapted from Scheibel et al. [L20].

processing stages: *data enhancement*, *visualization mapping*, and *rendering*. In order to accommodate for multivariate and multidimensional visualization, data enhancement can be split into data analysis, transforming *problem data* into *visualization data*, and filtering, transforming visualization data into *focus data* [202]. Tominski and Schumann recently adapted this further. Instead of focussing on processing stages, they introduce data stages that are processed using transformation operators and stage operators [238]. Their observation is most welcomed for a unified notation as well as improved implementation design of the various techniques involved in a visualization process.

For our purposes, we have adapted the visualization pipeline, refining the steps and terminology for synthesizing treemaps as illustrated in Figure 2.5. The source data is analyzed, enriched, and filtered as part of the data enhancement stage (also referred to as *preprocessing*). The result consists of a tree structure and attributes needed for the subsequent visualization mapping. This mapping's main purpose is to compute a treemap layout and generate a technical treemap description that contains all the information necessary for image synthesis in the final rendering stage.

The following subsections describe the fundamental concepts required for our visualization mapping. These include (1) definitions and classifications of *treemaps* and *treemap layouts*, (2) introductions to the concepts of *attribute space*, *reference space*, and *visual variables*, as well as (3) our preferences for the terminology used for *3D embeddings*.

2.2.1 Taxonomy of Treemaps

Visualizations that depict parent-child relationships while adhering to the containment property [124, 125] have been historically called treemaps. Since 1991, many variations and improvements of treemaps and treemap-like visualizations have been developed. To account for this evolution of treemaps, we have introduced a refined taxonomy that allows classifying all techniques distinctly [L20]. The taxonomy defines four classes of tree-visualizations; $\mathcal{T}_S$, $\mathcal{T}_C$, $\mathcal{T}_{IE}$, and $\mathcal{T}_{MT}$. These classes are designed as hyponymy, meaning that every class has an is-a relationship to its superclass, i.e., $\mathcal{T}_S \subset \mathcal{T}_C \subset \mathcal{T}_{IE} \subset \mathcal{T}_{MT}$, and can be defined as follows:

$\mathcal{T}_S$ *Space-filling treemaps* [9, 13, 263, 266] use "the full subdivision and distribution of a parent['s] surface or volume for its children, resulting in a space-filling depiction of the leaf nodes." [L20]

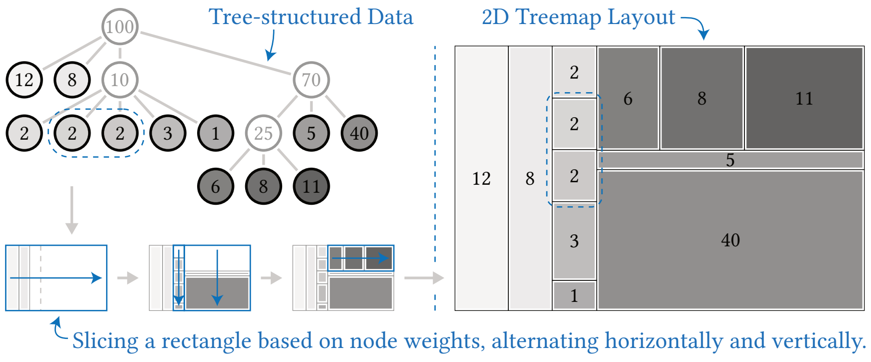

Tree-structured Data
2D Treemap Layout
100
12
8
10
70
2
2
2
3
1
25
5
40
6
8
11
5
Slicing a rectangle based on node weights, alternating horizontally and vertically.

Figure 2.8: Illustration of a treemap layouting process based on a small tree structure of weights using "slice and dice" algorithm introduced by Johnson and Shneiderman [125].

$\mathcal{T}_{\mathrm{C}}$ *Containment treemaps* [87, 117, 206, 258] do not require the area of a parent to be fully covered by its children.

$\mathcal{T}_{\mathrm{IE}}$ *Implicit edge representation trees* [90, 100, 111, 222] are not limited to containment, but use any implicit edge representations [209], i.e., containment, adjacency, and overlap.

$\mathcal{T}_{\mathrm{MT}}$ *Mapped trees* [175, 178, 226, 239] use either implicit, explicit, or vicinity relation encoding techniques in combination with a usually overlap-free spatialization technique.

Figure 2.6 shows sketches that represent the respective properties of each class. For a more comprehensive classification of related work, various examples, and a critical discussion we refer to Scheibel et al. [L20]. All techniques discussed in this paper will be presented, described, and discussed exclusively in the context of space-filling treemaps. Although in many cases, they are compatible with other treemap classes. Additionally, we assume that using paddings and margins in a treemap's layout does not result in a containment but a space-filling treemap.

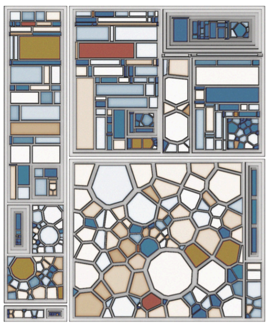


Figure 2.7: An example of a *mixed treemap* created using a *hybrid layout algorithm* by Hahn et al. which uses heuristics to pick from multiple algorithms [98]. Additional margins between inner nodes and children were created using a custom per-level post processing.

2.2.2 Treemap Layout

The layout of a rectangular treemap specifies the position and extent of every node-representing rectangle (*layout element*) and can be calculated using splitting or packing algorithms. These algorithms use (1) the tree structure combined with an iteration rule for well-defined, sequential processing of siblings and (2) an attribute as *weight*. Any domain or attribute subject to the order operation can be used to derive an iteration rule. Since most layout algorithms are applied top-down in a recursive fashion and attributes usually depend on leaf nodes, the weight-mapped attributes can be aggregated bottom-up to obtain weights for inner nodes:

The selection of a fitting algorithm depends on quality metrics, heuristics, templates, or pre-trained models and aims at improved readability, stability, adjacency, and arrangement. For example, the *average aspect ratio*, *average distance change*, or *relative direction change*, are frequently used quality metrics [97]. Algorithms can even be switched on an inner-node basis, resulting in mixed layouts as shown in Figure 2.7 [98, 149]. In any case, *layouting* is always a trade-off between readability and stability to changes in the source data or tree structure [20, 40, 230, 244, 255]. Since "no

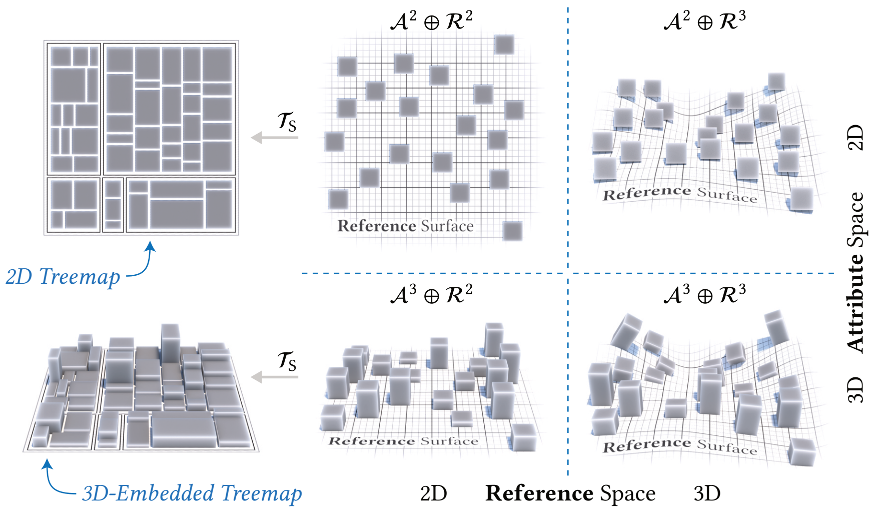


Figure 2.9: Illustrations of exemplary visualizations (depicting random data) based on the dimensionalities of attribute space and reference space, adapted from Dübel et al. [62] and Tominski et al. [238]. Moreover, treemaps (also depicting random data) in $\mathcal{A}^2 \oplus \mathcal{R}^2$ and $\mathcal{A}^3 \oplus \mathcal{R}^2$ are shown.

algorithm is superior in all [...] aspects and under all circumstances" [28], we consider algorithm selection and configuration as part of the task-specific treemap configuration [255, 256]. For a comprehensive overview and classification of existing layouting techniques we refer to Scheibel et al. [L19] and, without loss of generality, assume rectangular splitting for the treemaps and techniques discussed in this thesis (Figure 2.8).

2.2.3 Attribute and Reference Space

Most charts and diagrams use *graphical elements* placed and specialized within a two-dimensional *presentation space*. They are straightforward to create, exchange, and interact with, and we are accustomed to them from books, papers, websites, reports, and more. Yet, visualization techniques are not constrained to a 2D presentation space, and "a global distinction of 2D and 3D is [also] no longer sufficient." [62] Dübel et al. suggested distinguishing between the presentation of the *attribute space* $\mathcal{A}$ and the presentation of *reference space* $\mathcal{R}$ [62]. The dimensionality of each space is independent of the dimensionality and variateness of the source data and is instead subject to the design of the visualization.

The attribute space focuses on the dimensionality of the assembly of graphical elements. Each graphical element is created based on one or more visual variables depicted within these dimensions. The reference space focuses on the spatial location of graphical elements. For geo-spatial data, these are usually the locations of observations. In our case, visualizing tree-structured abstract data, positions are computed using a treemap layout algorithm. To uniquely identify attribute-reference-space combinations, the notation $\mathcal{A}^i \oplus \mathcal{R}^j = \{(a, r) \,|\, a \in \mathcal{A}^i, r \in \mathcal{R}^j\}$, with $i, j \in \{2, 3\}$ is used. Examples for all four combinations are shown in Figure 2.9.

2.2.4 Visual Variables

Visual variables describe "the graphic dimensions across which a [...] visualization can be varied to encode information." [200] In other words, they describe aspects of *graphical elements* that can be used to distinctively depict data values. Such graphical aspects were established in *thematic mapping*, beginning in the late 18th and increasingly in the 19th century [64, 160, 197]. Today's understanding of visual variables is commonly attributed to Jacques Bertin [24], who introduced the term *variables rétiniennes* (retinal variables). He focussed on variations in *forme*, *orientation*, *couleur*, *grain*, *valeur*, and *taille*, from french, translated to shape, orientation, color, texture, color value, and size respectively [25, 26]. Since then, visual variables have been steadily reiterated, evolved, and evaluated [37, 81, 245] and, in the context of $\mathcal{A}^2 \oplus \mathcal{R}^2$ visualizations, include, but are not limited to, *location*, *size*, *shape*, *orientation*, *color hue*, *color saturation*, *color value*, *texture*, *arrangement*, *crispness*, *resolution*, and *transparency* [200]. Each variable is more or less applicable to encode specific data types and differs in effectiveness on different tasks [161, 200].

Concerning the treemap visualization pipeline, the visualization mapping implements the mapping of attributes to visual variables and results in descriptions of graphical elements as part of the treemap description. The subsequent image synthesis requires a renderer capable of creating an accurate, description-conforming visual output. Consequently, the renderer is the primary limiting factor for the design and use of visual variables and is of significant concern to our research (implicitly reflected by our thesis statements $\mathcal{H}_E$ and $\mathcal{H}_R$). In analogy to the famous statement "[d]ie Grenzen meiner Sprache bedeuten die Grenzen meiner Welt" by Ludwig Wittgenstein [276], we have learned that the limitations of our renderers mean the limitations of our visualizations.

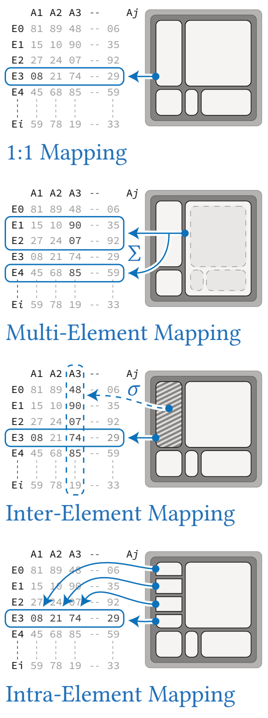


Figure 2.10: Illustrations of variations in relatedness between data elements (left) and graphical elements (right).

2.2.5 Variateness and Expressiveness of Visualizations

Similar to data, visualizations can be characterized by their variateness defined by the number of attributes depicted. A treemap mapping three distinct attributes to three distinct visual variables is a *multivariate visualization*. However, a single attribute mapped to multiple visual variables would result in a univariate visualization—a simple method to increase visual accuracy or emphasizes data characteristics for example. The term multivariate visualization, therefore, certainly implies but does not explicitly refer to the source data's variateness. Similar to this observation, we introduce degrees of *relatedness* that advance the classical 1:1 mapping between data elements and graphical elements of treemaps (Figure 2.10):

Multi-element Mapping A single graphical element can represent multiple data elements and encode, e.g., accumulated, topological, or otherwise associated information.

Inter-element Mapping A graphical element representing a single data element is not limited to exclusively depict data from that element. Additional visual variables can be used to superimpose any information derived from topologically or otherwise associated data elements.

Intra-element Mapping Multiple graphical elements can refer to the same data item but differ in the mapping of at least one visual variable.

With these notions of variateness and relatedness in mind, we define the *expressiveness* of a treemap as the measure of relations and visual variables available for the meaningful, simultaneous, unambiguous display of data. As noticed by Jacques Bertin, "[i]t is the designer's duty to make the most of this variation, [...] to flirt with ambiguity without succumbing to it." [26] To this end, we will individually discuss the supposed use and application of our techniques and describe measures that help preclude "chartjunk" [245] and facilitate treemap designs that mitigate visual clutter.

2.2.6 Designations of Treemaps in $\mathcal{A}^3 \oplus \mathcal{R}^2$

Treemaps in $\mathcal{A}^2 \oplus \mathcal{R}^2$ are called *2D treemaps* [124]. Every graphical element therein is located within two spatial reference dimensions, e.g., a rectangular coordinate system with x-axis and y-axis. All visual variables are limited to these two dimensions. Treemaps in $\mathcal{A}^3 \oplus \mathcal{R}^3$ are called *3D treemaps* [124, 210, 211] or *treecubes* [182, 194, 232]. They can make full use of three dimensions, both for positioning and for the design of the graphic elements. A treemap in $\mathcal{A}^2 \oplus \mathcal{R}^3$ would be similar to 3D-geovirtual environments using billboards [63, 214]—2D graphical elements that always face the virtual camera—such as glyphs or small charts for encoding data. To our knowledge, there are currently no treemap techniques explicitly designed for $\mathcal{A}^2 \oplus \mathcal{R}^3$.

The treemaps in $\mathcal{A}^3 \oplus \mathcal{R}^2$—the main subject of our research, especially due to statement $\mathcal{H}_E$—have received great attention and can be found in various variations and specializations. City-metaphors [75, 133, 271, 273], *Information Pyramides* [6], and *Step Trees* [30] may be mentioned here exemplarily among many others [33, 141, 262, 271, 280]. This attention is primarily due to their advantages over $\mathcal{A}^2 \oplus \mathcal{R}^2$ and $\mathcal{A}^3 \oplus \mathcal{R}^3$, namely, (1) the increased expressiveness for graphical elements, first and foremost, the introduction of height as an additional primary visual variable (Figure 2.11), (2) no occlusion of $\mathcal{A}$ by $\mathcal{R}$, and (3) no self-occlusion of $\mathcal{R}$ [62]. Despite this, no definite designation has been established in the literature. The notion *3D*, although used occasionally [30, 46, 84, 150, L1, 241], is inconvenient and misleading. It overlaps with the notion of "true 3-D treemaps [which] would be volumes partitioned on all 3 dimensions" [247] and suggest that the third dimension can be used to its full extent, which it cannot.

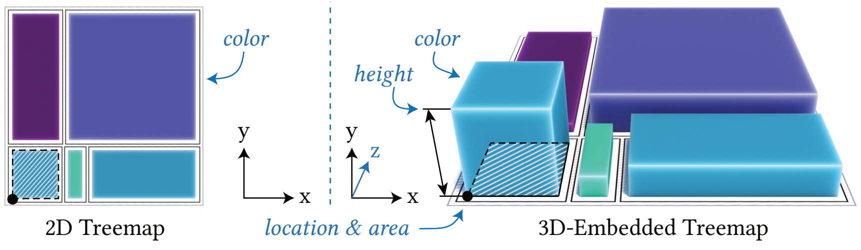


Figure 2.11: The visual variables location, area, and color used in a 2D and a 3D-embedded treemap. The three-dimensional attribute space allows for additional visual variables such as height.

In 1967 Bertin deliberately restricted the perspective and use of the third dimension, calling them stereograms (*stéréogrammes*) [24]. In 1992 David Turo and Brian S. Johnson used *2 1/2-D* [247], and, one year later, Johnson switched to $2^{+}D$ in his dissertation [124]. The first term emphasizes that

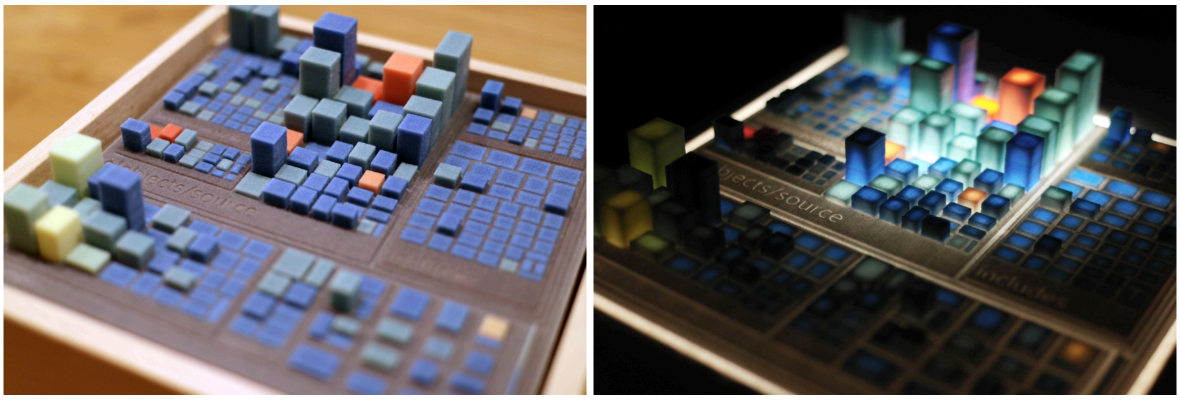


Figure 2.12: A treemap, 3D-printed in 2015, depicting the small software project *globjects* [R15]. The prototypical map was embedded in a wooden frame and the hollowed cuboids could be dynamically lit with LEDs.

only one side of the xz-hyperplane is used, the second that more than 2D but not quite all of 3D is used. Based on *2½D* used by Bohnet and Döllner [33], we originally introduced [L2] and used *2.5D* in most of our previous publications [L10, L4, L21, L5, L9, L18, L15, L13] simply because it is easier to read and write. In retrospect, however, all these variants are misleading and seem somewhat arbitrary: It is, for example, difficult to precisely define what a 50%-use of a 3D space means. To this end, we suggest establishing the more precise and straightforward term *3D-embedded* [L22, L23]. It denotes the embedding of graphical elements in a virtual 3D space, positioned using 2D locations, i.e., based on 2D treemap layouting, and assembled using 3D-capable visual variables. The most obvious visual variables of 3D-embedded maps, and 3D-embedded treemaps in particular, are location, area, color, and height (Figure 2.11).

2.2.7 **Alternatives to 2D Image Synthesis**

It should be noted that the final stage in the visualization pipeline does not have to be 2D image synthesis. The treemap description itself can be encoded as a 3D-file [L5, L16], e.g., for the purpose of preservation, exchange, or further processing. Also, the rendering stage can be (1) enhanced to support stereoscopic rendering for exploration in extended reality [76, 173], or (2) replaced by fabrication, e.g., 3D-printing [75, 203]. Out of curiosity, we 3D-printed two treemaps in the course of this work [L16]. We investigated the requirements, constraints, and problems associated with the process of physical fabrication. Our motivation was to allow software developers to capture versions of a software project as a unique and outstanding keepsake (Figure 2.12). Such considerations, however, have not influenced the concepts and techniques presented in this thesis. Instead, the concepts and techniques and our respective implementations made it very easy to explore and prototypically test such ideas.

2.3 **Visualization Process**

Most visualizations we deal with in our everyday lives are static in nature. They are created once, then provisioned and presented using print media, digital media, or as part of interactive computing notebooks, and then viewed. The data and information they convey are well-defined and well-tuned and serve to confirm or communicate knowledge we already expect or know. Such static visualizations do not allow direct interaction with graphical elements or the overall visual display. That is, they do not support generic tasks such as filtering, selection, zoom, or comparison through explicit manipulation of the visualization. However, they have qualities that facilitate interactions in a perceptual, explorative nature. These include "avoid distorting what the data have to say[, ...] encourage the eye to compare different pieces of data[,] reveal the data at several levels of detail, from a broad overview to the fine structure[, and] serve a reasonably clear purpose [...]." [245] These qualities enable interactions in that they convey multiple layers of detail and abstractions, each serving different tasks within a dynamic loop of perception and exploration. The general understanding of what interaction refers to has changed, though. What used to be printed infographics have become interactive, volatile charts or dashboards depicting real-time data such as oil prices, Coronavirus cases, supplies of natural gas, or CO_2 emissions.[2.4] Interactive graphical displays are ubiquitous and *interactions* are generally understood as explicit actions that directly modify the visualization.

In academia and professional industries, expressive visualizations are an integral, interactive visual interface to data, perceived and changed in increments to generate knowledge. A visualization is "expressive if it allows us to carry out the actions needed to acquire the desired information." [238] Actions denote tasks such as *exploring*, *describing*, *explaining*, *confirming*, and *presenting* [238]. Our ability to understand and explore the interplay of tasks and visualizations, e.g., to identify and recommend compelling visualizations for well-defined visual analytics tasks, rests on the technical capabilities of the visualizations available. An unresponsive, cluttered, complicated, visually unsightly data display is predestined to underperform and cause (technical) bias in a comparative study. Fundamentally complex visualizations are more complex to implement and, thus, particularly prone to this problem. This impacts visualization research in general and limits recommender systems [43], visualization design strategies [176], as well as perception or task-based evaluations [L17]. The awareness of these limitations and the challenges in visual analytics, e.g., expressed by Keim et al. [132], motivated us to improve 3D-embedded treemaps regarding (1) *scalability*, supporting little as well as massive data, (2) *interactiveness*, integrating 3D-map navigation metaphors and common interactions such as filtering, selecting, or highlighting, (3) *availability*, providing users on various platforms and devices access to 3D-embedded treemaps, (4) *visual fidelity*, incorporating high-quality rendering, and (5) *expressiveness*, facilitating multivariate depictions of data.

To approach these improvements in a structured way, we adhere to the *interactive visualization process* as consolidated by Jarke J. van Wijk [274] and the visual information seeking mantra introduced by Ben Shneiderman [215]—two complementary, unrestrictive, generalized frameworks in

[2.4]ZEIT ONLINE, s.v. "Energiemonitor", "Corona-Zahlen", accessed 22. March 2023, zeit.de

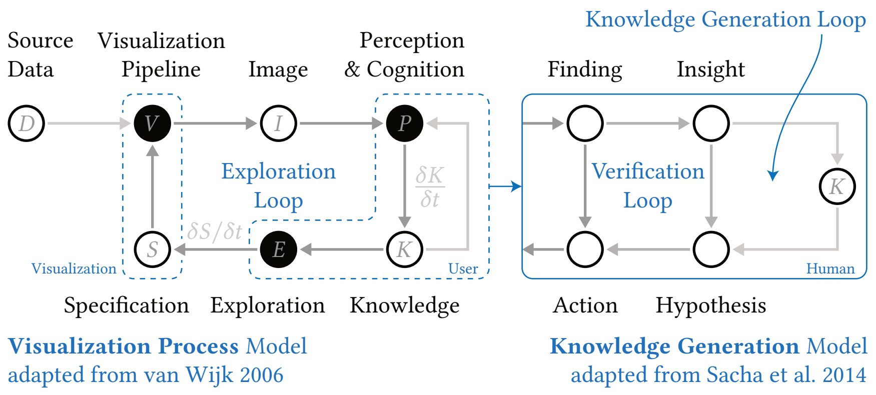


Figure 2.13: Illustration of the visualization process adapted from Wijk [274] (with image tweak [238]) and the knowledge generation model adapted from Sacha et al. [201] (empty circles represent artifacts, filled circles represent functions). The latter is interpreted as refined path from *P* to *E*.

visualization design. Wijk's visualization process and derived knowledge generation models model the interplay between user and visualization. The visualization pipeline (Figure 2.5) is embedded in an iterative, interactive loop of visualization, perception, exploration, and refinement (Figure 2.13). The visualization is configured using a specification that covers every aspect essential to a visualization, from data selection, mapping, and other representation configurations, such as navigational aspects, resulting in an updated image-based output. The user then perceives this image, explores it, and, eventually, gains knowledge, refines the specification, and thereby iterates the interactive loop. Knowledge generation models, most notably the one introduced by Sacha et al. [201], refine this process and facilitate consolidation, evaluation, and taxonomies in research but have not explicitly been utilized in this work.

Complementary to the visualization process, we found the *visual information seeking mantra* (VISM) to be of great support to our goals; "overview first, zoom and filter, then details-on-demand." [215] VISM introduces a sequence of generalized interactions instead of specific tasks, which eases the identification of technical requirements for a visualization's implementation. It provides a foundation for what a visualization should feature, facilitating (1) the mapping of fitting tasks and (2) the incremental refinement of visual variable configuration. It also provides guidance on how knowledge discovery is supported by choosing different interactions and in what ways the specification is modified and refined. More specific tasks such as *correlate*, *associate*, *distinguish*, *identify distribution*, *compare*, *access information*, and *identify value* are all supported by treemaps. However, in order to keep the implementation of our treemap improvements viable, we consider these as specializations of VISM.

The two concepts emphasize and remind us that the images of our visualization are most likely of temporary use and, more importantly, that the user's knowledge is not static but dynamically evolving. The user and the visualization interact mutually in a continuous loop of specification, visualization, perception, knowledge exploration, and refinement. For more refined and structured visualization design processes, refer to Tamara Munzner [176] as well as Christian Tominski and Heidrun Schumann [238].

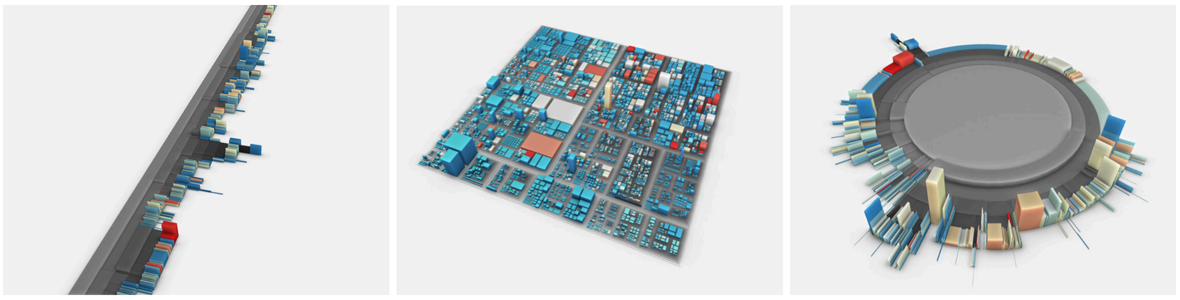


Figure 2.14: Examples of interactive, 3D-embedded maps of tree-structure data using layouts of icicle plots (left), treemaps (middle), and sunburst charts (right) created using *arboretum* [R14].

2.4 Software Cartography using Treemaps

“Unfortunately, it is impossible to represent software in its original form because it does not have any.” [147] The use of well-known cartographic techniques to create spatial representations for visualizing and analyzing SWSE data was seeded with the revival of treemaps by Johnson and Shneiderman [125]. Today this approach is commonly referred to as *software cartography*. Manifold mappings and applications of 3D-embedded treemaps have been accompanied by the introduction of various similar terms and metaphors. For example, when Knight et al. [136] discussed a *Software World* metaphor, they applied *city* and *district* metaphors directly to software visualization. This metaphor was later used by Langelier et al. [147] and applied in variations such as *Code City* and *Software City* by Wettel et al. [269, 272] and Steinbrückner et al. [226], respectively. Similarly, Viana et al. [257] and Balogh et al. [12] used the terms *JSCity* and *Code Metropolis*, respectively. Eventually, Kuhn et al. [142] introduced the term *thematic software maps*, though not in the context of classic treemaps.

2.4.1 Specialization of the Term Software Map

The term *software map* is commonly used within our research group [33, L1, 205, 242, L2] and used throughout this thesis as well. We prefer just ‘software map’ since it is neither desirable nor intelligible to account for all aspects of a software system within a single map and somewhat obscure what a *non-thematic* software map would be. Furthermore, the term does not rely on metaphors that may induce false expectations about the visual appearance of the map. Referring to a city, for example, could indicate buildings with detailed facades surrounded by streets and walkways covered by vehicles, pedestrians, urban furniture, and vegetation. Instead, it claims to make abstract SWSE data visible and locatable through map-like depictions, and suggests to facilitate exploration and communication. Moreover, omitting ‘thematic’ makes it straightforward to differentiate maps of different technical or visual characteristics, such as *interactive*, *stable*, *animated*, or *aggregated*.

Although, the techniques showcased in the following chapters are fit to be applied to a variety of similar visualizations (Figure 2.14), the scope of this work is set by the following specialization of a software map:

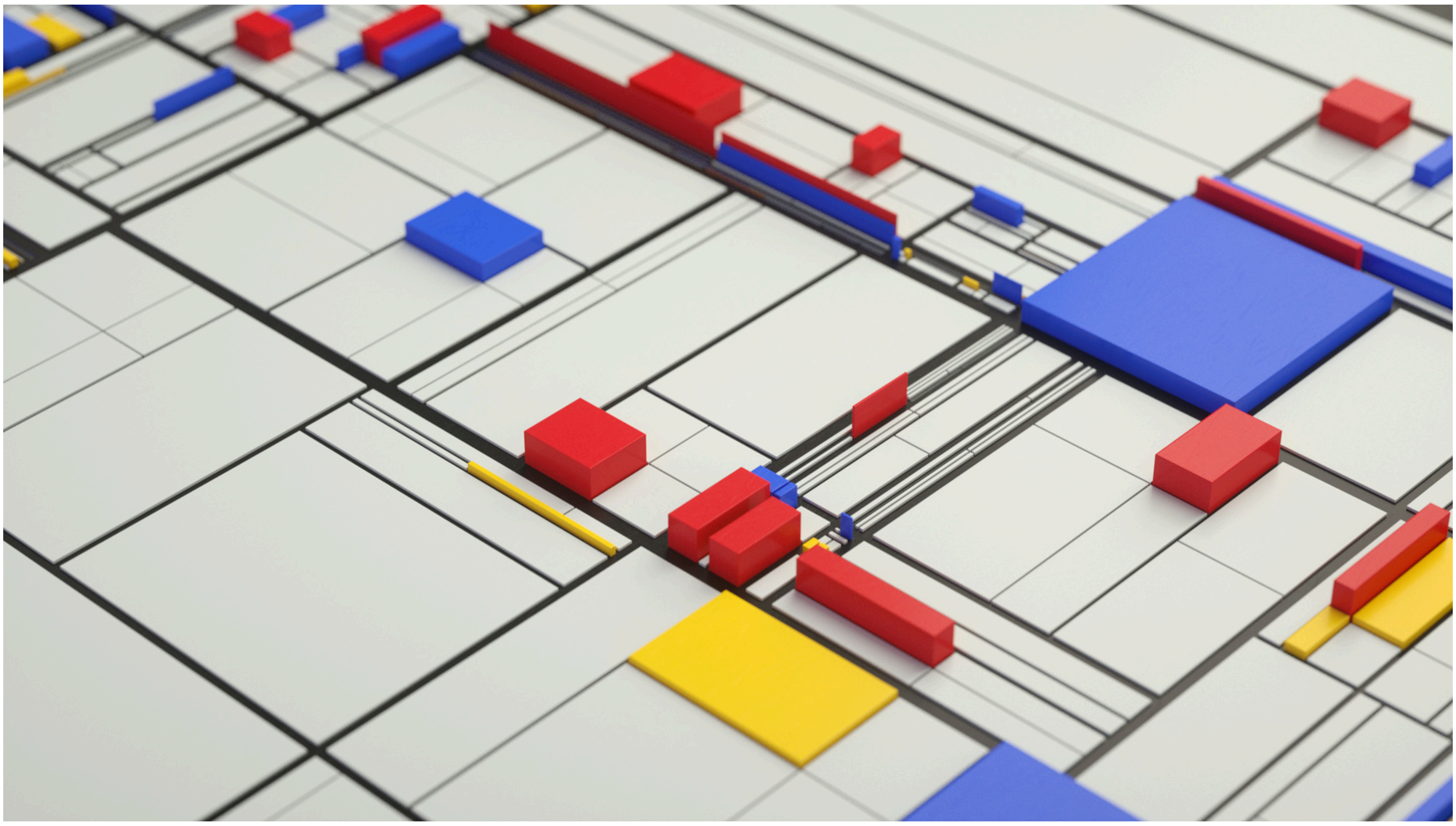


Figure 2.15: An offline rendering from the project "Algortihm 01" showing a 3D-embedded treemap using manually randomized data mapped to weights, heights, and a Piet Mondrian like color palette, by Dimitris Ladopoulos, 2017.

3D-embedded Software Map A *3D-embedded software map* (software map) is a containment treemap ($\mathcal{T}_\mathrm{C}$) in a two-dimensional reference space ($\mathcal{R}^2$) encoding software system and software engineering data using visual variables of a three-dimensional attribute space ($\mathcal{A}^3$).

2.4.2 General Delimitation of Research Contributions

Transitioning software maps from 2D to 3D-embedded treemaps may appear to be a minor change, but it has significant implications for their design, implementation, provisioning, and use for cartographing abstract data. As such, this work comprehensively explores this transition's challenges and opportunities. While techniques and ideas from cartography are relevant for navigation, interaction, and labeling, this work focuses on developing 3D-embedded treemaps and evaluating fitness to software analysis. Although references to works in cartography will be cited, they are not within the scope of this work.

Figure 2.15 shows a beautiful example of a 3D-embedded software map. It conveys the basic ideas and best-case aesthetic qualities while incorporating the traditional shortcomings of software maps. Every graphical element (cuboid) might depict a source code file of a hierarchical software system. Each cuboid's area might encode the *line of code* (LOC), its height, the number of changes in the last quarter, and its color, the number of different developers that have worked on that source file: blue indicating more than two, yellow precisely two, and red one or none in the case of generated code. The map could be used to spot developer activity and explore knowledge monopolies within the source code. However, for this specific map, the artist has used and tweaked random values for aesthetic purposes. Despite this, the map captures the conceptual baseline, published by Johannes Bohnet and Jürgen Döllner [33], that this thesis extends upon in a vast body of international, peer-reviewed publications [L1] to [L23]. Additionally, novel and customized research prototypes have been created, which allow for the development of large, dynamic, and interactive software maps across different platforms (such as Windows, macOS, Ubuntu, and other Linux-based distributions) and devices (including desktop, mobile, and web). Most of the findings presented in this thesis are based on these prototypes, which are part of a well-received collection of libraries and services [R1] to [R15].

3 Visual Variables for 3D-Embedded Treemaps

The contents of this chapter are based on the following original publications:

H. Würfel, M. Trapp, **D. Limberger**, and J. Döllner. "Natural Phenomena as Metaphors for Visualization of Trend Data in Interactive Software Maps". In: *Proc. EG CGVC*. 2015 [L2]

D. Limberger, C. Fiedler, S. Hahn, M. Trapp, and J. Döllner. "Evaluation of Sketchiness as a Visual Variable for 2.5 D Treemaps". In: *Proc. IEEE IV*. 2016 [L4]

D. Limberger, M. Trapp, and J. Döllner. "Interactive, Height-Based Filtering in 2.5D Treemaps". In: *Proc. ACM VINCI*. 2018 [L13]

D. Limberger, M. Trapp, and J. Döllner. "In-Situ Comparison for 2.5D Treemaps". In: *Proc. SciTePress IVAPP*. 2019 [L15]

D. Limberger, W. Scheibel, J. Dieken, and J. Döllner. "Visualization of Data Changes in 2.5D Treemaps using Procedural Textures and Animated Transitions". In: *Proc. ACM VINCI*. 2021 [L21]

D. Limberger, W. Scheibel, J. Döllner, and M. Trapp. "Visual Variables and Configuration of Software Maps". In: *Journal of Visualization* (2022) [L23]

D. Limberger, W. Scheibel, J. van Dieken, and J. Döllner. "Procedural Texture Patterns for Encoding Changes in Color in 2.5D Treemap Visualizations". In: *Journal of Visualization* (2022) [L22]

The mapping stage of the visualization pipeline transforms pre-processed and filtered data into depictable and reversibly encoded graphical primitives and scenes. This stage is essential for encoding data to visual variables, considering human capacities and abilities to interpret a depiction [265]. "[V]isual variables describe the graphic dimensions across which a map or other visualization can be varied to encode information." [200] Embedding 2D treemaps in a spatially three-dimensional attribute space provides an additional dimension for designing visual variables (Figure 3.2). We adapt, expand upon existing, and introduce new, specialized visual variables to enrich the visual vocabulary. "Increasing [this vocabulary] can provide for richer information resolution" [247] and allow for additional expressiveness over traditional 2D treemaps, i.e., supporting $\mathcal{H}_E$.

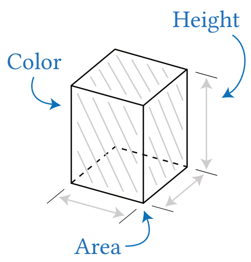


Figure 3.1: Fundamental visual variables available to a cuboid of a 3D-embedded treemap.

Several factors often limit the effectiveness and expressiveness of treemaps. First, using multiple visual variables such as area, color, and height (Figure 3.1) may exceed a user's capacity to process the displayed information

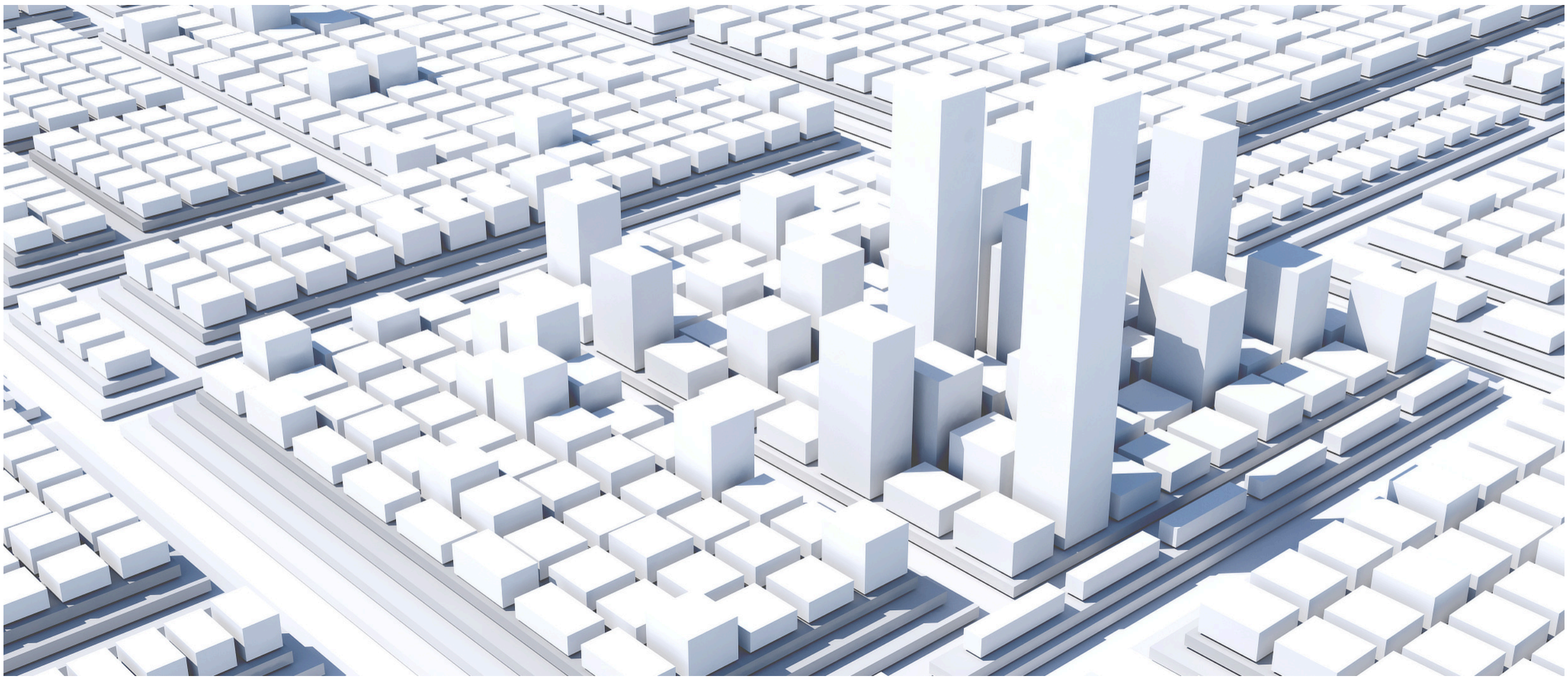


Figure 3.2: A rendering of a 'blank' treemap that only uses height and area for attribute mapping. It illustrates that there is sufficient space for the use and addition of other visual variables to serve a specific use case.

when confronted with complex and massive data. Hence, it is critical to carefully consider the selection and implementation of visual variables to prevent overwhelming users and impeding comprehension. Second, complex visual primitives can detract from a treemap's effectiveness (Figure 3.2). In line with the *visual information seeking mantra* (VISM), we aim to selectively depict additional attributes of multivariate data on demand, focusing on nodes of interest while maintaining the overall context. Third, the effectiveness of visual variables depends on the quality of their implementation, particularly in terms of rendering. For instance, poorly implemented transparency increases clutter and be counterproductive.

In our investigation, we focus on concepts and techniques applicable to cuboids. We outline selected visual variables and discuss value-adding adaptations, such as flattened inner nodes and height-based filtering. Primarily, we spotlight four techniques, namely, sketchy outlines and sketchy hatching [L4], physically-based materials and phenomena [L2], in-situ templates [L15], and animated transitions using procedural textures [L22].

3.1 Visual Variables of $\mathcal{A}^2$ and $\mathcal{A}^3$

The choice of a visual variable for mapping multidimensional, multivariate data is determined by the properties of the variable itself, including selective, associative, quantitative, length, and order properties. These properties significantly influence the perception and understanding of the presented information [44]. A catalog of *map themes* compiles commonly used attribute selections and mappings to visual variables relevant to specific tasks. A *map theme* defines attributes mapped to visual variables. It represents a topic or use-case-specific treemap template that assists users in visual analytics for data-driven decision-making. This involves to "(1) derive insight from massive, dynamic, ambiguous, and often conflicting data, (2) detect the expected and discover the unexpected, (3) provide timely, defensible, and understandable assessments, and (4) communicate assessment effectively for action" [132]. In the following, we outline visual variables in $\mathcal{A}^2$ and selected metaphors and techniques that have been or can be used with 3D-embedded treemaps to address these challenges.

To effectively employ visual variables of $\mathcal{A}^2$ [26, 44, 200] in a 3D-embedded treemap, we need to transfer them to corresponding visual variables in $\mathcal{A}^3$. While some transfers may be relatively intuitive, the added dimension can introduce side effects that must be explicitly addressed. For instance, extruding the area of a rectangle in 2D into volume in 3D requires the user to be aware of and able to differentiate between volume and ground area. Similarly, implementing color in a 3D environment might involve shading, shadows, or reflections, which should be accounted for when choosing a color scale. Decisions such as whether to apply color encoding to the top, lateral, or all faces of cuboids must ensure that the visual variable remains effective and comprehensible.

3.1.1 Visualization Mapping and Rendering

The *mapping* stage transforms tree structure and attributes into a treemap description, which comprises depictable and reversibly encoded graphical primitives and scenes. There is not necessarily an explicit representation of the resulting visualization object in memory. The result may only be volatile during visualization, especially from an implementation point-of-view: the distinction between mapping and rendering sustains more on a conceptual level [204, 241].

The visualization mapping provides attribute values adjusted to the visual variables, descriptions of graphic primitives, and additional data for rendering. The *rendering* stage transforms the visualization abstractions, i.e., the treemap description, into images using (real-time) image synthesis. As a result of the rendering stage, the mapped attributes are visually encoded in the output image.

The simultaneous use of multiple visual variables to encode information in a single graphic element (*superposition*) can be used to convey complex information. Table 3.1 [L23] enlists selected visual variables and their applicability to (1) inner nodes and leaf nodes, (2) 2D and 3D-embedded treemaps, and (3) the data type it most certainly can convey.

3.1.2 Visual Variables in $\mathcal{A}^3$

Some visual variables in $\mathcal{A}^3$ are limited to graphical elements with specific characteristics and not applicable otherwise. For example, arrow patterns applied to the lateral faces of cuboids may not be visible if the height of the cuboid is zero, resulting in a flat rectangle. Additionally, the choice of dependent or independent mapping plays a vital role in the effectiveness

		Node		Space		Data Type			
Visual Variable	Stage	Inner Node	Leaf Node	$\mathcal{A}^2 \oplus \mathcal{R}^2$	$\mathcal{A}^3 \oplus \mathcal{R}^2$	Nominal	Ordinal	Interval	Ratio
Area (Foot Print, Size)	M	✓	✓	✓	✓		✓	◦	✓
Color	R	✓	✓	✓	✓	✓	✓	✓	✓
Height	M	◦	✓		✓		◦	◦	✓
Transparency	R	◦	✓		✓	◦	◦	◦	✓
Light Emission (Glow)	R	◦	✓	◦	✓		◦	◦	✓
Stacking	M		✓	✓	✓	✓		◦	✓
Stacking (global layer)	M	◦	✓		✓		◦	◦	✓
Segments	M	◦	✓	✓	✓	✓	◦	◦	✓
Shape Type	M		✓	✓	✓	✓	◦		
Shape Parameter	M	◦	✓	✓	✓		◦	◦	✓
In-situ (Change, Diff)	M		✓	✓	✓		✓		
Contour Width	R	✓	✓	✓	✓		✓	◦	◦
Contour Color	R	✓	✓	✓	✓	✓	✓	✓	✓
(Contour) Stippling	R	✓	✓	✓	✓	✓	✓	◦	◦
Sketchy Contour	R	✓	✓	✓	✓	◦	✓	◦	◦
Surface Pattern (Texture)	R	◦	✓	✓	✓	✓	✓	◦	◦
Surface Noise (Texture)	R	◦	✓	✓	✓	◦	✓	◦	✓
Surface Shading (Texture)	R	✓	✓	◦	✓	◦	✓	✓	
(Surface) Hatching	R	✓	✓	✓	✓	◦	✓	◦	◦
Nesting-Level Margin	R	✓		✓	✓		✓	◦	✓
Color Weaving	R	✓	✓	✓	✓	✓	◦		
Height Threshold	R	✓	✓	◦	✓		◦	◦	✓

Table 3.1: M – Mapping | R – Rendering | ✓ – Supported | ◦ – Partially Supported.

of visual variables. Dependent mapping establishes a direct connection between dimensions and attributes, while independent mapping allows for a separate and flexible representation of dimensions and attributes. The following outlines selected visual variables, such as height, shape, and texture, each of which can uniquely contribute to a treemap's design.

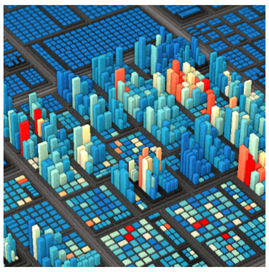


Figure 3.3: Height used as a visual variable by extruding the 2D shapes.

Height. The height of cuboids by extrusion of the rectangles [29, 50] serves as a secondary visual variable (Figure 3.3). We tend to convey information in the order of decreasing importance for the task: (1) color, (2) height, and (3) other visual variables. Height allows for an intuitive encoding of data changes by means of growing or increasing vs. shrinking or decreasing, respectively. However, height should not be used to directly depict negative or diverging scales, as this would result in downwards-facing cuboids. If a negative value range is relevant, it is mapped inversely to height. If absolute values are relevant, they could be mapped to height and the sign to color, shape, or texture. Generally speaking, a cuboid should be high when the underlying data is interesting.

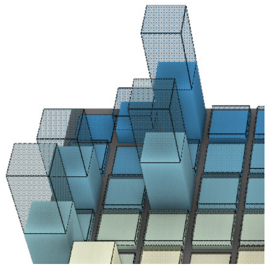


Figure 3.4: Transparency as a visual variable using dithering [L23].

Transparency. Transparency, for example, can encode an attribute measuring the relevance of a node. When an object disappears gradually, it becomes more irrelevant. Transparency can also be used to reduce occlusion and to encode different node states [157]. Last but not least, it allows to depict removed or planned components, goals, and irrelevant nodes or just to enhance the expressiveness and visual quality of texturing (Figure 3.4).

Convex Shapes. Pyramid-like shapes can be used to further encode an attribute or reduce occlusion [247]. The orientation of the leaf node's geometry [147] and the type, employing poly cylinders [162] and three-dimensional glyphs [32] are further suitable as visual variables. Convex shapes can also be constructed using, e.g., Voronoi tessellation, resulting in Voronoi treemaps [13, 98, 180]. When following the approach of space-filling curves for non-rectangular layouts, they can be used to create GosperMaps [9] and *stable and predictable Voronoi treemaps* [108] based on additively weighted power Voronoi diagrams. We do not consider to what extent our concepts and techniques can be applied to convex primitives in general and instead focus exclusively on cuboids.

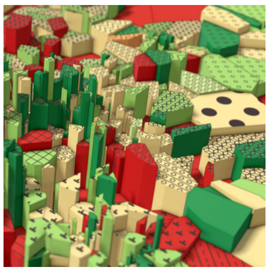


Figure 3.5: Top faces textured with patterns (nominal mapping) [L7].

Textures. Textures and patterns can be applied to encode categorical data [L7, 172] (Figure 3.5). Alternatively, texture intensity was suggested as a visual variable for scales with a natural zero [112]. Textures can be used to create a distinctive look and feel using, for example, physically-based materials. Procedural texturing can further be used to superimpose rulers or stripes, e.g., to make height effectively countable or at least increasing the accuracy of comparability between cuboids. Another approach is to use procedural textures to encode underlying data distributions [L18].

Juxtaposing and Complex Shapes. To encode multiple states or composition, data vases, stacked cuboids, or segmenting/fragmenting can be used. In remembrance of stacked bar charts, the extruded polytopes can be subdivided in height, allowing for the depiction of subcategories and their share of the overall height [84, 118]. This process can further be utilized, e.g., to encode evolution using *evolution segments* [226] and *data vases* [235].

Topology & Relations. For an emphasis on the topology with respect to the nested structure of nodes, cushion shading [275] or hierarchical stippling [221], variations of margins or padding, as well extruded, stacked inner nodes [29] can be applied. If other relations of nodes in addition to their tree-structured topology are of interest (e.g., functional dependencies or often-coinciding changes during the development process), edges, trajectories, or edge bundles can be superimposed on treemaps [42, 111, 224]. This approach, however, is visually constrained by the number of depicted relations and data set size. Superimposing relations using tubes on top of cuboids introduces additional visual complexity not inherent to the treemap metaphor. Outlines or emissive light can also be used to emphasize nodes due to user interactions such as filtering and selection or to highlight system activity [59].

For a discussion of common visual variables of $\mathcal{A}^2$, we refer to the overview given by Robert E. Roth [200]. Furthermore, we generalize color for the sake of simplicity to 'just' color but are aware of the many facets and impactful uses of color [56, 82, 104, 208]. Other visual variables, enlisted in the table but not mentioned yet, such as emissive light, color weaving, height threshold for filtering, nesting level contouring, shading, procedural texture patterns, sketchiness outlines and sketchy hatching, in-situ templates, etc., will all be discussed in the remainder of this chapter.

3.2 Sketchy Contours and Surface Hatching

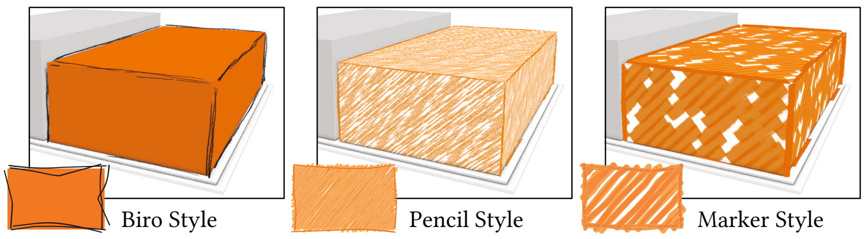


Figure 3.6: Three different illustration styles, i.e., ballpoint pen, pencil, and marker, are used to render a cuboid of a very simple 3D-embedded treemap [L4].

'To sketch out' means to create a rough or preliminary version of an idea, concept, or plan, typically in a quick and informal way. Artists, designers, and engineers commonly use sketching to visualize their ideas before creating a final product or design. Using non-photorealistic, artistic, and illustrative rendering changes how people read the depicted information [116]. It "offers different ways of communicating ideas of narrative, purpose, ownership, accuracy[,] and aesthetic" and it "may be reliably used as a visual variable on an ordinal scale, but [...] caution should be exercised when representing interval or ratio scale data." [278] It conveys visual imprecision, decreases people's confidence in the underlying data quality, and encourages constructive feedback, discussion, and participation [156, 283] as well as supports interactive exploration [39, 89].

This chapter's first spotlight is on *sketchiness* and shows how it can be used as a visual variable in $\mathcal{A}^3$. We applied sketchy rendering to the components of a treemap, imitating hand-drawn outlines and fillings. Thus, a cuboid's sketchy appearance can map uncertainty, imprecision, or vagueness, primarily captured by ordinal data with small range.

3.2.1 Sketchy Outlines for Interactive Visualization

The extent to which we can apply sketchiness must be within the ability to synthesize this style in real-time, i.e., for interactive, responsive frame rates. In addition, we want to map data to per-node sketchiness, which means the style and degree of sketchiness can vary across different cuboids, requiring the simultaneous display of different degrees of sketchiness. Our implementation is split into two rendering techniques that can be efficiently processed independently, one for outlines and the other for hatches. However, we do not map data to them individually but always combine them to obtain an emphasized, coherent sketchy appearance.

We modeled and examined a small group of three distinguishable hand-drawn styles which we applied to the non-photorealistic representation of cuboids using outlines and textures [207]. These styles are ballpoint pen outlines (*biro*), outlines imitating colored pencils combined with slightly desaturated hatching (*pencil*), and thick outlines and hatches imitating text markers (*marker*), as showcased in Figure 3.6.

Sketchy outlines are synthesized by exaggerated graphical inconsistencies, which cause the impression of vagueness. We achieve this by multiplexing frequency and amplitude in a stylized, sketchy way [87, 227], using a technique with parameterized perturbation, overdraw, and texture.

Perturbation. The deviation or disturbance from a stable or steady state (*perturbation*) based on noise is used to modify the straightness of outlines. The maximum perturbation of a stroke correlates to the degree of sketchiness [35]. Amplitude, frequency or number of octaves, and type of noise are direct and easy-to-use parameters for controlling the degree of sketchiness (Figure 3.7). Rather than altering geometry, this is achieved more efficiently in image space [151, 179]. To avoid a "shower door effect" [169]—graphical elements appearing to be "swimming through [a] texture" [145]—edge-aligned billboards in object space [61] can be used. Instead, we use spatially anchored 3D simplex noise [188] as a source for perturbation intensity. This means that we use the spatial coordinates of an outline within the 3D space in which the treemap is embedded as input to the noise function.

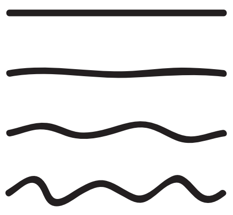


Figure 3.7: An example for increasing sketchiness (top to bottom) of a line using perturbation [L4].

Overdraw. The display of a single outline is not limited to a single stroke. Overlaying multiple strokes of the same texture but with varying perturbations (*overdraw*) can increase the degree of sketchiness. This can be done, e.g., by shifting the 3D noise look-up randomly for every stroke (Figure 3.8). Varying the intensity (pressure) per stroke can further increase the aesthetic appeal of a sketched outline without significantly affecting the perceived degree of sketchiness. It is essential to balance creativity with clarity and readability when using overdraw. Using too many overlaying strokes or style variations can be counterproductive, decrease visual clarity, and increase visual clutter. We suggest restricting sketchiness to a single, consistent style per treemap and keeping the maximum number of distinct lines for overdraw rather low [35].

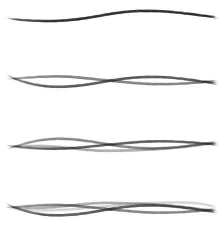


Figure 3.8: An example for increasing sketchiness (top to bottom) of a line using overdraw [L4].

Texture. Thickness and grain of a drawn line are influenced by the drawing tool, the pressure being applied, and the texture of the paper being used. Such characteristics can be captured and synthesized for real-time rendering using image-based or procedurally-generated textures. Textures

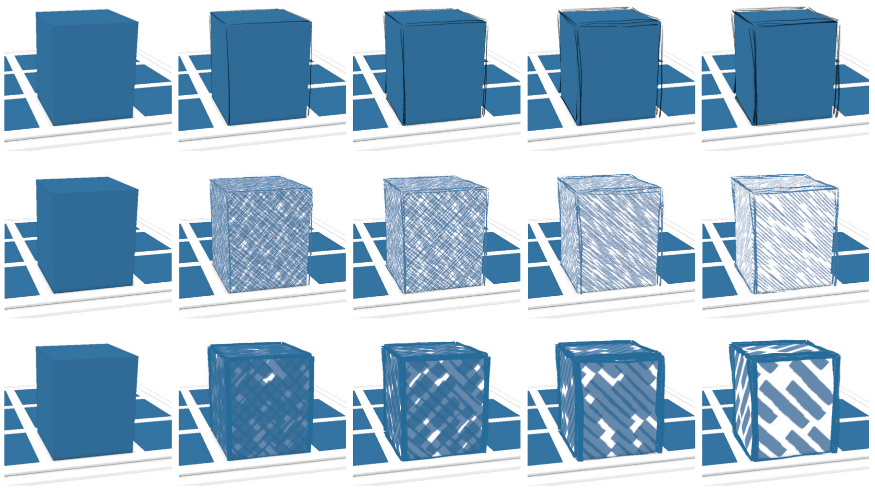


Figure 3.10: A cuboid rendered using sketchiness to encode "uncertainty, imprecision or vagueness" [L4]. The scale ranges from no sketchiness (left) to maximum sketchiness (right), using five distinct degrees of sketchiness. The number of overdrawn outlines is subsequently incremented, and the intensity of the hatching (if present) is designed to decrease evenly [L4].

can be sourced by scanning or photographing hand-drawn lines on paper (Figure 3.9). Minor variations of a style can be stored in a texture atlas to avoid unwanted repetition of patterns by randomly selecting a variation per stroke from the atlas. The style should match the anticipated treemap size or, more precisely, the typical screen size of cuboids displayed in that style. Small cuboids drawn with lines that are too broad or prominent will immediately introduce visual clutter. Less prominent styles should be used with increasing treemap size and node count. Unlike perturbation and overdraw, we use stroke textures to improve the sketchiness's aesthetics, not for direct data mapping.

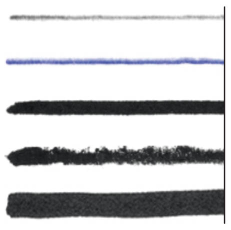


Figure 3.9: Examples of textures capturing different styles of sketchy lines based on different drawing tools. From top to bottom, graphite pencil, ballpoint pen, sharpie, crayon, and marker [L4].

3.2.2 Surface Hatching for Interactive Visualization

Hatching describes the use of lines or strokes to create shading or texture. Precomputed [189, 268] or recursive procedural *Tonal Art Maps* (TAMs) [229] can be used for this purpose. These sets of texture images with different stroke densities, i.e., hatch intensities, can be suitably blended during fragment shading. They also use specially adapted mipmaps to ensure consistent stroke intensities and widths in image space. Their parameterization is similar to sketchy outlines; it includes the fill style, stroke texture(s), and hatch intensity levels. Hatching and stippling can be used to convey texture, tone, and shape. Hatching involves creating parallel or crosshatched lines in closely spaced groups, while stippling involves using small dots or marks to create a similar effect.

Rather than representing light and shading information, the intensity of the hatching indicates the degree of sketchiness (Figure 3.10). This also makes it easier to maintain an existing thematic mapping, such as to color. The propagation of the principal direction of the hatching usually emphasizes the surface orientation [134, 151]. We have decided to exclude the direction or orientation of the strokes for data mapping for the time being, although it could be used for cuboids since they have zero curvature.

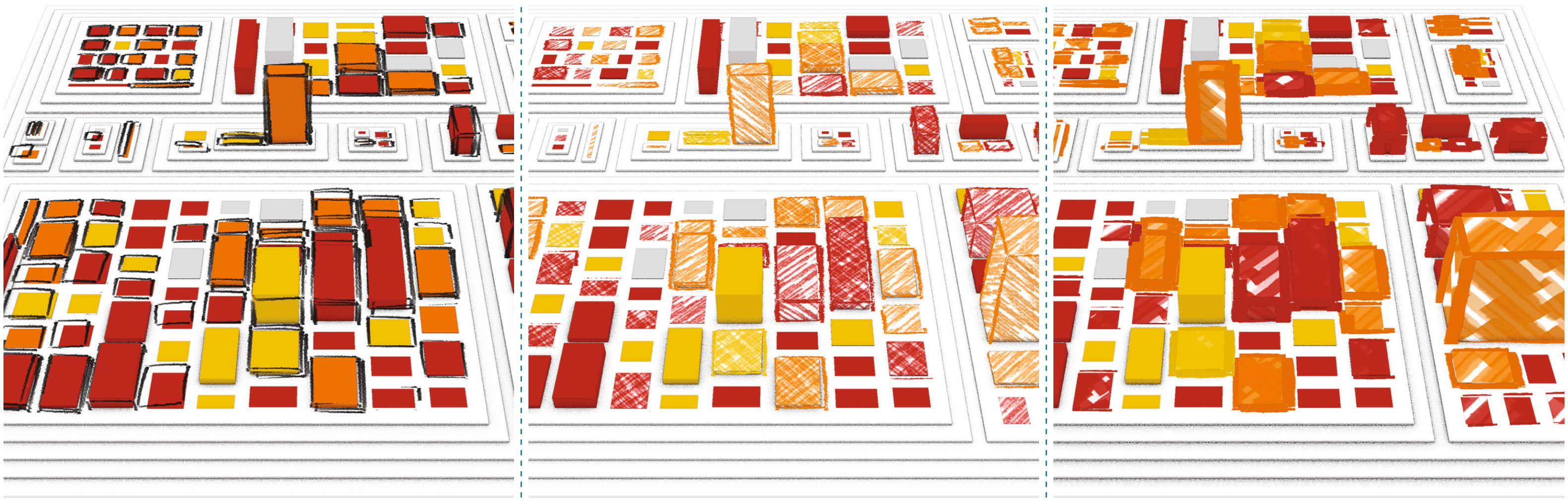


Figure 3.11: A 3D-embedded treemap applying three different styles of sketchiness. An additional attribute is mapped to the degree of sketchiness for each leaf node (including none) [L4].

3.2.3 Sketchiness as a Visual Variable in $\mathcal{A}^3$

Sketchy outlines and surface hatches can be combined into a promising candidate for an independent visual variable, *sketchiness* (cf. Figure 3.10). It appears to "have the capacity to carry information in its own right" [278] and allows us to map any ratio-scale data, while being most effective for ordinal data with a small range [L4]. When used in conjunction, the unique texture characteristics make it particularly useful for encoding *uncertainty*, *imprecision*, or *vagueness*. It can encode multiple, distinguishable degrees of sketchiness and be used both on demand and complimentary. It does not show substantial interference with other visual variables, such as color and height, likely due to the regular and simple shape of the cuboids.

Our user studies suggest that sketchiness shows similar qualities in $\mathcal{A}^3$ as it does in $\mathcal{A}^2$. Users were capable of recognizing an *order* when different degrees of sketchiness are depicted. Difference in style also matters, especially for accuracy in selective, pre-attentive processing. Here, for example, the pencil style significantly outperformed the marker style. We also confirmed one of our expectations, that the participants' ability to pre-attentively process sketchiness decreases as the number of elements increases. This underlines our remarks on increasingly large treemaps and favors an isolated, on-demand use: First, an exclusive use of outlines might lead to poor pre-attentive processing. Second, when using hatching, the TAMs' size and stroke-scales must be adjusted to the treemap size since our ability to pre-attentively process the individual nodes decreases.

Using five degrees of sketchiness, users could bring depicted elements in an order for multiple styles. It was even possible to account for shading without sacrificing the clarity of each individual degree. These results demonstrate that sketchiness is an effective technique for extending the expressiveness of 3D-embedded treemaps (Figure 3.11).

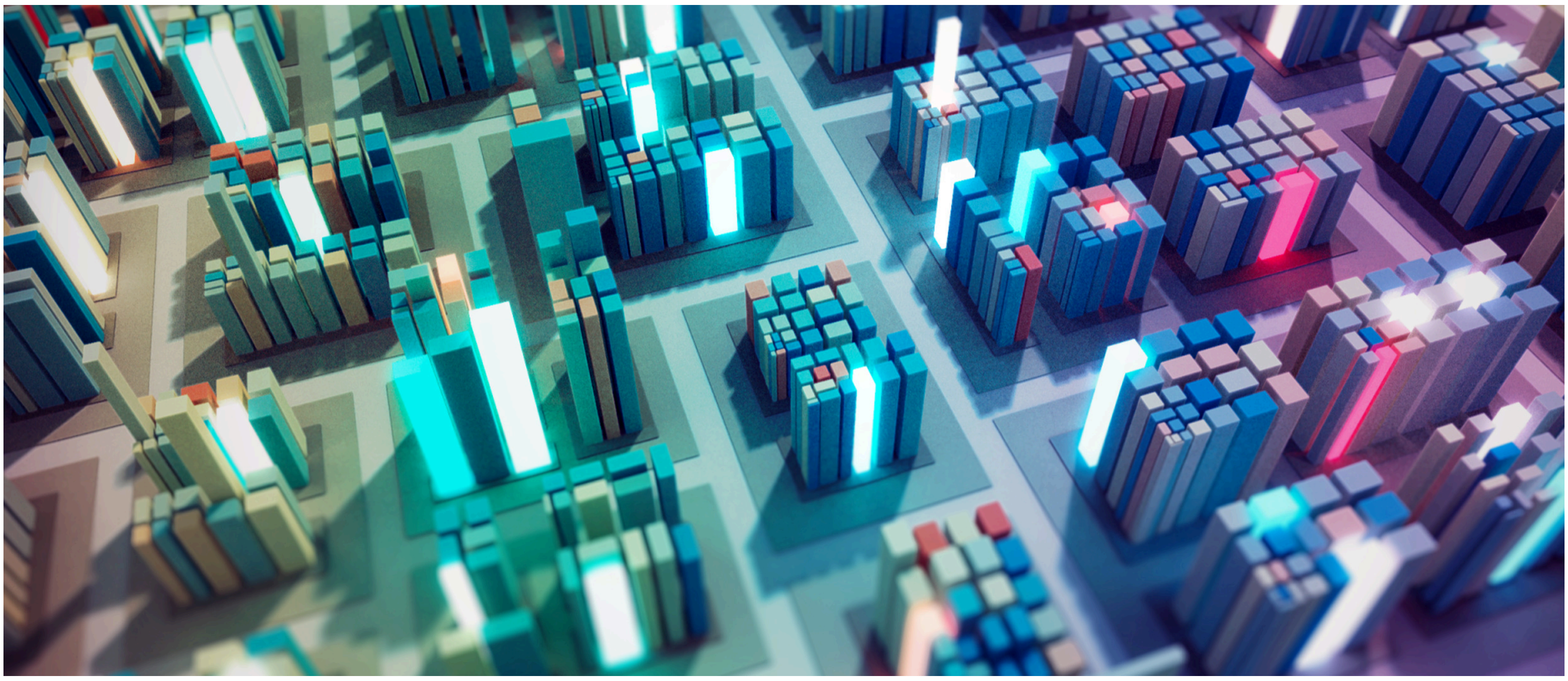


Figure 3.12: A treemap rendered using path-tracing for more accurate light propagation. It allows us to highlight individual cuboids using light-emitting materials for the cuboids' surfaces.

3.3 Physically-based Materials and Phenomena

The physical world provides us with numerous indicators of activity, such as heat or light. When something is active, we readily associate it with these two elements, for example, if something is glowing from heat. Similarly, if there is light in a building at night, we assume someone might inside (Figure 3.12). Starting with this idea, we investigated how such associations can be used as a visual variable. "When faced with unfamiliar concepts, our cognitive system searches for the best mapping between the unknown concept and existing knowledge of other domains." [284] If we leverage this knowledge, e.g., by using physical-based materials or phenomena, we might be able to make treemaps more intuitive and accessible to users.

This approach can be extended beyond the appearance of individual leaf nodes to neighborhoods of nodes or inner nodes. Like activity, whole areas within the treemap could be influenced by aging, degeneration, or desolation in terms of their associated materials, or even destruction within their environment. This includes being overgrown by grass, gathering dust, and being exposed to natural forces such as fire, rain, wind, or natural disasters. Using natural phenomena to create visuals immediately triggers common sense warnings and reminds us to be cautious and considerate of others when using such an approach. The mere extrapolation of possibilities and scenarios highlights the potential impact of visual communication and the emotions it can evoke in the viewer. Therefore, we must use visuals thoughtfully and deliberately, considering not only their aesthetic appeal but also their potential impact on the viewer.

This chapter's second spotlight is on physically-based materials and phenomena and describes how they can be used as a visual variable in $\mathcal{A}^3$.

We briefly discuss roughness, shininess, rustiness, and emissive light and, w.r.t. weather phenomena, summarize how animated rain and fire can be integrated. These metaphors can be used to emphasize trends within the data, and enable us to "emotionalize the visual communication by providing memorable visualizations." [34, L2]

3.3.1 Physically-based Materials for Visualization

In real-time rendering, the material of a surface determines how light interacts with it. Over the past few years, rendering engines have moved away from simplified, highly approximative *Bidirectional Reflectance Distribution Functions* (BRDFs), such as Lambertian, Oren-Nayar, or most prominently Blinn-Phong. Most rendering engines have shifted towards quasi-standardized, physically more accurate approximations, commonly referred to as *physically-based rendering* (PBR) [2]. This shift is accompanied by a consolidation of more accurate BRDFs,[3.1] their parameterizations, computations, material libraries, and respective availability.[3.2]

When we identify suitable metaphors that can communicate an object state we pair them with a contrary phenomenon and take advantage of the human visual system to rapidly process visual cues such as shading and textures [106, 107]. For instance, the roughness of a surface can serve as a suitable metaphor to visually connotate the up-to-dateness of the data depicted. High surface roughness can convey a negative state, data being old or out-of-date, and can be optionally amplified by texture-based and geometry-based displacement, resulting in a rough appearance. For the opposite end of that spectrum, low surface roughness in combination with a high specularity can be used to create a shiny and clean cuboid, resulting in a positive state; that is, the depicted data is up to date (Figure 3.13).

The following briefly summarizes the technical characteristics of *roughness*, *shininess*, *rustiness*, and *emissiveness* (by means of radiant emittance in the visible electromagnetic spectrum). It assumes the use of a physically-based material system with a metal-roughness parameterization commonly available, for example, in the *Unreal Engine 4*,[3.3] as used by Würfel et al. [L2], as well as in popular WebGL and WebGPU based rendering systems.

Rustiness can be achieved by using a material or texture depicting a rusty surface. A threshold and a procedural noise-based mask are used to control the exposure of rust (Figure 3.14).

Roughness is initially created by increasing the material's respective roughness parameter. However, by combining geometry tessellation, normal mapping, and vertex displacement, it can be increased further. Depending on the required degree of roughness, noise-based

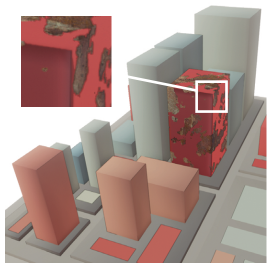


Figure 3.14: A cuboid with a third of its surface exposing a rusty material.

[3.1] Brent Burley. *Physically Based Shading at Disney.* SIGGRAPH '12 Course Notes, disneyanimation.com/[../...]disney_brdf_notes_v3.pdf. 2012.

[3.2] Lucasfilm Advanced Development Group. *Material X: An Open Standard for the Exchange of Rich Material and Look-development Content Across Applications and Renderers.* materialx.org. 2017.

[3.3] Epic Games, Inc. *Unreal Engine 4.* unrealengine.com. 2015.

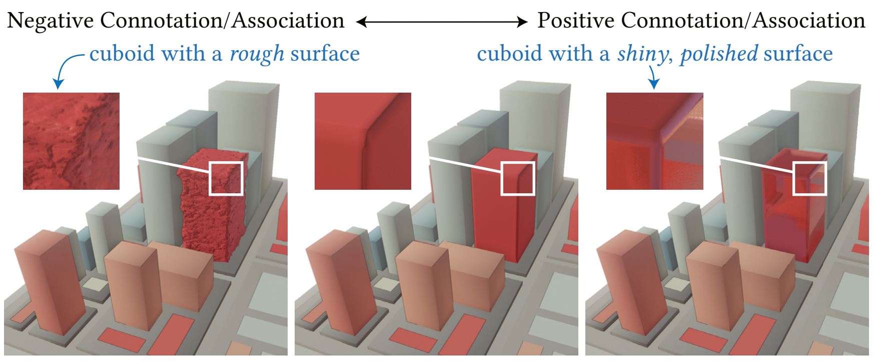

Negative Connotation/Association
Positive Connotation/Association
cuboid with a *rough* surface
cuboid with a *shiny, polished* surface

Figure 3.13: An example of adding a connotation to the data mapping using a rough-to-shiny metaphor. This metaphor communicates deterioration or negative deviation (roughness) and improvement or positive deviation (shininess) without need for further introduction of the mapping semantics [L2].

normal mapping (not affecting the actual geometry) or tessellation with subsequent vertex displacement can be used.

Shininess is created by increasing the metalness of a physically-based material while decreasing its roughness, i.e., increasing its smoothness. In doing so, the object will reflect more of their surroundings, making them appear more reflective and shiny.

Radiant Emittance is modeled by assigning a cuboid's color to its emissive color, multiplied by an emissive factor. Visually appealing emissiveness either requires path-tracing or adequate emulation of light bleeding (bloom) using image-based post-processing.

Concerning the 'rough-shiny' spectrum, a simplified, more efficient approach exists that is especially easy to integrate for web-based applications of treemaps. Instead of a material system, image-based lighting with just a single, precomputed texture is used: an Irradiance Environment Convolution Cubemap or *Prefiltered, Mipmapped Radiance Environment Map* (PMREM).[3.4] This environment map is convoluted with more scattered sample vectors for increasing roughness levels, creating blurrier reflections. The sequentially blurrier results for increasing roughness levels are stored as mipmaps of the cubemap and can be easily fetched using roughness as the level-of-detail. The cube map and its custom mipmaps can efficiently be provisioned by and reconstructed from a single image [185].[3.5]

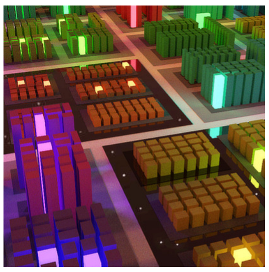


Figure 3.15: Physically simulated light emittance used as visual variable.

Light radiating, 'glowing' cuboids support pre-attentive processing and are especially useful for emphasizing and highlighting purposes. Physically simulated, emissive light increases the visual quality, is aesthetically pleasing (Figure 3.15), and never fails to impress engineers of real-time rendering systems. Classical highlighting techniques, such as simple color modulation and outlining, can provide sufficient means for highlighting and are easy to implement, making them a suitable fallback option, particularly for interactive, web-based visualization.

[3.4]Emmett Lalish. *Fast, Accurate Image-Based Lighting.* drive.google.com/[..]/PMREM-submission.pdf. 2020.

[3.5]Daniel Limberger and Philipp Otto. *Simplified Transmission of a Cubemap with Custom Mipmaps using a Single Texture.* webgl-operate.org/[..]/ibl-map.png. 2020.

3.3.2 Weather Phenomena for Visualization

Humans frequently use linguistic metaphors in idioms that involve natural phenomena, such as 'to go up in flames' or 'grow grass upon'. They can be suitable for communicating additional information, for example, trends underlying the depicted data. Natural phenomena, such as weather effects, can affect the area surrounding treemap items or a group of adjacent items. Phenomena such as fog, haze, clouds, rain, or even fire can be used to visually encode additional information for data mapped to multiple nodes or comprised by inner nodes. Panas et al. used this approach to extend the use of a realistic 3D-city metaphor for depicting software back in 2003 [186]. They assigned quality metrics to building textures and incorporated visual effects such as fire and bolts to highlight hot spots in code execution and frequent component modifications. All of these phenomena can take on many different forms and intensities. Including them in visualization design can help emphasizing or directing attention to regions of interest within a map (Figure 3.16).

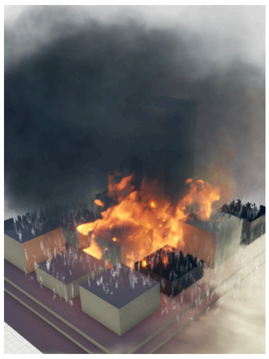


Figure 3.16: Superposition of the weather phenomena rain, clouds, and fire within a tiny treemap.

Incorporating weather phenomena into data visualization requires sophisticated rendering techniques and preferably continuous rendering for animations, contrasting with the interactive but progressive generation of images we generally use (cf. chapter 5). For example, clouds, rain, and fire can be created using particle systems that combine physical simulations of particle movement, textured billboards, and appropriate blending. However, refining and configuring particle systems to match the intended phenomenon and aesthetic is not trivial and requires creative tweaking of many aspects of the systems. More modern rendering engines allow for volume-based simulations, offering clearer parameterization and more direct control. Nevertheless, it is difficult to estimate to what extent the realism of the phenomena affects visual communication. A glyph-based approach, e.g., showing a flame or cloud emoticon, might be sufficient or even better suited in many cases.

3.4 In-Situ Templates

When visualizing data, we assume a fixed temporal reference point, i.e., all data points refer to the same temporal context. To better understand the evolution and changes in the data, it may help to understand how the data arrived at its current state and the extent to which it has changed recently. For simplicity and without limiting generality, we assume a temporally coherent tree structure, e.g., using a union tree [244] that considers all nodes within a range of revisions [L22, 206]. When working with visualizations, we likely experienced the most straightforward approach; we just switch interactively between revisions and try to detect changes. Even assuming that the treemap layout would be stable [20, 254], this task is error-prone and quite tedious.

A side-by-side comparison or *small multiples* in case of more than two revisions [47, 205] allow us to capture the overall evolution of the data and identify points of interest throughout time. However, to locate individual data changes, two or more maps must be compared meticulously, which

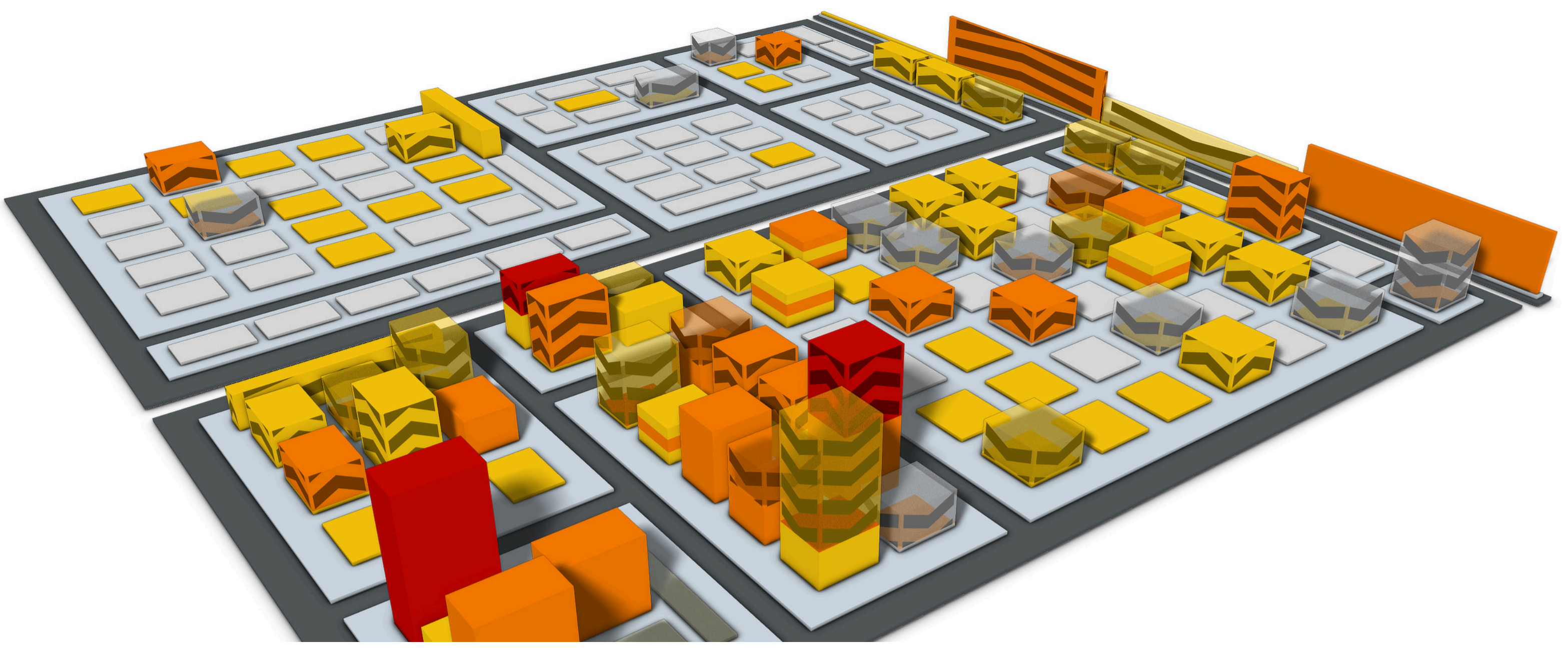


Figure 3.17: Example of a 3D-embedded treemap using in-situ templates for a simultaneous two-state mapping to height and color. The implementation of templates can be challenging. For example, the display for a drop in height (cf. I_{12} in Figure 3.18) uses procedural arrow textures [L21] and order-independent, stochastic transparency [68].

is again error-prone and tedious. If our interest is in reading individual changes accurately, the more straightforward approach is to map changes themselves to visual variables. For example, the absolute difference between the two states of a node can be mapped using height and the sign of the difference using color. Alternatively, one can map the values of one point in time to height and the degree of change with respect to another point in time using a five-color divergent palette. We refer to this type of explicit encoding of changes of two data states as *change mapping*. With this approach, however, changes are depicted with limited context; we cannot accurately deduce the value of both states.

This chapter's third spotlight showcases graphical elements that convey alterations in color, height, and area directly 'in place' (*in-situ*). We introduce cuboid geometry templates that allow two attribute values, e.g., of different time references, to be (1) mapped simultaneously while also (2) emphasizing their direction of change.

3.4.1 **Two-State and Multi-State Mappings**

Our objective is to visually represent the corresponding values of an *original state* and a *comparative state* of a data element simultaneously. We assume that the different states are individual attributes that share the same semantic and data characteristics. These can include but are not limited to different temporal references of a measurement, thresholds, nominal values, or precomputed statistical values. Consequently, we avoid using designations such as 'before' and 'after' or 'former' and 'latter.' Simultaneously mapping attribute values of two states enables direct comparison and an unbiased, equivalently accurate read-out of the difference. We refer

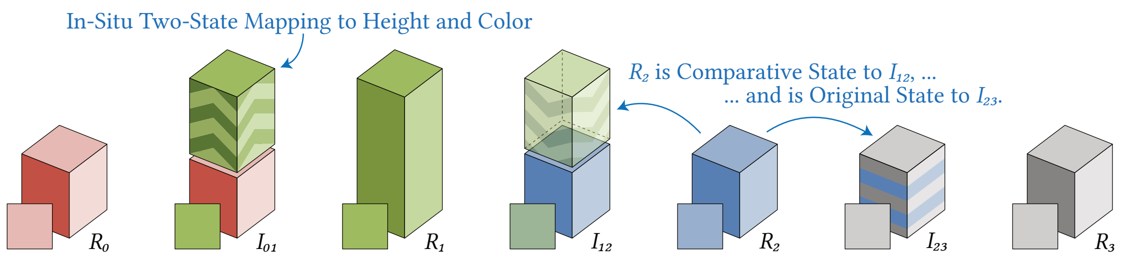

In-Situ Two-State Mapping to Height and Color
R_2 is Comparative State to I_{12}, ...
... and is Original State to I_{23}.
R_0
I_{01}
R_1
I_{12}
R_2
I_{23}
R_3

Figure 3.18: Illustration of our preferred in-situ templates for simultaneous two-state attribute mapping using color and height. Given are the four revisions R_0 to R_3 for two attributes encoded by the color and height of a node. For example, the in-situ template I_{01} shows an increase in height and a color change when R_0 is compared to R_1. The squares on each template's bottom left illustrate how the template would look from above (2D treemap scenario). The small 'levitation gaps' in I_{01} and I_{12} pursue an aesthetic function and have been added due to user feedback.

to this as *two-state mapping*, while a mapping with more than two states would be a *multi-state mapping*. In contrast to a *change mapping*, a two-state mapping can implicitly or explicitly encode the difference visually but is (1) not limited to the specific semantic of that difference and (2) ensures the two attribute values to be encoded explicitly.

Cuboids can be augmented to encode height, color, and area differences, one at a time or all together in-situ [L15]. These templates can be used as graphical elements for intra-element and inter-element mapping. For visual comparison, various approaches can be used, such as juxtaposition (placing graphical elements side-by-side), superimposition (overlaying graphical elements on top of each other), explicit encoding, and animated transition for the visual display of original and comparative states [85, 144]. Our templates are designed to be superimposed onto a single treemap and use any of the three methods for visual comparison within their local scope. Conceptually, our approach shares similarities with *property towers* by Steinbrückner et al. [225] and *bricks* by Wettel et al. [271], as they are both capable of encoding multiple states or even the composition of an attribute. Other approaches, such as variants of *data vases* [235] and glyphs [110, 220], are similarly promising but deviate more from the cuboid appearance.

We identified significant issues with many templates that initially seem plausible and obvious. These issues are primarily related to reading direction or orientation and occlusion. Furthermore, we account for the top view of each template (basically resulting in a 2D treemap) that we prefer to support for orientation and overview in our 3D-embedded treemaps. Since minor modifications and variations can lead to a multitude of templates, we focus on exemplary ideas and emphasize their respective problems while advocating for the most promising templates (Figure 3.18). Even though templates for the simultaneous two-state mapping to height, color, and area have been developed [L15], we only discuss templates that support color encoding, height encoding, and both simultaneously.

3.4.2 Two-State Height Mapping

Given are the two states R_4 and R_2 as original and comparative states, respectively, as shown in Figure 3.19. The first template, A, exhibits a

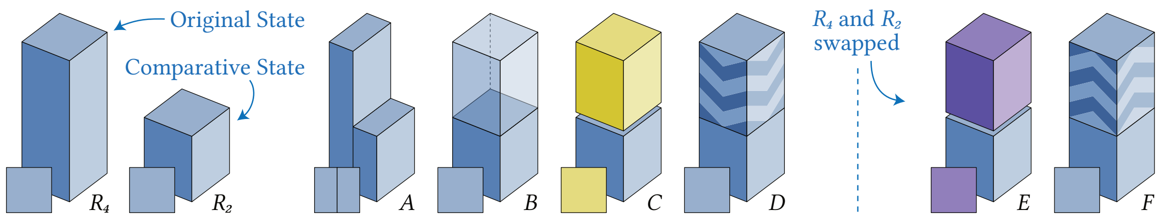


Figure 3.19: Excerpt of in-situ templates for multi-state mapping to height, shown on the example of R_4 and R_2.

side-by-side arrangement of two cuboids, most comparable to plain height mapping. However, it is not rotation-invariant [231], and navigation may invalidate the reading direction. This can be remedied by dynamically adjusting the template orientation during navigation or restricting the virtual camera position and view direction to a predominantly south-north direction. Template *A* is also problematic because it is designed for rather squarified cuboids, which is often not the case. Depending on the layout, cuboids with more or less unfavorable aspect ratios often appear on larger maps (Figure 3.17).

Template *B* is more promising, showing the negative difference to the original state as a transparent cuboid. However, this requires dithering, depth sorting, or *order independent transparency* (OIT) [68], which can introduce visual clutter and degrade depth cues if done poorly. Moreover, it does not work for positive differences (e.g., swapping R_4 and R_2).

Templates *C* and *E* map the sign of the difference to distinct colors, e.g., yellow and violet for negative (decrease) and positive (increase), respectively. At the same time, this approach is comparatively easy to implement and understand, provided that the colors do not interfere with the treemap's actual color mapping. Templates *D* and *F* replace color with explicit directional encoding using procedurally generated arrow patterns on the lateral faces of the cuboid's difference area.

We prefer *B* with *D* for negative differences and *F* for positive differences. In addition, we use a *levitation gap*, a small gap that has a constant height in pixels, i.e., independent of the distance between the gap and the virtual camera; This is done for aesthetic preferences and helps to discern minor differences better. It can be combined with all templates except *H* to *N*.

3.4.3 Two-State Color Mapping

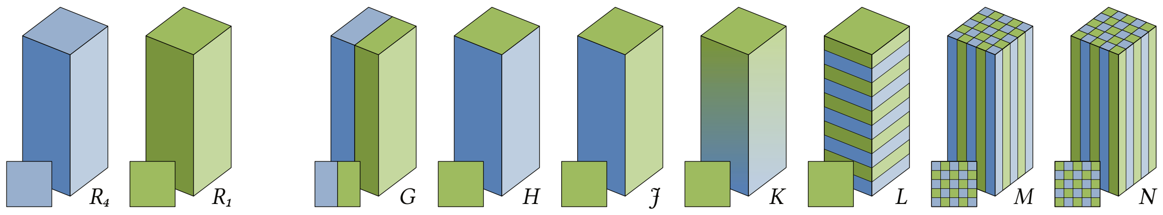


Figure 3.20: Excerpt of in-situ templates for multi-state mapping to color, shown on the example of R_4 and R_1.

Template *G* splits the cuboid vertically, with each half distinctively encoding one of the two attribute values by its color. This approach is analogous to A, which is problematic for reading direction and elongated aspect ratios. It should be noted that this template is applicable only if padding between siblings is applied, which, however, we assume for all 3D-embedded treemaps; otherwise, the halves of a split cuboid might be indistinguishable from two contiguous cuboids (also applies to template A for that matter).

Template H encodes the comparative state on the top face. K extends this to two opposite lateral faces. In both cases, the reading direction is unequivocal. However, depending on the camera's position and field of view, some cuboids might only expose a single face to the viewer, making it unreliable for interactive exploration in 3D. Furthermore, only one state can be encoded when the cuboid's height is zero (or viewed from above).

Template K uses a vertical color gradient from the original to the comparative state, ending with the comparative state on the top face. If not used properly, the color gradient can become incomprehensible and introduce colors that cannot be mapped back to the data. Intermediate colors must be computed by first interpolating the attribute values and then map the resulting values to the treemap's color scale.

Alternatively, color patterns based on procedural texturing can visually encode differences between two states. Template L (and I_{23}) uses a horizontally aligned stripe pattern on the lateral faces to alternate between the two states. This template is rotation-invariant and can be read even when partially occluded. Like templates H and K, it does not work for cuboids with zero height. Using a vertically aligned stripe pattern and a grid on the top face, as shown with templates M and N, addresses this issue but poses problems with reading direction and elongated aspect ratios. All considered, we prefer template L (Figure 3.20) among all the color templates shown. The use of stripes (1) allows for comparing differences between cuboids more accurately, (2) makes it less susceptible to occlusion, (3) employs an unambiguous reading direction (top face is comparative state), and (4) facilitates the extension to procedural arrow patterns. The width of the stripes must be carefully configured to ensure the two colors are discernible with respect to the cuboids' size on the screen.

3.4.4 Two-State Height and Color Mapping

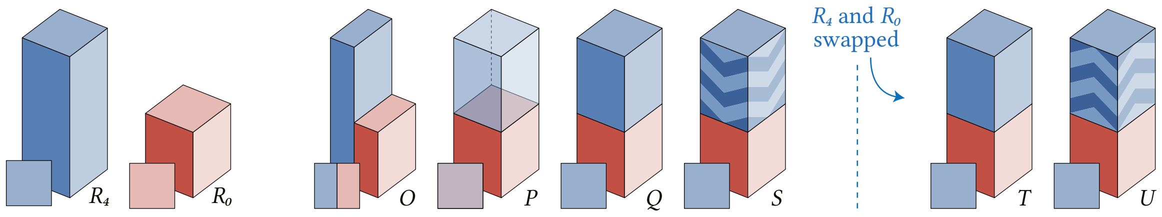


Figure 3.21: Excerpt of in-situ templates for multi-state mapping to color and height, shown for R_4 and R_0.

To simultaneously encode two states for height and color, we noticed that most height templates (exceptions are C and E) can be directly augmented with a second color; A becomes O, B becomes P, and D and F become S and U, respectively (Figure 3.21). The template Q cannot be used because the resulting cuboid remains the same regardless of whether R_4 or R_0 is used as the comparison state (T).

With all the above observations on templates, the following Table 3.2 lists our preferred combination of templates for the different cases in concurrent two-state mapping to height and color:

	$\text{color}_1 = \text{color}_0$	$\text{color}_1 \neq \text{color}_0$
$\text{height}_1 = \text{height}_0$	–	L
$\text{height}_1 > \text{height}_0$	F	U
$\text{height}_1 < \text{height}_0$	B & D	P & S

Table 3.2: Matrix showing our in-situ template preferences for a simultaneous two-state mapping to color and height. The state attributes are indexed as 0 and 1 for the original and comparative state, respectively. When a non-qualitative color mapping is used, the color scale itself encodes the sign of the difference of the values mapped to color.

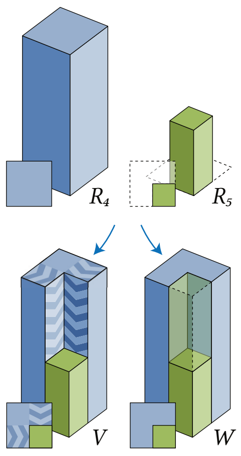


Figure 3.22: Two in-situ templates for simultaneous multi-state mapping to color, height, and area, shown for R_4 and R_5 [L15].

Templates for coincident mapping of height, color, and area in two states have been developed [L15] but are not covered in this thesis. To give an idea of how this might look, Figure 3.22 shows two templates that support this purpose. Each template, V and W, encodes nine pieces of information in-situ, i.e., three value pairs and the sign of their differences. The more attribute values comprised in-situ, the more training and focus are required for the user to read even a single node.

Instead of replacing many cuboids of the treemap with these templates, we can place them next to the treemap on the ground plane of our 3D virtual environment. There, these and other templates could dynamically illustrate two-state mappings for a few selected nodes using separate, perfectly square, and richly labeled representations.

3.5 Animated Procedural Textures

When the assignment of an attribute value to visual variables of a treemap changes, we can make the change more pleasant by not switching instantly. Instead, we can animate the transitions between the visual representations of the respective values. This can be done by creating animated transitions for height and color from the former to the latter state. Drawing on our findings for the in-situ display of differences [L15], we investigated how animated transitions can support and highlight temporal changes within a series of attribute values [L21]. We developed our in-situ templates for static display, primarily to support image-based sharing or printing. These templates encode changes statically, but with a bit of imagination, they can be read as building instructions for designing animated transitions. With this in mind and considering our arrow patterns, we explored texture patterns and their capacity to animate a change in color mapping.

This chapter's fourth spotlight focuses on animated transitions for color mapping using procedurally generated texture patterns within $\mathcal{A}^3$. We highlight four patterns and their encoding and compatibility characteristics,

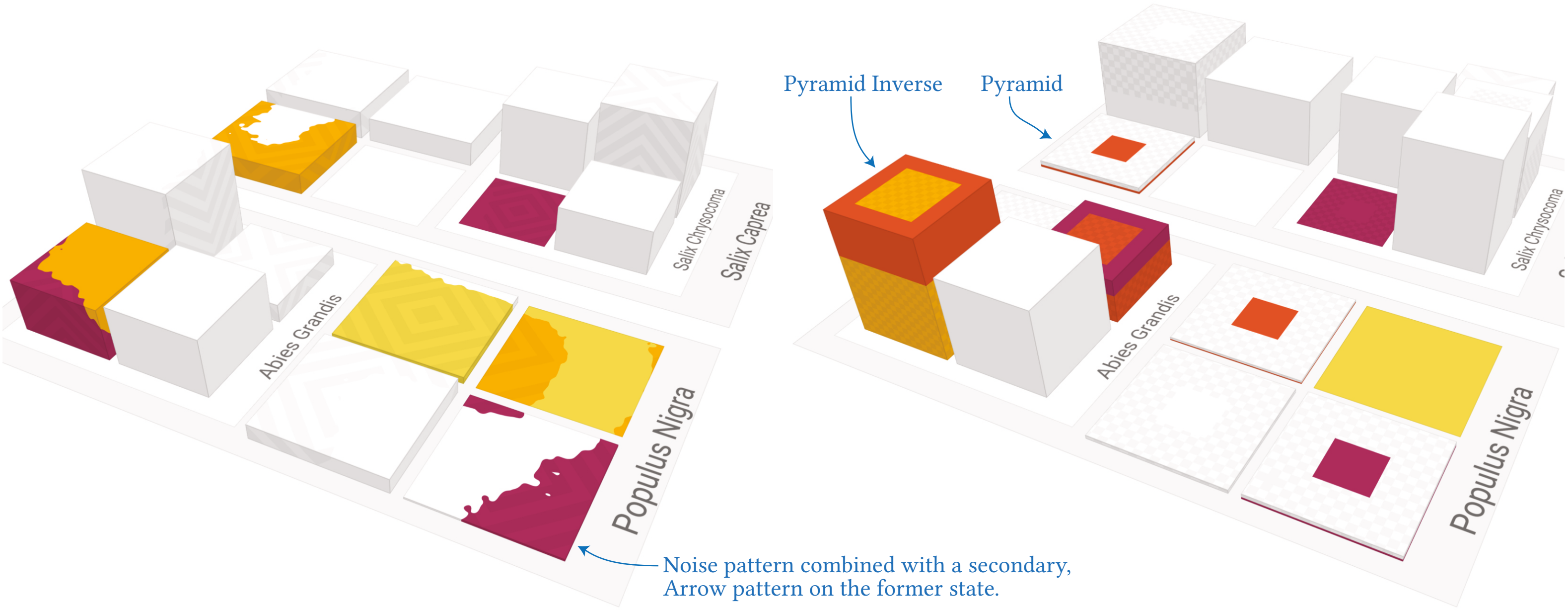


Figure 3.23: A synthetic treemap with n revisions of random attribute values is shown during two animated transitions between successive data revisions. The examples each show a combination of two procedural patterns used together to animate transitions between revisions; a noise pattern supported by arrows (left) and a pyramid pattern supported by squares. The secondary patterns emphasize the direction of change during animation and provide encoding for static display [L21].

showing how they can convey data changes while maintaining a visually appealing and easily interpretable representation.

3.5.1 Transition, Animation, and Change

Transition describes the change from a node's state or mapped value to another, while an animation is a continuous progression over multiple states through transitions. Transition does not encode time or duration but only progress $t \in [0, 1]$ where 0 marks the start and 1 the end of a transition. The control value t is then mapped to the *pattern progress* $\lambda \in [0, 1]$. This separation allows tweaking the linear mapping $\lambda = t$ using, for example, easing functions for aesthetic and user-experience purposes [119].

A pattern maps the transition progress for a fragment on a cuboid's surface, resulting in a binary per-fragment color choice. This can be achieved using *algorithmic drawing*,[3.6] *signed distance functions*,[3.7] or *procedural textures* [93], all of which can be implemented in fragment shaders. These patterns are applied to the top and lateral faces of the cuboids, allowing both colors to be shown on the surface. Let $C = [0, 1]^3 \subseteq \mathbb{R}^3$ denote the unit cube in the Euclidean space $\mathbb{R}^3$ that is used to provide a local parameterization of a treemap node. Its surface S is given by the boundary of C, i.e., $S = \partial C$. A pattern on the surface S is a function given by

$$P : S \times [0, 1] \to \{c_f, c_l\}, (p, \lambda) \mapsto P(p, \lambda) = P_\lambda(p), \qquad (3.1)$$

[3.6] Patricio Gonzalez Vivo and Jen Lowe. *The Book of Shaders.* thebookofshaders.com. 2015.
[3.7] Inigo Quilez. *Computer Graphics, Mathematics, Shaders, Fractals, Demoscene and More.* iquilezles.org/articles/. 1994.

where c_f and c_l are colors representing the value of the former state and the latter state. Furthermore, a pattern P satisfies the following properties:

- $\forall p \in S, \forall \lambda \in [0,1] \, : \, P(p,\lambda) \in \{c_f, c_l\}$,
- $\text{area}(\{P_0 = c_l\}) = 0$ and $\text{area}(\{P_1 = c_l\}) = \text{area}(S)$, and
- $\forall \lambda_1, \lambda_2 \in [0,1] \, : \, \lambda_1 < \lambda_2 \implies \text{area}(\{P_{\lambda_1} = c_l\}) \leq \text{area}(\{P_{\lambda_2} = c_l\})$,

where area denotes the usual area of surface in $\mathbb{R}^3$ and the set $\{P_\lambda = c_l\}$ denotes the subset points on S, where the pattern takes the value c_l. The first property (1) states that for any point on the surface and any pattern progress value, the pattern function will output either the former or latter color, ensuring no mixed colors are introduced. Concerning the latter color, the properties further describe that (2) at the beginning, none of it is visible, and at the end, the entire surface is covered by it, and (3) in between, the covered area steadily increases as the pattern progresses advances.

From the perspective of the visual variable, animation always transitions between two states of that visual variable. We previously describe these states of such a two-state mapping as *original* and *comparative* states, focusing on the in-situ comparison. Procedural textures supported the in-situ templates and emphasized the direction of change for the spatial dimensions, such as a change in height or area. Now, we shift our focus to change display, switching to *former* and *latter* state. The goal is an encoding that allows for identifying the former and latter state, magnitude, and direction of change. We identify these as the *direction of animation*, the *magnitude of change*, and the *direction of change*, respectively.

The representation of change should visually correspond to the change in data (*visual-data-correspondence*) [135], meaning the actual difference between two values and that difference's sign. Animating transitions by interpolating height and color attributes is an evident and sound approach [30, 146]. We developed procedural texture patterns to emphasize and support animated transitions in color, as demonstrated in Figure 3.23 [L21, L22]. The introduction of secondary, supportive patterns allows for an accurate display of change direction, even for a static display of an image. There are more aspects to adopting treemaps over time, such as layout stability [92, 206, 244, 255]. Our approach aims to augment height and color mappings already in use and rely on the user's existing understanding of treemaps while emphasizing changes in the data.

3.5.2 **Procedural Patterns for Animated Change Display**

We explore transition-aware texture patterns designed to be effective for static images and dynamic animated transitions. These patterns mainly differ in their appearance, granularity, and the surfaces they are intended for. We have proposed seven variants in a previous study [L21], but in this work, we will focus on four: (1) *Pyramid*, (2) *Squares*, (3) *Noise*, and (4) *Arrows* (Figure 3.24).

The patterns are intended as conceptual starting points, not as a comprehensive list of possible patterns. Additionally, a pattern is not precisely

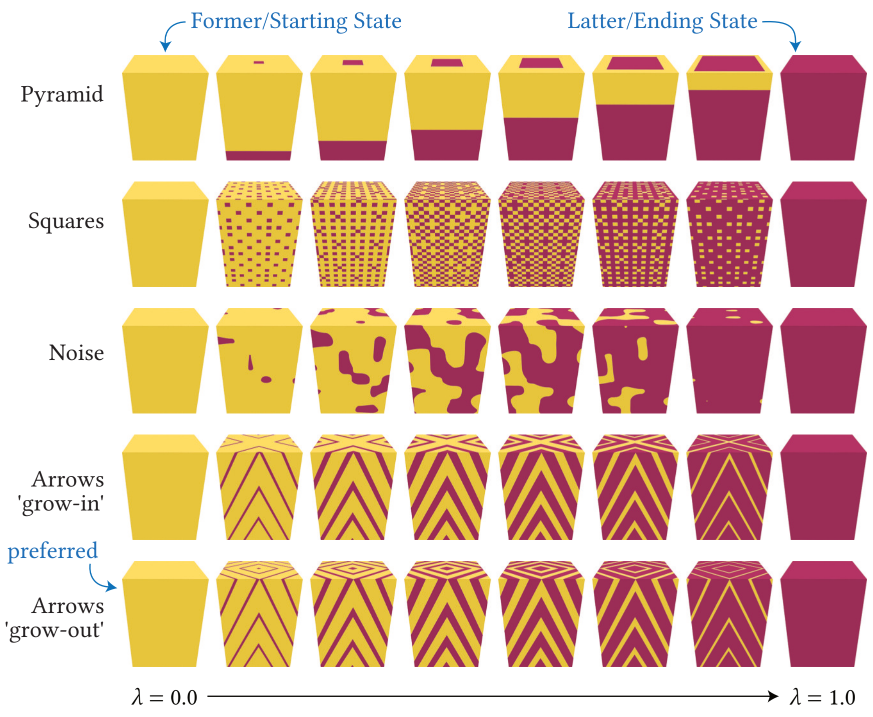


Figure 3.24: Excerpt of procedural patterns [L21] for changes encoded in color. The 'grow-out' arrow pattern is our preference over the 'grow-in' variant since the top-face direction matches growth (and reduction when inversed).

defined by a specific implementation or a fragment-precise definition of the binary decision to be made. Instead, we aim to capture patterns that differ and represent alternative approaches. All patterns, however, encode the progress by the area ratio of the colors representing the two states.

Pyramid. This pattern builds upon the *Pillar* pattern—latter color growing from bottom to top using the cuboid's lateral faces, and only at the very end, covering the top face—by including the top faces. Its name describes the metaphorical process used to derive an implementation of the pattern: a virtual pyramid is embedded in the cuboid and slowly pushed towards and through the top. All cuboid fragments intersecting with the pyramid are assigned the latter upcoming color. This intersection surface corresponds to the pattern's progress for lateral faces as well as for the top face. It can also be inverted to encode the direction of change.

Squares. This pattern is similar to per-fragment dithering. When used on high-dpi displays or in print, dithering may result in the visual blending of the two colors, potentially making the colors difficult to distinguish or introducing colors outside of the used color scale. We use a scalable dithering pattern that is scaled and applied to the cuboid's faces in world space rather than display pixels. This reduces the visual blurring of colors while providing a transition. Designing the overhang from a square onto neighboring faces is challenging from an implementation standpoint. A straightforward approach is to arrange the squares and loosen their aspect ratio to create rectangles, so no overhang occurs.

Noise. This pattern employs 3D Perlin noise [188] and a threshold to partition the surface into the two colors, resulting in organic-looking surfaces and transition behavior. The scale of the noise must be adjusted to the treemap's size. We derive the scale based on the number of nodes in the tree, as this measure loosely corresponds to the share of the footprint area one node has in relation to the size of the whole treemap.

Arrows. This pattern generates the appearance of alternating arrows of different shades and includes the top face as well (full). The arrows can be inverted to encode the direction of change. However, the top face's direction is not unambiguous: assume the arrows point from the middle on outwards. Considering only the top face, this will most likely be read as an increase. When considering these arrows crossing into the lateral faces of a cuboid, they suddenly indicate the opposite, a decrease. Since we explore the 3D-embedding of treemaps, we favor the latter variant since it considers the cuboid as a whole.

We propose a set of characteristics to capture the patterns' abilities for information encoding. Furthermore, we suggest superimposing two patterns to enhance the change direction for a static display, such as when the transition progress is at $\lambda = 1/2$, halfway through the transition.

3.5.3 Pattern Characteristics

We introduced five encoding characteristics and four compatibility characteristics for the texture patterns. The encoding characteristics can be summarized as follows:

$\nabla_{Monotonic}$ Every point of the surface transitions from former to latter color only once, ensuring visually consistent transition process (smooth) without flickering.

$\nabla_{DirStates}$ Unambiguously discerns former and latter states in a static image or during paused animation. This considers different viewpoints and the potential occlusion of parts of a cuboid.

$\nabla_{DirChange}$ Unambiguously identifies the direction of change in value (increase or decrease) in a static image or paused animation. This considers different viewpoints and the occlusion of parts of a cuboid.

$\nabla_{RatioLat}$ The ratio of colors on lateral faces matches a transition's progress. Lateral faces' area scales with the value mapped to height and partially with the value mapped to weight.

$\nabla_{RatioTop}$ The ratio of colors on the top face matches a transition's progress. The top face area scales with the value mapped to weight, making it relevant for top view or 2D treemaps.

The compatibility characteristics are:

$\nabla_{IndHeight}$ Mapping of change difference and change direction does not compromise a height mapping, including a difference being mapped.

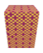
	Encoding – ∇E						**Compatibility** – ∇C				
Pattern	$\nabla_{\text{Monotonic}}$	$\nabla_{\text{DirStates}}$	$\nabla_{\text{DirChange}}$	∇_{RatioLat}	∇_{RatioTop}	$\sum_{\nabla E}$	$\nabla_{\text{IndHeight}}$	$\nabla_{\text{IndWeight}}$	∇_{TreeSize}	$\nabla_{\text{Polygonal}}$	$\sum_{\nabla C}$
Pyramid	✓	–	–	✓	◦	2.0	◦	✓	✓	◦	3.0
Arrows	✓	◦	✓	✓	✓	4.5	✓	✓	◦	–	2.5
Noise	✓	–	–	◦	◦	2.0	✓	✓	◦	✓	3.5
 Squares	✓	–	–	✓	✓	3.0	✓	✓	◦	◦	3.0
Py. + Sq.	✓	✓	✓	✓	◦	4.5	◦	✓	✓	◦	3.0
No. + Ar.	✓	✓	✓	◦	◦	4.0	✓	✓	◦	◦	3.0

Table 3.3: ✓ supported | ◦ partial support | – unsupported. Revised evaluation [L21, L22].

$\nabla_{IndWeight}$ Mapping of change difference and direction does not compromise the weight mapping, including a difference being mapped.

$\nabla_{TreeSize}$ The pattern remains meaningful and readable across varying treemap sizes, i.e., it adjusts easily for large or very large treemaps while retaining other characteristics.

$\nabla_{Polygonal}$ The pattern is suitable for treemaps with polygonal shapes, maintaining its visual metaphor and implementation. This also allows the pattern to be applied to extruded, non-rectangular shapes.

Table 3.3 gives an overview of the patterns and their encoding and compatibility characteristics. It is based on technical limitations on a feature level and poses constraints for using proposed patterns. However, additional limitations might be imposed by user expectations or user experience. The scores $\sum_{\nabla C}$ and $\sum_{\nabla E}$ are the weighted sum of the respective characteristics. A supported characteristic count as 1, a partially supported characteristic as 0.5 and an unsupported characteristic as 0.

Creating a pattern that meets all proposed characteristics remains a challenge. All proposed patterns are monotonic by design, and among the patterns proposed, the Arrows pattern excels in encoding characteristics. Most patterns support the $\nabla_{RatioLat}$ characteristic, but in the case of Noise, it is not trivial to design a function that matches the color ratio to the progress value. The $\nabla_{RatioTop}$ characteristic is supported by fewer patterns, with the Pyramid pattern only partially providing this characteristic as the ratio of colors is skewed to the current progress value. The Pyramid pattern falls short in the independence from height changes ($\nabla_{IndHeight}$) as the color border depends on the cuboid's height, causing non-linear movement during simultaneous color and height animation. In terms of large treemaps ($\nabla_{TreeSize}$), the Pyramid pattern performs well, while other patterns, such as the Squares pattern, might struggle in situations with limited screen space. The support for non-rectangular treemaps ($\nabla_{Polygonal}$) varies across the proposed patterns.

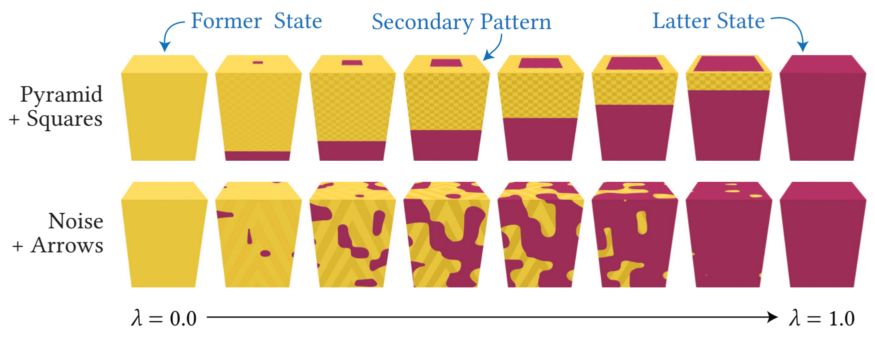


Figure 3.25: Combination of two procedural patterns. A secondary pattern supports the primary pattern placed only on the parts representing either the former (as can be seen here) or the upcoming value [L21].

3.5.4 Pattern Composition

Invariant to all patterns is that the direction of state and change value can be encoded simultaneously. We propose superimposing two different patterns, using a *primary pattern* as before but a *secondary pattern* as an additional indicator for the direction of change. This allows for a direct, unambiguous display of change direction when paused and animated, thus supporting $\nabla_{DirStates}$ and $\nabla_{DirChange}$.

The choice of the primary pattern does not limit the secondary pattern. We suggest using a pattern distinct to the primary one, ideally with a uniform distribution of colors. This ensures that the pattern is distinguishable from the main pattern and visible without regard to the current transition state. Figure 3.25 shows examples using the Pyramid and Noise patterns with Squares and Arrows as secondary patterns, respectively.

The secondary pattern, derived for a static progress value $\lambda = 1/2$, is applied using a blend of the base color and a darker shade. Occupying half the applied surface area, we propose using the surface encoding the former color for superimposition. As a result, this pattern is exposed (decreasingly) only towards the end of the transition since the former color's surface will vanish. Additionally, it should gradually fade in, not being visible at the transition's beginning, to avoid an abrupt appearance at the start.

3.5.5 Animation Control

We can use animation to display the continuous progression over multiple dataset states and transitions, showcasing changes throughout an entire treemap and across multiple data snapshots [L22]. Animation is closely linked to time and duration, from running all transitions simultaneously to executing them sequentially. To enhance usability, we implement explicit management of individual transitions, grouping similar changes, and applying constraints such as separating animations based on visual variables, ordering value decreases before increases, and grouping animations by proximity in the treemap.

A virtual, global time advances as the configured animation progresses from one dataset state to another. Per-node transition control values are updated for each time step, and the treemap is redrawn. The process concludes when all transitions are complete. Our prototype allows for independent, simultaneous transitions per node for every mapped data

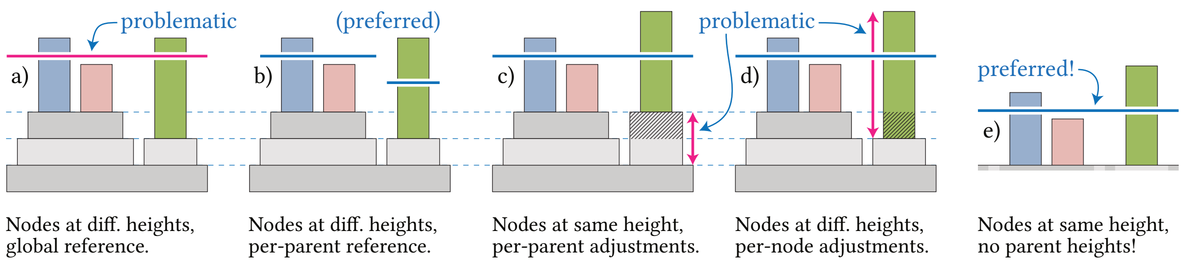

problematic
a)
(preferred)
b)
c)
problematic
d)
preferred!
e)
Nodes at diff. heights, global reference.
Nodes at diff. heights, per-parent reference.
Nodes at same height, per-parent adjustments.
Nodes at diff. heights, per-node adjustments.
Nodes at same height, no parent heights!

Figure 3.26: Variations of height reference for simultaneous use of cuboids for leaf and inner nodes (a to d). The magenta and blue horizontal lines suggest deceptive and genuine height readout, respectively. With the virtual camera's altitude presumed to be above 20° we favor using overlapping or nested rectangles for inner nodes (e). This reduces occlusion and visual complexity while preserving nesting clues, maintaining labeling of parents, and enabling height comparison with and without height reference [L13].

attribute, enabling the ordering and masking of transitions based on various factors such as transition type, user-defined values, and metadata.

3.6 Value-added Adaptations for 3D-Embeddings

Closing this chapter, we briefly present value-added adaptations for 3D-embeddings of treemaps. We cannot provide a comprehensive set of design guidelines for treemaps but focus on specific techniques and augmentations. Such guidelines, especially concerning perception [138] or use of color [191, 208, 279] are not specific to treemaps, but relevant for every visualization, and thus not the focus of this paper. Please refer to geographic, thematic maps, and commonly used sequential, diverging, and qualitative color schemes for more information on color. The following remarks are on the stacking of inner nodes, the resolution of attribute values and their visual representation, and the benefits of using a visual height reference.

3.6.1 Height Mapping for Inner-Nodes

We initially adopted 3D-embedded treemaps with inner node stacking [33], as we were familiar with this approach and aware of its use by others. Height mapping for inner nodes in treemaps typically involves creating platforms [4] or pyramid frustums [5], and placing child nodes on top of these structures, as depicted in most figures above so far (e.g., Figure 3.2). However, alternative approaches for nested depiction exist, such as using full spheres [16] or hemispheres [14] in combination with transparency. Transparency can be employed to reduce occlusion and encode different node states [157].

While working on height-based selection and filtering in 2018 [L13], we fundamentally questioned that practice and switched to a flattened display of inner nodes: simultaneous use of cuboids for leaf and inner nodes should be applied with care, as it can increase occlusion and visual complexity. In most application domains, tree-structured data commonly has a modest hierarchical depth. Nonetheless, an extruded display of inner nodes might suggest an intentional and meaningful data-to-height mapping, which it usually is not. Furthermore, heightened and stacked parents aggravate

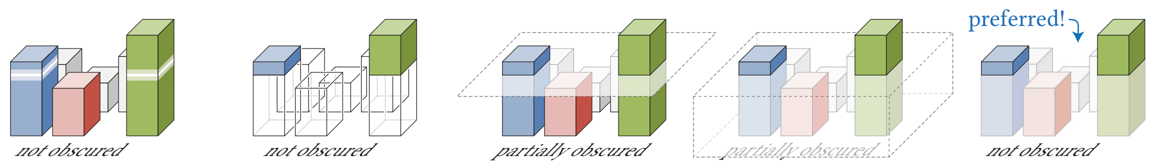


Figure 3.28: Illustrations of variations for the visual display of a height reference, f.l.t.r., emphasizing intersections, using stilts, using an explicit surface, using an enclosing surface, and using an implicit surface per cuboid [L13].

height comparison on the display of leaf nodes, even with additional tools such as a height threshold or reference. Figure 3.26 illustrates these issues. We recommend using nested and flat parent nodes for better structural comprehension, more accessible labeling, and predictability of item locations within a treemap. If not stated otherwise, the remainder of this work assumes nested and flat inner nodes, as used in Figure 3.23.

3.6.2 Height-based Filtering using Reference Surfaces

Height estimation in a 3D-embedded treemap might be hindered due to perspective foreshortening of cuboids, as well as occlusion (Figure 3.27) or height distorting parent node representations [29, 67]. We introduced a *height reference* [L13] that (1) increases the accuracy of height readings, (2) allows for fast and precise identification of similar or correlating nodes, and (3) prevents miscommunication of height, e.g., caused by overlapping or occluding cuboids. This technique is based on water-level metaphors or *movable baselines* used in 3D histograms to "cognitive aid to augment a user's ability to compare the [height] values." [281]

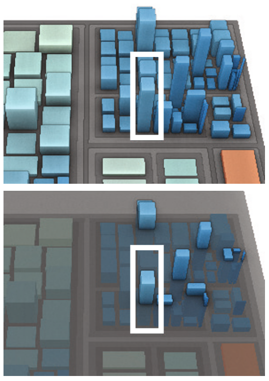


Figure 3.27: The height of the rear cuboid (framed in white) is partially obscured and can lead to incorrect reading in this particular case.

The goal is to increase the added value of height in a 3D embedding. *Height reference* denotes the depiction of a height threshold that can be interactively modified and enables height-based queries. Such details-on-demand queries involve identifying nodes of interest based on their height. This comprises the identification, filtering, selection, and comparison of nodes of specific, height-related characteristics:

Highest Nodes Identification of order of top-most nodes.

Similar Nodes Accentuation of nodes of similar or same height (peers).

Thresholded Nodes Exclusion or inclusion of nodes that fall below or are above a threshold, respectively.

We introduced an interaction mechanism that enables interactive, precise, and direct manipulation of the height reference within the treemap metaphor [L13]. A preliminary user study indicated that using the height reference increases accuracy for reading heights.

For the rendering of a height reference, multiple techniques come to mind; via intersections, an explicit surface, a closed surface, an implicit surface, stilts, and transparency bases as depicted in Figure 3.28. The most promising approaches are intersections and implicit surfaces. In the latter, all cuboids are desaturated below the height threshold on a per-fragment basis. When applied to all cuboids, the impression of a transparent, infinite

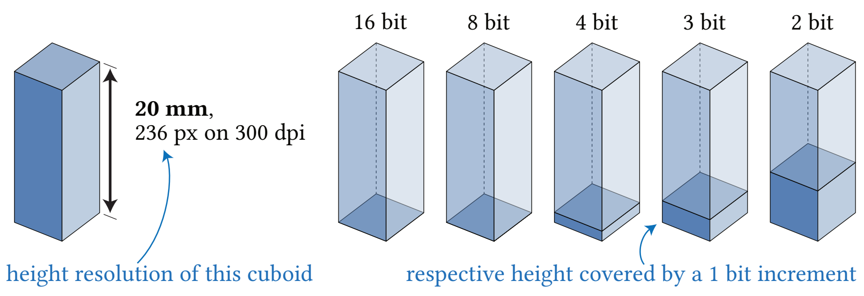

16 bit
8 bit
4 bit
3 bit
2 bit
20 mm,
236 px on 300 dpi
height resolution of this cuboid
respective height covered by a 1 bit increment

Figure 3.29: Various resolutions in respect to a cuboid with a height of 2 cm, printed with 300 dpi. It emphasizes that 3 or 4 bits for mapping data to height may already be sufficient, especially in larger treemaps.

surface at the height threshold is created. Inner nodes are not obscured and can be easily accounted for, and screen-space processing is unaffected. Instead of a treemap-global use, this technique can also be applied to an arbitrary subset of nodes. The overall visual display of the treemap's structure is fully preserved.

3.6.3 Resolution of Visual Data Encodings

Height mapping in treemaps is often applied with an excessive data resolution that surpasses the available pixel resolution. It further overlooks humans' tendency to overestimate the height of distant objects compared to objects in the foreground—we found significant differences in the estimation error for different camera angles [L4]. The previously discussed use of a height reference aims to mitigate these drawbacks.

Fortunately, the required resolution for a compelling visual display is usually much lower than the resolution of the mapped data. We recommend reducing the data to a resolution appropriate for the map theme or task. For example, attributes mapped to area, color, or height do not require a 32-bit resolution. Often, reducing attribute resolution to a few bits can improve readability through discriminability and emphasize the results of the preceding data analysis (Figure 3.29). This improvement can also be supported from a perceptual perspective: height with perspective foreshortening is challenging to compare on a per-pixel basis.

The exact size or height of a node should be of subordinate importance. Most often, transforming the data to a categorical data type, such as *irrelevant*, *low*, *medium*, and *high* (for color, height, or change), or *lower-outlier*, *below average*, *average*, *above average*, and *upper-outlier* (for area), leads to easily comparable and more effective visualizations [270]. Alternatively, when depicting change, the degree of change might be well accounted for using five or seven discrete levels of a diverging scale. For instance, five distinct colors can indicate *substantial increase*, *increase*, *stagnation*, *decrease*, and *substantial decrease*.

3.6.4 Addressing Non-Visual Challenges

When a 2D or 3D-embedded treemap is barely supportive during exploration, it can help to re-evaluate the visualization design in light of the given task. The impact on a treemap's effectiveness is often significantly influenced by the *data enhancement* stage that precedes *visualization mapping*. The problem or task data must always be carefully prepared for visualization. This includes resampling, normalization, outlier filtering, and accumulating weighted leaf-node data to inner nodes.

Here are some assumptions on data enhancement and treemap configuration that can help take advantage of the visual variables of $\mathcal{A}^3$.

- Ensure height mapping correlates larger heights with higher relevance for specific tasks. For example, consider an inverse height mapping when small or negative values are task-relevant.
- Handle data outliers appropriately, e.g., by removing them, using non-linear mapping, introducing additional visual variables, or employing multi-range height mapping, such as stacked percentiles (preferred).
- Maintain consistent cuboid heights for respective interactions by adjusting the height of cuboids when the data or attribute mapped changes. Specifically, we advise against height-based filtering and selection by modifying the height mapping (e.g., clamping and normalizing the mapped data on the fly).
- Opt for a smaller height scale to reduce occlusion and promote preattentive color processing. Allow color, weight, structure, and labeling to primarily support any given task, while height and other visual variables preferably take subordinate roles for details on demand and other subsequent explorative queries.
- Use discretized height mapping if exact height is unimportant, as this can improve height comparison and identification.
- When fine-grained differences in height mapping are of interest, enable users to spot differences accurately, for example, by making the mapping of differences more explicit and emphasized.

These assumptions can aid in designing more expressive and compelling 3D-embedded treemaps that harness the power of visual variables in $\mathcal{A}^3$, facilitating a more insightful data exploration experience.

4 Level-of-Detail and Labeling for 3D-Embedded Treemaps

The contents of this chapter are based on the following original publications:

D. Limberger, W. Scheibel, S. Hahn, and J. Döllner. "Reducing Visual Complexity in Software Maps using Importance-based Aggregation of Nodes". In: *Proc. SciTePress IVAPP.* 2017 [L8]

D. Limberger, W. Scheibel, M. Trapp, and J. Döllner. "Mixed-Projection Treemaps: A Novel Approach Mixing 2D and 2.5D Treemaps". In: *Proc. IEEE IV.* 2017 [L9]

D. Limberger. "Interactive, Adaptive Level-of-Detail in 2.5D Treemaps". U.S. pat. 9953443. Seerene GmbH. 2018 [L10]

D. Limberger, A. Gropler, S. Buschmann, J. Döllner, and B. Wasty. "OpenLL: an API for Dynamic 2D and 3D Labeling". In: *Proc. IEEE IV.* 2018 [L11]

D. Limberger, M. Trapp, and J. Döllner. "Depicting Uncertainty in 2.5D Treemaps". In: *Proc. ACM VINCI.* 2020 [L18]

As the amount of hierarchical data increases, 3D-embedded treemaps often become cluttered making it difficult to interpret the visual representation. "Clutter is the state in which excess items, or their representation or organization, lead to a degradation of performance at some task." [199] In this chapter, we explore and demonstrate techniques that improve the readability and effectiveness of 3D-embedded treemaps. We begin by investigating the dynamic aggregation of nodes, which allows for the adaptation of graphical elements based on node-based scoring [L10, L8]. This method enhances the readability of 3D-embedded treemaps while managing visual complexity. Next, we discuss the visual display of aggregates, offering insight into data summary and emphasis techniques that further improve treemap readability [L8, L18]. Moving on, we explore dynamic labeling, addressing the challenges associated with labeling information-abundant displays and providing high-quality rendering of text in 3D [L11]. Lastly, we outline partial 3D-embedding using mixed projections [L9] and discuss their compliance with the visual information seeking mantra.

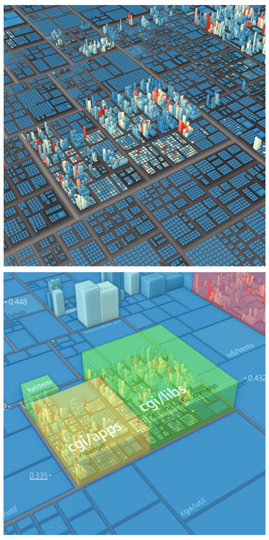


Figure 4.1: 3D-embedded treemap (top) decluttered using Aggregation and Labeling (bottom).

With these techniques at our disposal, we strengthen the potential of 3D-embedded treemaps and support our second hypothesis $\mathcal{H}_S$: reducing visual clutter and complexity while facilitating the interactive exploration of extensive data through aggregation and labeling (Figure 4.1).

4.1 Dynamic Aggregation of Nodes

Visualizing large hierarchical data by 3D-embedded treemaps can result in visual clutter, which is more pronounced than in 2D treemaps [72]. The 1:1 mapping of nodes to cuboids contributes to this clutter [199], increasing visual complexity and cognitive load—the effort used in working memory to accomplish a given task [103]. In extreme cases, multiple nodes may be depicted within sub-pixel space, distorting their visual display and hindering meaningful interpretation (Figure 4.2). Treemaps are particularly suited for handling large data quantities in the first place, easily outperforming many other visualization techniques, which may struggle with just hundreds of elements.

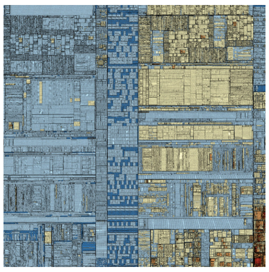


Figure 4.2: Treemap seen from the top, depicting tens of thousands of nodes, most cluttered within sub-pixel space.

To address this, we will use *appearance distortion* [65], which involves abstraction through aggregation by strategies that change the gestalt of node representation. We present a dynamic *level-of-detail* (LoD) technique that uses per-node scoring for aggregation. Aggregation thereby describes the process of combining multiple graphical elements into a single, simplified representation to reduce complexity and improve readability. Our technique (1) adheres to established aggregation guidelines, (2) allows for multi-resolution—referring to varying levels of detail depending on the user's focus—depictions of multivariate data, and (3) facilitates annotation and efficient identification of nodes of interest. Our scoring approximates the importance of nodes by considering *degree-of-interest* (DoI) measures, including screen size and user interaction.

Aggregation creates a topological overview, assisting users in recognizing patterns and correlations [171]. It was used in many forms, e.g., as data-dependent LoD for pattern detection in time series [102], as progressive refinement strategy in treemaps [198], and for subdivision of shapes [66]. The two latter approaches only focus on the scarcity of rendering resources and limited screen size. The treemap topology can serve as the basis for zooming during identifying and exploring areas of interest through user navigation [31, 155]. We support this implicitly through navigation (zoom) and explicitly through user interaction (fold, unfold), both through scores. Our approach reduces the need for user navigation in the first place and relies on automated DoI approximation instead.

4.1.1 Degree-of-Interest Scoring of Nodes

In our approach, we built on a "*Degree of Interest* [DoI] function, which assigns to each point in the [tree-]structure, a number telling how interested the user is in seeing that point, given the current task." [80] A node's DoI is determined by weighted scoring of attribute values, visual appearance, and user interaction. Our adaptive LoD uses minimal aggregation of interesting areas while maintaining valuable context information summarized, i.e., aggregated, at lower resolutions. This can reduce the need for user navigation by guiding the user to relevant data and decreasing the number of irrelevant, unnecessary navigation steps.

We ensure that apparent data features do not disguise less prominent but significant details conveyed by nodes with high DoI (*nodes of interest*). Our multi-resolution scoring allows for aggregation control on a global and local per-node basis. Globally, nodes of certain hierarchy levels may not be aggregated, such as the first two, as these capture the foundational, usually very stable, characteristic structure of the hierarchy represented. Locally, the virtual camera's viewport and distance to graphical elements are used to approximate screen size. Information density can be changed through user interaction or *focus+context* concepts such as lenses [240].

A node n is scored by score functions s_c that map to the closed interval $[-1, +1]$, striving either for or against aggregation with $s_c > 0$ or $s_c < 0$, respectively. $c \in C$ denotes one of various DoI criteria. The total score of a node $\bar{y}$ is accumulated using a weighted mean of the set of scores:

$$\bar{y} = \frac{\sum_c \omega_c s_c}{\sum_c \omega_c}, \tag{4.2}$$

with ω_c allowing for non-negative, use-case specific emphasis of scores.

4.1.2 Score Propagation and Processing

The adaptive LoD process by means of scoring and aggregating is illustrated in the following algorithm (algorithm 4.1):

```
Function process(tree, map theme)
    nodes ← nodes of tree
    attributes ← attributes in map theme
    foreach attribute in attributes do                    // attribute accumulation
        accumulate(post_order(nodes), attribute, aggregation operator)

    functions ← score functions in map theme
    foreach function in functions do                      // calc scores for criteria $s_c$
        score(post_order(nodes), function)
    foreach node in nodes do                              // derive each node's score $\bar{y}$
        scores ← list of the node's scores
        node.score ← weighted_mean(scores)

    // render tree, i.e., trigger recursive draw on root
    root ← root node of tree
    threshold ← aggr. threshold of map theme
    draw(root, threshold)

Function draw(node, threshold)
    if node is leaf then                                  // render node as leaf
        render_leaf_cuboid(node)
        return
    if node.score ≥ threshold then                        // render node as aggregate
        render_aggregate_cuboid(node)
        return

    render_inner_cuboid(node)
    foreach child in node's children do
        draw(child, threshold)
```

Algorithm 4.1: Scoring and aggregating of nodes [L8].

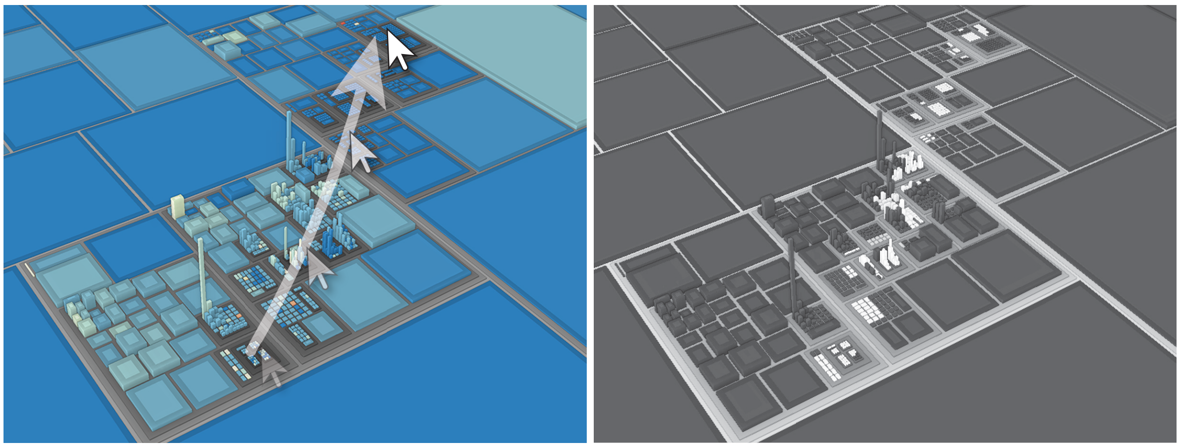


Figure 4.3: An example of a treemap using a *mouse-over score* s_{mov} for dynamic aggregation control. Thereby, a node's score 'heats up' when the cursor passes nearby, and slowly 'cools down' afterward. This is illustrated in the figure on the right, where the score is assigned to a color, from low to high, shown in gray and white, respectively. Alternatively, a *gaze score* s_{eye} based on gaze tracking could be used similarly.

Given a tree-structured dataset, we accumulate attributes bottom-to-top, score nodes, derive total scores, and finally render and aggregate the nodes top-to-bottom. The functions `accumulate` and `score` represent fold operations, which recursively aggregate attribute values and scores, respectively. They use an *aggregation operator* (cf. subsection 4.2.1) that, for example, computes an average, some deviation, or a summed total. Scoring and score weights ω_c used in `weighted_mean` can be tailored to specific use-case strategies, accounting for the given task and domain. Finally, each inner node is drawn with or without its children using `render_inner_cuboid` or `render_aggregate_cuboid` respectively. A node drawn without children is referred to as an *aggregate*.

Interaction-based Scoring. Interaction-based scoring [L8] complements user navigation with direct node interaction capabilities and supports the VISM. A quasi-binary *fold score* s_{agg} enables direct user control over the aggregation state of inner nodes through toggling. Initialized with 0, it toggles to either $+1$ or -1. Per-node interactions often occur alternating with user navigation and must be processed immediately. Additional focus-of-attention measures can include the camera's look-at direction (*spot score* s_{cam}), cursor position (*mouse-over score* s_{mov}), and gaze data (*gaze score* s_{eye}) using eye-tracking (Figure 4.3).

View-based Scoring. View-based scoring [L8] forms the basis of most LoD techniques and scores the visibility of graphical elements on the screen. A *screen-space area score* s_{ssa} approximates the number of pixels eventually used to visualize a single cuboid. Occlusion queries—mechanism to query "the number of samples that pass the depth and stencil tests"[4.1] for primitives—or the axis-aligned bounding rectangle of a cuboid projected

[4.1]Ross Cunniff, Matt Craighead, Daniel Ginsburg, Kevin Lefebvre, Bill Licea-Kane, and Nick Triantos. *OpenGL Extension: ARB_occlusion_query*. registry.khronos.org/OpenGL/extensions/ARB/ARB_occlusion_query.txt. 2007.

into screen space can be used to approximate this score. If there is insufficient screen space for the visual display of its children, or if it resides in sub-pixel space itself, then it scores towards aggregation. A screen-space threshold can invoke aggregation much earlier, not just avoiding sub-pixel junk but accounting for screen size and pixel density.

As view-based scoring depends on the virtual camera's configuration and state, it is implicitly controlled by user navigation. Therefore, it needs to be updated immediately after a user has completed a navigation operation and presumably continues exploring the data.

Attribute-based Scoring. Attribute-based scoring [L8] provides principal scores for automated DoI approximation and appropriate initial aggregation. The *variance score* s_{var} calculates an attribute's variance or value range, indicating an inhomogeneous distribution of attribute values among child nodes. This score is used to argue against aggregation and encourage additional exploration. A *child count score* s_{cc} quantifies the number of immediate children or the absolute number of all contained leaf nodes. This score lessens the chance of aggregating structurally complex nodes, which may not necessarily correlate with the nodes' footprint. Additionally, a node's isolation with respect to its surrounding neighborhood can be measured using a *local outlier factor* [36]. All these scores need to be processed only if the data itself or the mapping of it changes.

Some scores are less stable and may be distractive due to frequent incomprehensible aggregation state changes. Continuous scoring and subsequent aggregation state changes during ongoing navigation and interaction might cause distraction. Hysteresis control, similar to temperature-controlled devices, could be applied by introducing a switch cool-down period or adjusting the score's weight. The key factor is enabling users to intuitively comprehend and, when possible, anticipate the aggregation behavior, which allows for optimal utilization during interactive exploration.

4.2 Visual Display of Aggregates

For the aggregation of 2D and 3D graph clusters, the clusters' bounding box can be used for the aggregates' shape [15]. The footprint of all underlying nodes, including padding, is geometrically aggregated by the aggregate's bounding box. This aggregation strategy is common in tree visualizations such as icicle plots [130] and polar treemaps [52].

With our approach, we replace the associated inner cuboid and all subjacent leaf and inner cuboids with a single *aggregate* cuboid. The inner node becomes a leaf node and could be represented as such. The aggregate's visual variables are used to appropriately summarize the mapped attribute values of underlying nodes (Figure 4.4). This process results in information loss and increases the uncertainty of the treemap by (1) obfuscating whether or not to explore and investigate further and (2) hindering the identification of relevant nodes, outliers, or patterns.

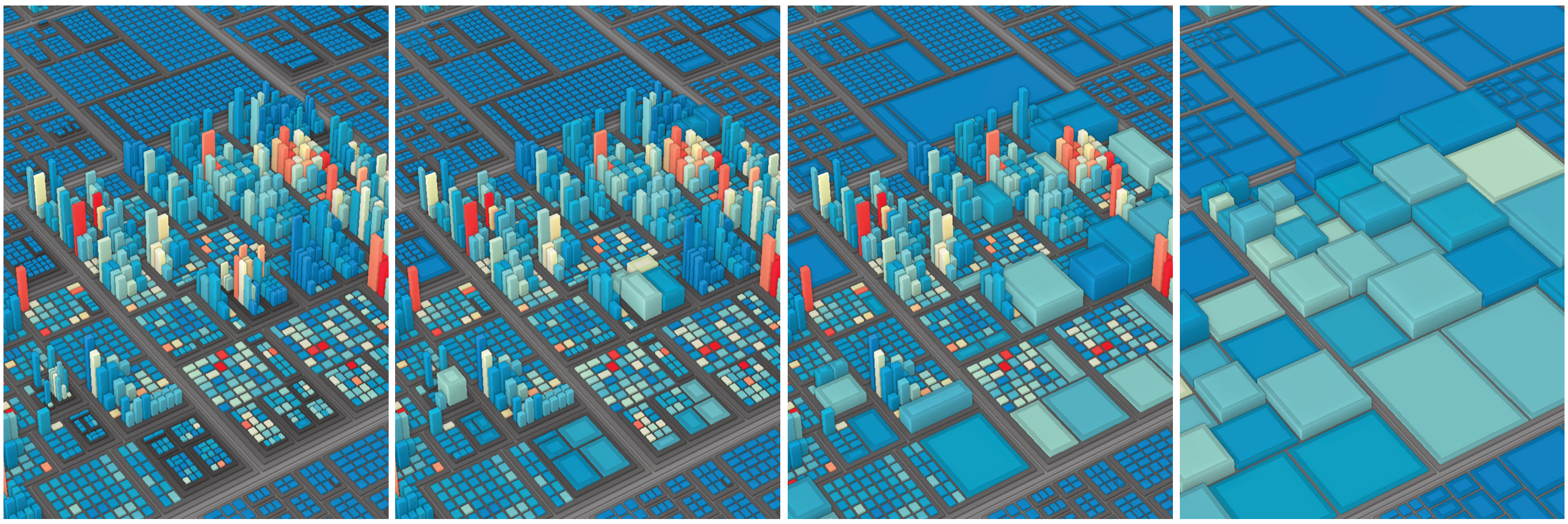


Figure 4.4: Stepwise simplification of a treemap using aggregation, ranging from little aggregation (left) to strong aggregation (right). The LoD is controlled using DoI scores and, as shown here, is based on the tree level.

Elmqvist and Fekete [66] introduced guidelines for hierarchical aggregation in 2010. These guidelines describe the characteristics of the resulting display of aggregates from an observer's perspective. Though not specifically designed for 3D-embedded treemaps, we apply all of their following guidelines to the visual display and use of aggregates:

G1 Entity Budget states that a maximum of displayed entities should be maintained.

G2 Visual Summary advises aggregates to convey information about their underlying data.

G3 Visual Simplicity requires aggregates to be clean and simple in their presentation.

G4 Discriminability demands a distinguishable presentation of aggregates and data items.

G5 Fidelity indicates that abstractions and, thus, their resulting aggregates may lie about their underlying data.

G6 Interpretability suggests aggregates always remain correctly interpretable within the visual mapping.

These guidelines are not a checklist but rather require interpretation within the visual mapping and domain.

4.2.1 Aggregation Operators for Color and Height

To reduce the loss of information caused by aggregation, such as the loss of underlying data distribution and structure, we introduce additional accumulation strategies for the attribute mapping applied to aggregates. These strategies involve using extended aggregation operators that account for specific data characteristics, including but not limited to outliers, variance, weighted average, minimum, and maximum.

For a given inner node i, a fold $\Phi(i, v, o)$ can be applied, which traverses the recursive structure of i and builds up a single, aggregated attribute value using an aggregation operation o on the map theme's attribute v. Using an arithmetic mean operator for aggregation is denoted as $\Phi(i, v, \bar{n})$, with $\bar{n}$ as an operator that calculates the mean within a set of attribute values. This operator might be insufficient for a given task, as interesting attributes might cancel each other out or remain unnoticed due to their marginal share (e.g., due to a high number of children). Instead, attribute aggregation operators might favor attributes that deviate from the mean. Operators such as $\Phi(i, v, \bar{n}_A)$ and $\Phi(i, v, \sigma^e)$ can be used for this purpose.

The $\bar{n}_A$ operator derives the weighted average. Each attribute value is weighted by the attribute associated with the area of the node. For example, the node's spatial arrangement can be considered to emphasize nodes w.r.t. their area. For large treemaps, we have observed that outliers of height or color are often found in nodes with medium to small footprints. The σ^e operator is an operator that weights each attribute according to its deviation from the mean: $|a - \bar{n}_i|^e$, where a is an attribute value and e is an exponential deviation amplifier. Figure 4.5 illustrates the effect of selected operators on an aggregate's color.

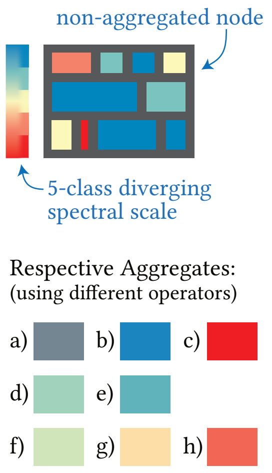


Figure 4.5: Different operators applied to an aggregates color mapping: a) visual average (out of color scale), b) minimum, c) maximum, d) $\bar{n}$, e) $\bar{n}_A$, f) σ^1, g) σ^2, and h) σ^8.

4.2.2 Nesting Level Contouring

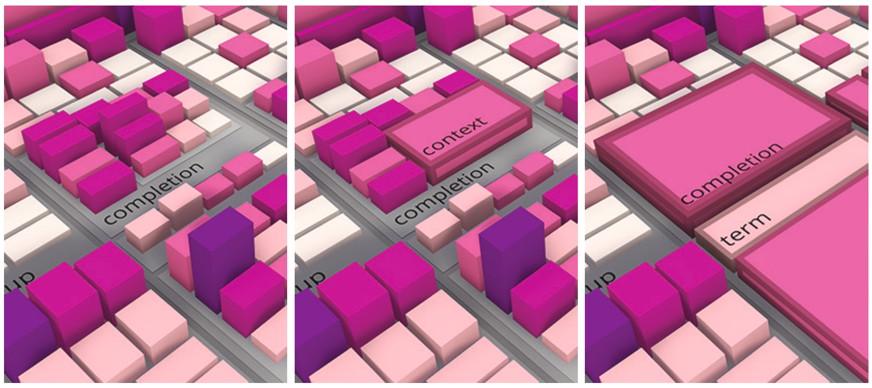


Figure 4.6: f.l.t.r.: step-wise aggregation of nodes. Nesting level contouring on the aggregates makes them discernable from leaf nodes and indicates the number of underlying hierarchy levels [L8].

Aggregates without further specialization can be indiscernible from the display of leaves. One way to address this is by truncating the aggregate cuboids or adding a contour, e.g., through a luminance offset, causing a resemblance to padding. Truncation, not applicable in 2D, increases the visual complexity of the cuboids and is sometimes challenging to recognize. Contouring, on the other hand, is sufficient to satisfy the discriminability of aggregates to non-aggregates.

Contouring can also convey additional information about the underlying structure. For example, multiple contours can hint at the degree of aggregation, indicating the number of an aggregate's subjacent hierarchy levels (nesting level). We refer to this nesting-level-based contouring as *NL Contouring* and use consecutive luminance offsets for the nested contours (Figure 4.6). In order to avoid clutter caused by too many nested contours, a threshold can be applied and combined with a gradual reduction of the luminance offset towards the innermost contour.

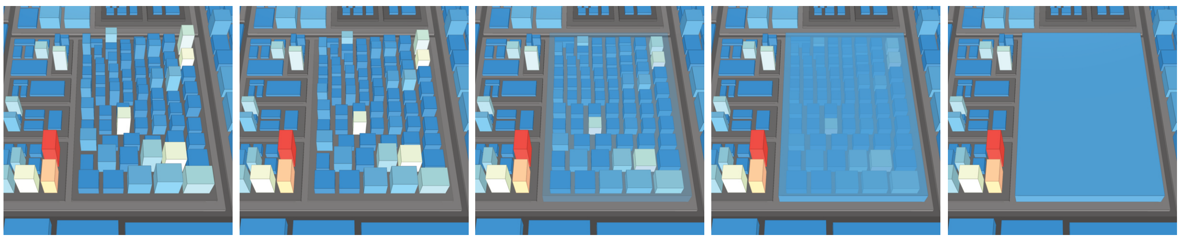


Figure 4.7: Illustration of an animated transition from the non-aggregated (left) to the aggregated state (right) of an inner-node. Over time, the height attributes are smoothly aligned to the aggregates height and the individual color attributes are interpolated to the aggregates color attribute.

4.2.3 **Animated State Transitions for Aggregates**

To enable comprehensible transitions between various LoDs, we support animated transitions that can smoothly transform a node state from non-aggregated to aggregated and vice versa. Given the preattentive nature of motion, aggregation state changes must be applied with care, either instantaneously or through temporary transitions. A node-local transition control, linked to the cause of aggregation, enables transitions ranging from explicit, noticeable, fast to less distractive, unobtrusive, slow and can be configured per treemap.

When directly caused by user interaction, fast or even instant transitions are preferred to not hinder the user in their task. Transitions caused by data changes or view-based scoring, on the other hand, can be slower and more unobtrusive. When a node transitions into an aggregate, the aggregate fades out, revealing the inner cuboid and all its child nodes with their height limited to the aggregate's height. Fading out can be done using techniques such as dithering or transparency. After an aggregate has dissolved, its child nodes, previously limited in their height, grow to their actual height (Figure 4.7). This approach is particularly helpful when changing the aggregation threshold or enabling view-based scoring.

4.2.4 **Color Weaving**

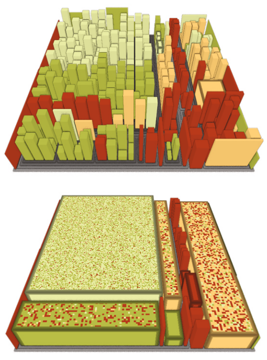


Figure 4.8: A small treemap (top) using color weaving on top of aggregates (bottom) to indicate the ratio of color values mapped to its children.

Aggregates introduce uncertainty, e.g., with respect to the degree of abstraction, deviation, or summarization of the underlying, accumulated data. To decrease the uncertainty inherent to aggregates we can use *color weaving* [22, 96, 251] (Figure 4.8). This encodes multiple colors within the same surface or space and effectively reflects a ratio of colors [96]. Additionally, small charts labeled onto the aggregates' top faces can be displayed. We focus on the former and adapt color weaving for use in 3D-embedded treemaps. The technique is applicable to leaf nodes for data item uncertainty and to aggregates, addressing data distribution or deviation uncertainty.

For visual encoding of data distribution, three techniques can be considered: (1) noise with spatial frequency, regularity, and contrast [55, 112], (2) color weaving, and (3) chart glyphs/icons [34, 220]. The first, although easy to implement, challenges users to visually decode data distribution or the existence of outliers based on signal frequencies, octaves, and intensities.

Especially for small aggregates or for maintaining a clean overall appearance, small glyphs depicting charts (*chart glyphs*) can be used (Figure 4.9). These include a stacked chart, a pie chart, a bar chart, and a 'color weaving chart', and can be labeled on top of aggregates or displayed as a prefix of an inner node's label (regardless of its aggregation state).

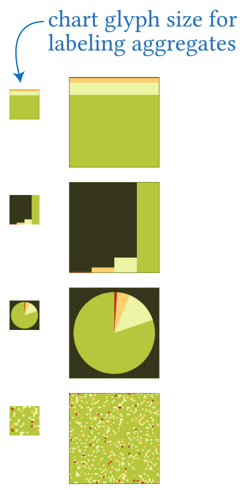


Figure 4.9: Example of four different *chart glyphs*, intended as small dynamic icons, e.g., prepended to the labeling of aggregates.

With the help of color weaving, we can explicitly depict uncertainty and mitigate it in our visualization. This allows for guiding and aiding users during interactive exploration, e.g., whether to investigate specific nodes, identify relevant nodes, detect outliers, or prevent data from being accumulated in misleading ways. Color weaving can also reflect the underlying data concerning other attributes, resulting in unweighted and weighted depictions. This approach might be suitable for the partial aggregation of modules with a large number of immediate children.

Color weaving can be combined with NL Contouring without limitations [L18] and supports not only *interpretability* (G6) but also improves *fidelity* (G5, aggregates may lie about their underlying data). Furthermore, aggregates' discriminability (G4) is increased, and there is no interference with preexisting shading, contouring, shadowing, or highlighting.

4.2.5 Evaluation & Discussion

"As more items are added to a display, populating a greater volume of feature space, there is less room in feature space to add new salient items." [199] Consequently, we create more space for salient items when de-populating the feature space, i.e., using our dynamic LoD. For evaluation, we focused on visual clutter using feature congestion and performed a user study on visual search and the impact of NL Contouring. We also review the aggregates and related techniques concerning the six aggregation guidelines.

Evaluation of Visual Search. We performed a preliminary user study to investigate the impact of aggregation on visual search. Participants were asked to find nodes of interest in different treemaps with respect to height and color mapping. The study used six pairs of images containing non-aggregated and aggregated views with automated attribute-based scoring, different camera perspectives, and datasets. We measured the *task-completion time* and *error rate* for each participant and each task (cf. [L8] for a detailed study design).

Participants recognized the use of the LoD technique and rated it as significantly improving the visual search task. However, they also noted that the visual separation of aggregated and non-aggregated nodes needed improvement, which we addressed for the follow-up study. Both error rates and task completion time were significantly reduced using our technique. The average task completion time was reduced by about 20% but failed to show a significant effect.

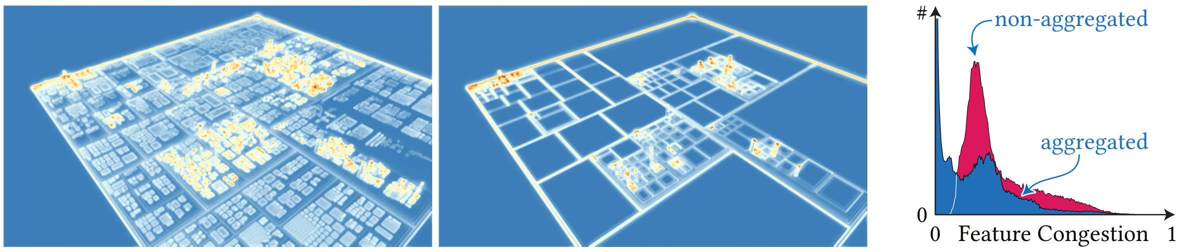


Figure 4.10: Color-mapped feature congestions (from low, 0 or blue, to high, 1 or red) for a treemap are shown as a measure of visual clutter. The histograms indicate reduced visual clutter for the aggregated variant (right).

Evaluation of Visual Clutter. To confirm the actual reduction of visual clutter by using aggregates, we compute feature congestion [199]. It measures difficulty searching through a complex display and can be used as a usability indicator of importance-based aggregation. Feature congestion was computed for multiple image pairs, each depicting the same treemap, non-aggregated and aggregated (Figure 4.10). Our LoD technique resulted in an average reduction of about 50% across these image pairs.

Evaluation of Nesting Level Contouring. We performed a user study to investigate the readability of NL Contouring. The study aimed to determine if users can correctly identify aggregated datasets using NL Contours. In a questionnaire, an aggregated version of a treemap was shown alongside four valid or invalid depictions (multiple correct answers possible). Most of the 720 answers given by the 12 participants were correct (cf. [L8]), indicating that using NL Contours can be effective.

G1 Entity budget is always impacted by viewport size. Furthermore, view-based scoring using a minimal screen-space area threshold may limit the number of cuboids. The budget can also be restricted to certain hierarchy levels. Assuming persistent node traversal, the score for aggregation is applied when the budget is exceeded.

G2 Aggregates with same or similar attribute mapping, depending on the used aggregation operators, ensure to visually summarize the underlying data. NL Contouring further allows for assumptions about the node's nesting depth. However, the aggregate does not capture the underlying data structure regarding data localization, number of nodes, value, or distribution patterns of the mapped attributes.

G3 For visual simplicity, we rely on the cuboid for aggregation, which is a simple shape. The visual appearance can be augmented using NL Contouring, glyphs, and labels.

G4 Contouring and labeling ensure the discriminability of aggregates over leaf nodes and non-aggregated inner nodes.

G5 Concerning fidelity, the visually conveyed information of aggregates can be configured according to given tasks or importance measures using different aggregation operators.

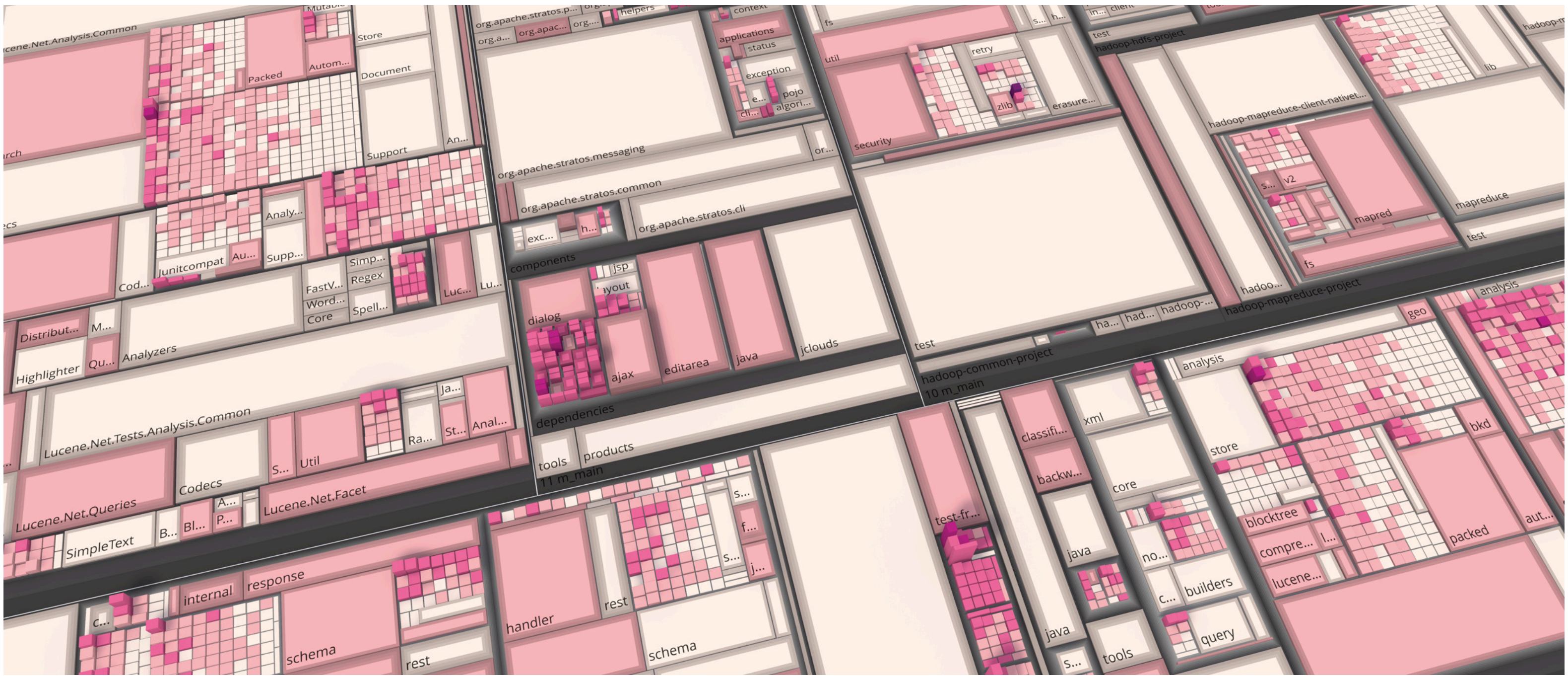


Figure 4.11: Example of a large treemap depicting source files of a software project, using aggregates and labels [L8].

G6 The aggregation relies on the existing mechanisms of a 3D-embedded treemap. Except for the NL Contouring, users are not confronted with any inconsistencies and, as our evaluations indicate, have no problems interpreting the data. However, the choice of aggregation operators might impact the user's performance and should be communicated (e.g., through a legend or map theme).

The presented aggregation technique is capable of "reducing a large dataset into one of moderate size while maintaining dominant characteristics of the original dataset" [58] while satisfying guidelines for aggregation. It (1) requires no layout re-computation, (2) allows for (mostly) unambiguous and self-consistent aggregates, (3) implements well-known interaction concepts, and (4) allows for additional annotation. It enables multi-resolution depictions of complex information, facilitates efficient identification of important nodes, and supports the VISM.

There are limitations; most notably for cases where there are few inner nodes with a massive number of children, aggregation and the information it visually comprises become useless. Some form of partial aggregation may be a solution to address this issue. Nevertheless, there are also some remaining prospects. For example, we would like to explore how aggregation can be used in communication and *locus of attention* (focal point of a user's cognitive focus), guiding, especially when exploring collaboratively.

4.3 Dynamic Labeling in 3D-Embedded Treemaps

Fekete and Plaisant noted in 1999 that "[a] major limiting factor to the widespread use of information visualization is the difficulty of labeling information abundant displays." [71] Since then, visualization has become as widespread as ever. Labeling in 3D visualizations, though, is still a limiting factor today, and rendering APIs and visualization frameworks still lack label rendering and placement capabilities.

There are several reasons why labeling has remained a relatively unexplored area in real-time computer graphics. High-quality text rendering and placement in 3D are complex tasks that often require considerable resources for prototype development. Furthermore, modern rendering APIs, such as OpenGL, WebGL, and Vulkan, do not provide built-in support for text rendering. Industries such as the games industry prioritize artistic orchestration of labeling and its design. Dynamic and flexible, labeling is usually only required to a small extent. This does not exactly create demand for standardized labeling technology.

In the following, we will briefly touch the topic labeling and text rendering, showcase descriptive label placement using OpenLL, and discuss the implementation of 3D-embedded map legends.

4.3.1 Labeling and Text Rendering

Most 3D-based visualizations today use 2D labels anchored to 3D projected screen positions [L1]. This usually works well but does not account for per-pixel occlusion with the visualization's graphical elements, making the labels appear detached from the virtual 3D scene. If text is rendered in 3D, it is often limited to a few instances, usually neither dynamic nor interactive, and often lacks a qualitative visual display. Moreover, there is no text rendering support in any of the major graphics APIs such as *Open Graphics Library*[4.2] (OpenGL), *Web Graphics Library*[4.3] (WebGL), or *Vulkan Graphics and Compute API*[4.4] (Vulkan).

We developed a glyph-based typesetting and rendering engine [R1, R4] in combination with a font asset generator supporting web-based, hardware-accelerated high-quality labeling. *Signed distance fields* (SDFs) are used for glyph shapes to accommodate varying display resolutions, text sizes, and orientations [79]. Per-pixel screen-space derivatives are employed for glyph edge anti-aliasing [190], and multiple texture channels may be added to preserve sharp corners [88]. More complex, spline-based sampling approaches are considered to overcome bitmap and SDF-based limitations [60] fully, but as of now, "[…such a] method requires considerably more computation." [148]

Different techniques have been proposed to augment treemaps with text. One approach is to use the nodes' surfaces to integrate labels in the form of *internal annotations* [L8] (Figure 4.11). This can affect the layout computation if applied to inner nodes as well [152, 155]. Another approach

[4.2] Mark Segal and Kurt Akeley. *The OpenGL Graphics System: A Specification - Version 4.6.* khronos.org/registry/OpenGL/specs/core/46.core.pdf. 2019.

[4.3] Khronos Group. *WebGL Specification.* khronos.org/registry/webgl/[...]/1.0. 2019.

[4.4] The Khronos Vulkan Working Group. *Vulkan 1.3.243 - A Specification.* registry.khronos.org/vulkan/specs/1.3/pdf/vkspec.pdf. 2023.

is *external labeling*, which is not treemap-specific but introduces common problems like "overlaps and data occlusion" [71]. External labels can be positioned as hovered text near the node [219] or with a connected line to indicate association [29]. The label size may be used to encode importance [120].

We introduced *Open Label Library [L11, R5]* (OpenLL) as a low-level, implementation-aware specification for dynamic, hardware-accelerated rendering and adaptive text placement in virtual 3D environments [L11]. It complements rendering APIs and transmission formats, such as *GL Transmission Format*[4.5] (glTF).

In gaming production, high-end labeling is attributed to artistic efforts rather than programming and is the result of tweaking and pre-computing assets in compliance with a title's art direction [L11]. Hardware-accelerated text rendering does not rely on standard font formats but requires individual glyph sprites in texture atlases (*glyph atlas*). The creation of glyph atlases for font faces in itself is already an obstacle here; first, the font file must be parsed, and individual glyphs (characters) must be rendered into high-resolution bitmasks. These provide the basis for highly downscaled signed distance field computations, which must be packed into rectangular spaces. In addition to the glyph atlas, a *font description file*, e.g., using the quasi-standard bitmap font format[4.6] is required, accounting for kerning and other typesetting related information. These steps do not account for the required parameter tweaking w.r.t. distance field scaling, glyph resolution, atlas size and packing variations, glyph padding, and more. We created an all-in-one font-asset generation service [R6] that takes over this laborious process. While configuring the glyph atlas, a live preview of the font face in 3D is shown, and various rendering settings for quality control are exposed.

4.3.2 **Descriptive Label-Placement using OpenLL**

A descriptive labeling approach is used, differentiating between the actual text, the placed label, and proxies. Kakoulis and Tollis provide a comprehensive introduction to labeling problems identified within cartography and rendering maps [129]. In our context, labeling refers to placing—not rendering—labels into a virtual 2D or 3D environment (Figure 4.12). There are various types of labels, such as internal or embedded [159], external, boundary, and excentric [181]. Labeling techniques include positioning names on maps [115], point feature placement [51], external label management in 3D object space [233], or force-based labeling [252]. Labels can also be used as proxies for interaction, as shown by Balata et al. [11].

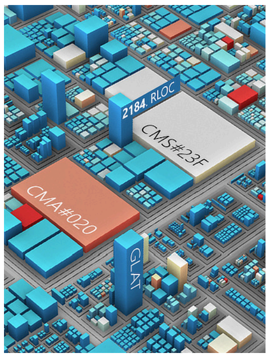


Figure 4.12: Enhancing treemaps using dynamic text placement.

Our labeling API OpenLL focuses on managing and automatically placing many labels in virtual 2D and 3D environments in real-time. It does not provide implementation specifics, e.g., for computational math, parallelization, or hardware-accelerated rendering, nor does it inflict a particular

[4.5]Khronos Group. *glTF Specification, 2.0.* github.com/KhronosGroup/glTF/[..]/specification/2.0. 2019.

[4.6]angelcode.com/products/bmfont/

implementation design. Our web-based implementation [R4] has been successfully used in various recently published visualizations [8, 139, 260].

The API introduces *font face*, *text* (a dynamic sequence of characters), and, most importantly, a *label*. The label includes configuration for typesetting, positioning, orienting, and rendering regarding a font face and text, i.e., a glyph sequence. Placement descriptions include positions, margins, anchors, proxies, constraints, or proxies such as lines and points. A *label cloud* is available for data-driven, selective labeling. Finally, a *proxy* consists of implicit shapes like spheres, cubes, quads, points, lines, or splines used to (1) partially approximate 2D and 3D scenes or, similarly, (2) describe dynamic anchors such as a virtual camera or an area of interest. By using a proxy, we can detach the scene complexity from label placement, reducing the computational complexity of label placement and configuration complexity for developers.

We can create labels that reference a proxy's vertices, edges, or areas as an anchor. A cuboid proxy, for example, could expose each of its corners, edges, center points, or lateral surfaces for anchoring. Proxies can also be used for detecting overlapping labels, accounting for proxies occluding labels, or using their extent for level-of-detail control.

A *renderer* receives labels, handles the computation of their placements, and draws them according to their description. With this approach, specialized types of labels may be created, such as *Position2DLabel*, *Position3DLabel*, *Rectangle2DLabel*, *Projected3DLabel*, *Spline3DLabel*, and *LabelCloud*. The following shows an examplary usage of three different labels, a label theme, and a single renderer (Listing 4.1):

```
const fontFace = FontFace.fromFile('opensans.regular.fnt');
const theme      = new LabelTheme();
theme.fontFace   = fontFace;
theme.fontSize   = 0.125; // world space size
theme.alignment  = 'left';
theme.lineAnchor = 'baseline';

let label0 = new Position2DLabel({
    text:       'Hello Word!',
    fontFace:    fontFace,
    fontSize:  { value: 12.0, unit: 'pt' },
    alignment:  'left',
    position:  { x: 32, y: 8, unit: 'px' },
    direction: { x: 0.0, y: 1.0} });

let label1 = new Position3DLabel();
label1.theme     = theme;
label1.alignment = 'right';
label1.position  = { x: 0.0, y: 0.0, z: 0.0}
label1.direction = { x: 0.0, y: 0.0, z: 1.0}
label1.up        = { x:-1.0, y: 0.0, z: 0.0}

let label2 = new Spline3DLabel();
label2.theme     = theme;
label2.position  = { x: 0.0, y: 0.5, z: 0.0}

let renderer = new GlyphRenderer();
renderer.superSampling = 'rgss'; // rotated grid supersampling
renderer.modelViewProjection = camera.modelViewProjection;
renderer.draw([label0, label1, label2]);
```

Listing 4.1: Initialization of different types of labels using *webgl-operate*.

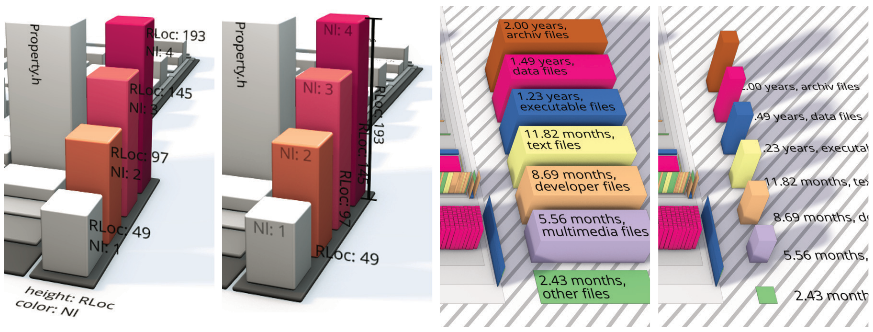


Figure 4.13: Examples of 3D-embedded map legends. They use cuboids with representative attribute values, placed right next to the treemap and labeled accordingly. They allow to communicate color and height mapping, and can indicate their value ranges.

4.3.3 3D-Embedded Map Legends

A legend explains the mapping of a visualization. For this purpose, we originally resorted to traditional means such as an overlay, a status bar, a whole side panel with detailed descriptions for novices, or simplified illustrations to explain the mapping. Eventually, we explored the extent to which map legends can be embedded in the same $\mathcal{A}^3 \oplus \mathcal{R}^2$ space used for visualization (Figure 4.13). To this end, we designed multiple layouts based on cuboids placed outside the treemap. Synthetic attribute mappings were used to create representative graphical primitives for aspects of our map theme, such as showcasing the value ranges of height and color by depicting multiple cuboids within that range. Labeling was then used to describe the attribute values or names, all within the virtual 3D scene.

We want to explore this idea further and move the map theme configuration into the 3D space for future work. By making the map legend not static but an interactive interface, exposing it to the user to manipulate the attribute mapping directly should be possible.

4.4 Partial 3D-Embedding

Partial 3D-embedding presents an alternative approach to LoD, providing on-demand detail. Similar to unfolding an aggregate, we want to expose 3D-embeddings selectively, on demand. We present a technique that tilts the inner node representations using affine transformations and animated state transitions, mixing orthogonal and perspective projections within a single treemap. By dynamically mixing different projections, we enable a seamless integration of 3D-embedded treemaps for regions of interest into 2D treemaps. This facilitates the communication of additional information by selectively tapping into a three-dimensional attribute space. Users can control the projection through manual and automated tilting, offering direct control or utilizing animated state transitions.

These interactive *mixed-projection treemaps* reduce the need for complex navigation metaphors typically associated with $\mathcal{A}^3 \oplus \mathcal{R}^2$. Translated to *focus+context* and *overview+detail* workflows, visual complexity for overview and context display is reduced while visualizing additional data in focus or detail areas. In virtual terrain and city model visualization, multi-perspective views with bent projections provide a near-field depiction of increased detail and a context area of less detail [154, 187].

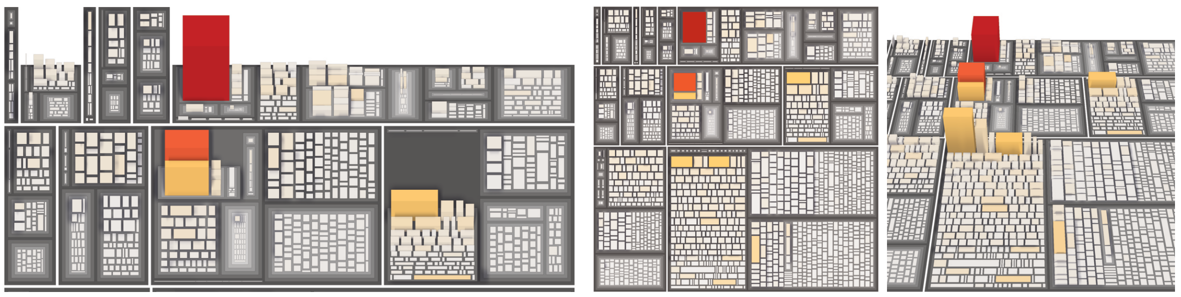


Figure 4.14: A 2D treemap (center) and a 3D-embedded treemap (right) depicting the same data using color and height. In contrast, a mixed-projection treemap (left, excerpt of the upper half) with selected nodes tilted reduces visual clutter and occlusion. It allows to toggle from $\mathcal{A}^2$ to $\mathcal{A}^3$, exposing details on demand, not accounted for in the 2D treemap [L9].

Our approach uses attributed vertex clouds [204] for interactive rendering. It uses a single geometry encoding and employs the same pipeline for both 2D and 3D-embedded nodes, thus, enabling smooth transitions through interpolation. This provides the foundation for the development of a node-local tilt operator. The height mapping and other visual variables of $\mathcal{A}^3$ are always available but may go unnoticed for non-tilted nodes due to the orthographic projection. As a result, the outcomes of screen space effects, such as shading, local ambient occlusion, shadows, and contouring, remain consistent. This can be seen, for example, in Figure 4.14, where shadows can be spotted in all three projections.

4.4.1 Node-local Tilt Operator

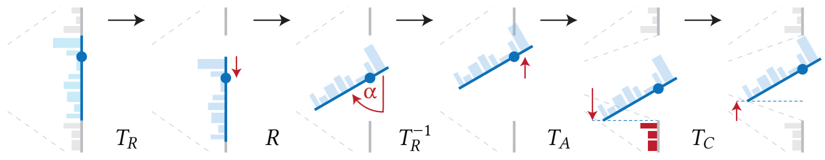


Figure 4.15: Illustration of the individual transformations of Λ for an inner-node n with perspective projection applied [L9].

The node-local tilt operator is a two-part mechanism that uses a sequence of parameterized affine transformations [L9]. Given a 2D treemap, tilt denotes the rotation of an inner node. The tilt operation combines a transformation Λ and a projection Γ. The tilt transformation shifts the node's rotation axis by T_R using a relative offset $\tau \in [-1, +1]$, with -1 shifting to the node's bottom edge and $+1$ to its top edge. Then, it rotates the node by the tilt angle α using R and anchors the node by T_A using a preferred relative location $\upsilon \in [-1, +1]$. The complete affine transformation is illustrated in Figure 4.15 and defined as follows:

$$\Lambda = [T_C]\, T_A T_R^{-1} R T_R. \tag{4.3}$$

T_C denotes an optional translation that reduces occlusion introduced when using a perspective projection. It uses the camera's eye position and, e.g., the node's vertical extent. The tilt projection Γ mixes two given projections P_0 (former) and P_1 (latter) w.r.t. the node-local tilt angle and a global angular threshold β. It is defined as:

$$\Gamma = (1 - t)\, P_0 + t P_1, \tag{4.4}$$

with $t = \alpha\beta^{-1}$, clamped to $[0, 1]$. If a perspective projection is configured for P_1, P_0 should be the respective orthographic projection, i.e., covering the same treemap region to ensure smooth interpolation.

The transformation and projection are applied per-node. They can be in different states at any time for any node. This process is restricted to global parameters for β, τ, and υ. α has to be increased for the transition, starting at $\alpha = 0$ for no transition up to the desired final tilt angle.

4.4.2 Parameterization of Node-local Tilt

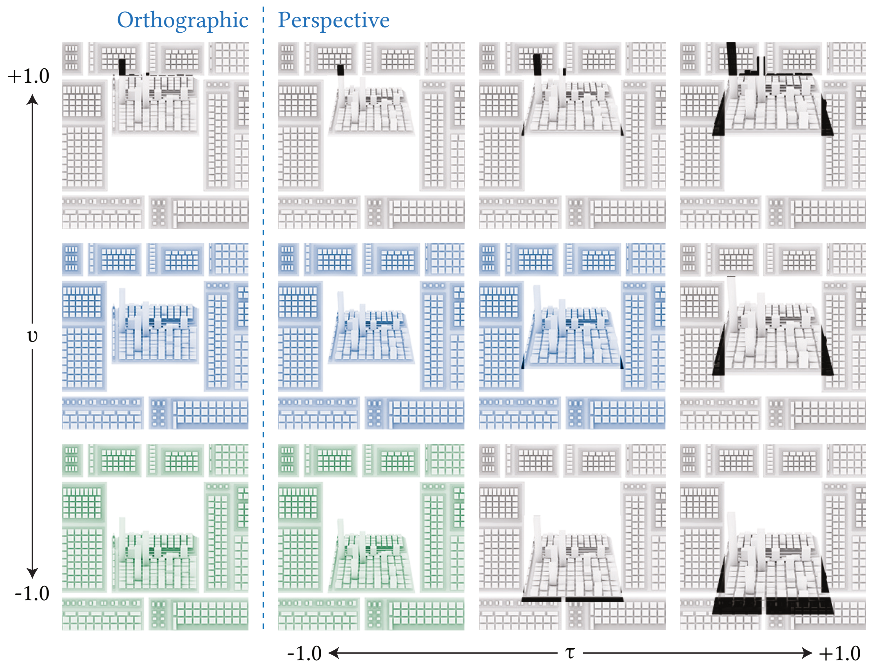


Figure 4.16: Parameter space for τ and υ, orthographic and perspective projection, with $\alpha = 60°$. Our preference for the configurations is colored in blue and green for partially valuable and valuable, respectively.

The zoom and pan metaphor is used for navigation in the 2D treemap, and users can interact directly with treemap nodes based on direct manipulation metaphors. Two tilt modes are available: *manual tilt* and *automated tilt*. Manual tilt enables the user to seamlessly increase and decrease the tilt angle of any node, while automated tilt allows the user to invoke a preset tilt angle or un-tilt any node with a single input event. For the animated transitions, arbitrary easing [119] can be applied. Additional interactions with visual variables in $\mathcal{A}^3$ can be enabled, such as height-based filtering.

While perspective projections provide additional information about depth and are often easier to interpret, orthographic projections facilitate the comparison of nodes with respect to height and area. Orthographic projection also eliminates occlusion of adjacent, non-tilted nodes in the overview. When perspective projection is preferred, we suggest to use $\tau < -0.5$ and move the tilted node 'behind' the others (Figure 4.16).

The inclination of inner nodes introduces unused screen space, which may partially be covered by high cuboids, but may also be further exploited. A lower alignment seems most beneficial, as it minimizes the risk of occluding nearby nodes and provides most additional space for information display (Figure 4.17). A tilt degree of $\alpha = 0°$ results in no additional height information and a tilt of $\alpha = 90°$ yields most occlusion, but also a node local skyline, that could be used for an overview in height distribution. Angles between 30° and 60° were found to be reasonably supportive.

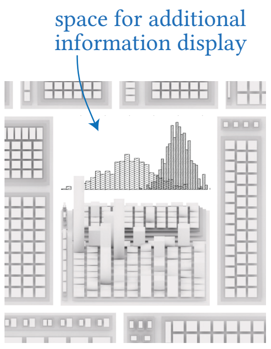


Figure 4.17: The inclination of the tilt introduces unused screen space.

A partial 3D-embedding counterbalances the limitations inherent to a full 3D-embedding, e.g., occlusion and perspective-foreshortening. This approach has been implemented and demonstrated feasible for tilting parameterization and efficient image synthesis.

Figure 4.14, for example, represents an analysis of the open-source project POCO.[4.7] The relatively small map consists of 5 775 nodes, each representing one source code file. The number of *real-lines-of-code* (RLOC) is mapped to the area, the *cyclomatic complexity* to color, and the *average nesting level* of a source code file is mapped to the height. This map theme is primarily used to detect source code files with disproportional nesting level in comparison to the implemented business logic.

The most straightforward use of a mixed projection can even be a global one; before starting on partial 3D-embeddings, we used orthographic perspective blending for our virtual camera. Whenever the camera is within a pre-defined angle to the up-vector, we start blending the camera projection into a respective orthographic one, ending with a fully orthographic projection when looking from above, resulting in a 2D treemap.

[4.7] Project available at github.com/pocoproject/poco; Analysis of a revision from 03/24/2009.

5 Web-based Provisioning of 3D-Embedded Treemaps

The contents of this chapter are based on the following original publications:

D. Limberger, B. Wasty, J. Trümper, and J. Döllner. "Interactive Software Maps for Web-based Source Code Analysis". In: *Proc. ACM Web3D*. 2013 [L1]

D. Limberger, W. Scheibel, S. Lemme, and J. Döllner. "Dynamic 2.5D Treemaps using Declarative 3D on the Web". In: *Proc. ACM Web3D*. 2016 [L5]

D. Limberger, K. Tausche, J. Linke, and J. Döllner. "Progressive Rendering using Multi-frame Sampling". In: *GPU Pro 7: Advanced Rendering Techniques* (2016) [L6]

D. Limberger and J. Döllner. "Real-time Rendering of High-quality Effects using Multi-frame Sampling". In: *ACM SIGGRAPH 2016 Posters*. ACM, 2016 [L3]

D. Limberger, K. Tausche, J. Linke, and J. Döllner. "Progressive Rendering using Multi-frame Sampling". In: *GPU Pro 360 Guide to Rendering* (2018). GPU Pro 7 Reprint [L12]

D. Limberger, M. Pursche, J. Klimke, and J. Döllner. "Progressive High-quality Rendering for Interactive Information Cartography using WebGL". in: *Proc. ACM Web3D*. 2017 [L7]

Today's interactive information cartography clients, e.g., for visualization of geo-referenced data or non-spatial software system information, are preferably accessible to users via web applications. For their provisioning, we need to consider access to and transmission of data, availability of the application, and robustness and responsiveness for interactive exploration with that data on arbitrary clients. For non-trivial visualization, we are highly reliant on hardware-accelerated low-level graphics APIs. Simultaneously, we target end-user platforms with heterogeneous hardware and software, especially mobile devices. This imposes additional challenges and burdensome constraints on their development, such as (1) having direct access to neither memory nor resource monitoring, and (2) being constrained to a tiny subset of graphics API features, WebGL more restrictive than comparable APIs for embedded systems, *Open Graphics Library for Embedded Systems*[5.1] (OpenGL ES).

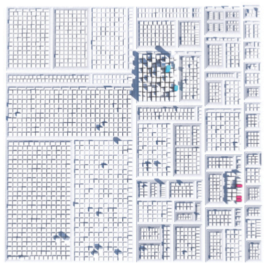


Figure 5.1: Top view on a 3D-embedded treemap created using an approach on the opposite end of image-synthesis spectrum; offline path-tracing using Autodesk's 3ds Max.

Most modern rendering techniques are designed and highly optimized for single-frame execution and are not accessible for web-based graphics (Figure 5.1). Their assumptions and requirements on GPU features and CPU resources make them incompatible with responsive rendering that works

[5.1] Khronos Group. *OpenGL ES Version 3.2*. khronos.org/registry/OpenGL/specs/es/3.2. 2019.

on many end-user devices and platforms. This was further exacerbated by a major vendor's delay in supporting WebGL2,[5.2] coupled with the deprecation of OpenGL without endorsing an open successor like Vulkan. This development strongly undermined the cross-platform compatibility of numerous research efforts over the years. However, the incremental nature of user interactions in information visualization often does not require strict continuity in rendering due to (1) the lack of dynamic objects and animations, (2) infrequent data changes, and (3) primarily discontinuous changes of the virtual camera.

To support our responsiveness statement $\mathcal{H}_R$, we present concepts and techniques that enable high-quality and responsive rendering of 3D-embedded treemaps using a web-based graphics API. We begin with a discussion on rendering quads and cuboids for treemaps. For high-quality aspects of rendering (post-processing), we present accumulative, multi-frame rendering [L3, L7, L6, L12] and describe effects that help increase the visual quality of our treemaps. Finally, we show how dynamic, interactive, web-based 3D-embedded treemaps can be created and configured using one of our treemap modules, *treemap.ts* [R12].

5.1 Rendering of Rectangles and Cuboids

For developing and testing most of the concepts and techniques presented in this thesis, we primarily used two prototypes for interactive 3D-embedded treemaps that we developed and maintained. First, we started with a desktop targeting implementation using C++ and OpenGL, which eventually became our internal research prototype called *Arboretum*[5.3] [R14]. Arboretum allows millions of nodes to be loaded and displayed within milliseconds from start to first frame (Figure 5.2). Also early on, and independent of that, we created a renderer with alternative design goals using C++ and OpenGL ES. With the advent of WebGL, we immediately ported that renderer to it and had our first web-based renderer for 3D-embedded treemaps; plainly named *treemap.ts* [L1, R12].

As a side product of these two use case agnostic visualization tools, we have continuously extracted non-treemap related aspects into open source rendering frameworks and auxiliary libraries, mainly *gloperate* [R1] and *webgl-operate* [R4]. These capture fundamentals required for fast, robust, and efficient deployment of interactive, hardware-accelerated visualizations focusing on $\mathcal{A}^3 \oplus \mathcal{R}^2$ and $\mathcal{A}^3 \oplus \mathcal{R}^3$. *webgl-operate*, for example, is the basis of other visualizations such as the *Software Forest* [8], a spatiotemporal data visualizer called *RoomCanvas* [139], and a 3D scatter plotter [260] for the interactive exploration of clusters in massive data.

In the following sections, we outline selected implementation details on the rendering of leaf and inner nodes and briefly compare the similarities

[5.2]Khronos announced pervasive support of WebGL 2—the development of which started in 2013, released in 2017—from all major web browsers on February 9, 2022, khronos.org/blog/[...]support-from-all-major-web-browsers

[5.3]The term arboretum refers to a botanical garden or park where trees, shrubs, and other woody plants are cultivated for scientific, educational, and ornamental purpose.

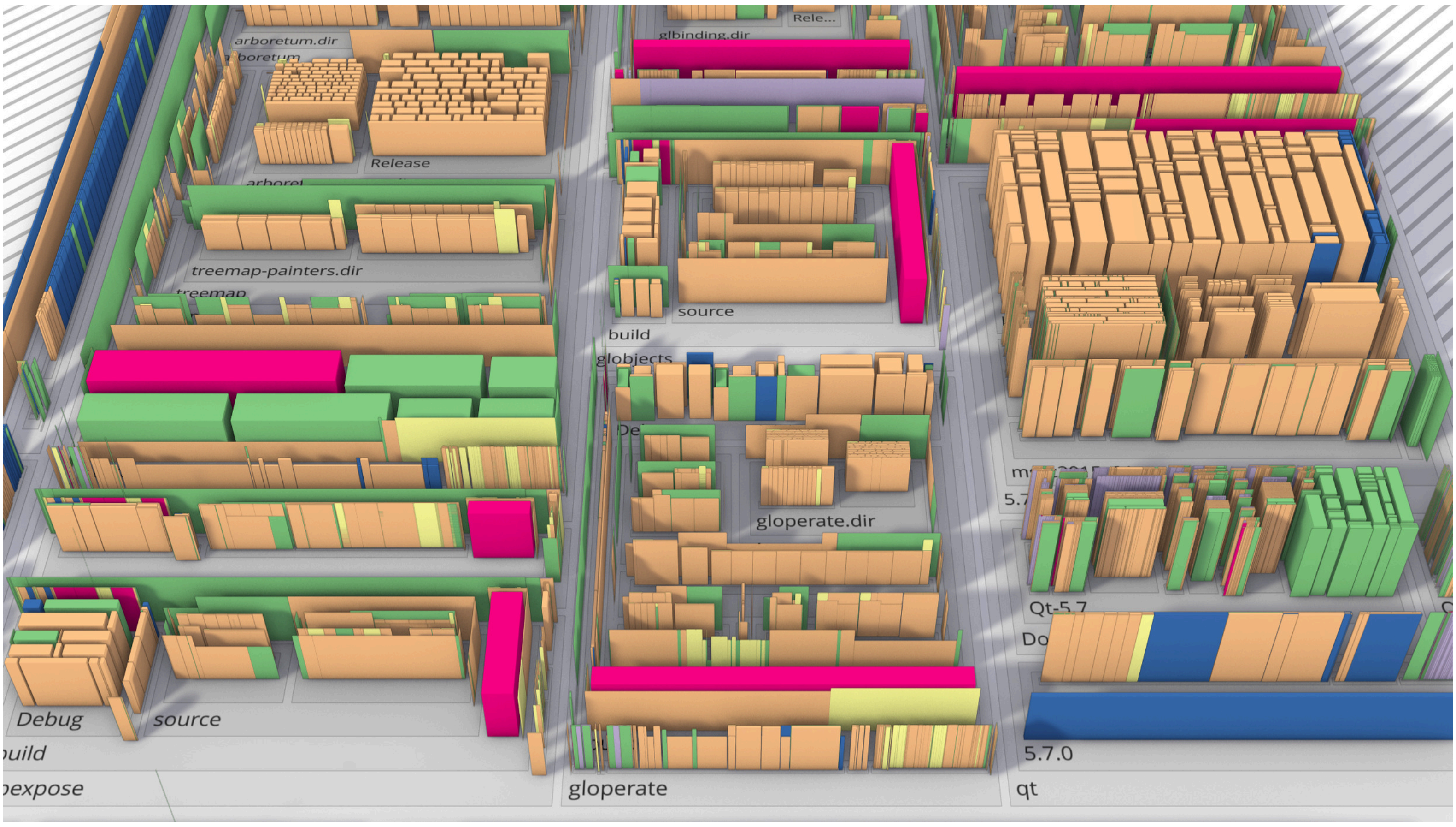


Figure 5.2: Rendering of an interactive file explorer using 3D-embedded treemaps created with *arboretum* [R13].

and differences between WebGL-based rendering and a much more capable attributed vertex cloud approach [204, 241] as employed by *arboretum*. Additionally, we discuss the benefits and limitations of using external, existing, third-party rendering engines such as three.js or babylon.js in the context of research-driven development of visualization techniques.

5.1.1 WebGL-based Rendering of 3D-Embedded Treemaps

In our WebGL-based rendering of 3D-embedded treemaps, we support the layering of cuboids, stacking cuboids on top of each other with or without a levitation gap. Layered cuboids can be used as graphical primitives for depicting changes or compositions. These layers are rendered from top to bottom since the virtual camera is usually positioned above the treemap, pointing down. Furthermore, we discard bottom faces for all cuboids of the bottom-most layer, We use instanced drawing with a single or a few (in the case of in-situ) geometry templates and attribute buffers containing the mapped, pre-processed, GPU-optimized attribute values required for the visual variables. Treemap layouting is part of the visualization mapping, and the renderer only processes the resulting quads and cuboids 2D extents (comprised of two four-component vectors).

The cuboid (and quad) color is assigned in the shader using a dedicated color scale lookup attribute for rendering purposes. The mapping is done earlier, which allows us to match the color attribute resolution to the often much smaller resolution of the color scale. The height, similarly, is also applied to the geometry in the vertex shader but was mapped to a height of 8bit or less during visualization mapping. When the height of a cuboid is zero, lateral faces are discarded.

Since we promote using flat representations of inner nodes (cf. Figure 5.2), z-fighting artifacts will most likely appear. To avoid these and relieve fill rate, we use a strict rendering sequence in combination with stencil masking: First, cuboids (and aggregates) are rendered with depth testing and stencil masking+writing enabled. Then, the inner nodes are rendered from bottom to top, still using stencil masking+writing. Finally, we disable stencil masking and rely on depth testing for the subsequent drawing of labels, gizmos, and transparent cuboids.

Font assets are created using our font asset generator [R6], and color scales are either predefined in presets or created dynamically [R3]. We create image-based ID buffers for interaction with the graphical elements and use readback caching (and asynchronous readback when available). Procedural textures, outlines, emphasizing, and levitation gaps are all created in shaders. For the cuboid's shading, we either deploy physically based rendering or use static shades per face orientation for lighting.

For filtering, we use CPU-side, range-based masking of leaf nodes. This masking can be managed using our degree of interest scoring to implement a dynamic level of detail. Once all quads and cuboids are drawn into multiple render targets, post-processing continues the image synthesis.

5.1.2 On the Use of 3rd-Party Renderers

In developing our prototypes, we often opt for a low-level approach to rendering while utilizing third-party libraries for other aspects, such as math libraries (e.g., glm, gl-matrix), event handling systems (e.g., rxjs), or comprehensive frameworks for desktop development. We recognize the advantages of using established rendering engines, including time savings, community support, extensibility, integration with existing tools, and improved stability. However, as low-level graphics engineers, we also recognize the potential drawbacks of these engines, particularly in terms of performance and flexibility. Most notably, when due to layers of abstraction, access to underlying low-level APIs and extensions is omitted.

Performance in established engines is typically optimized for preferred use cases, such as loading and displaying static or animated geometries, but not necessarily for creating tens of thousands of dynamic cuboids that might change every frame. These engines often enforce scene concepts, bake lighting, and introduce overhead through material systems or optimization structures like bounding volume hierarchies, typically not intended for massive dynamic scene changes. Additionally, the abstraction from hardware and other low-level API specifics, while beneficial for focusing on higher-level concerns, can sometimes limit access to low-level APIs. We have encountered these and similar issues with several engines used

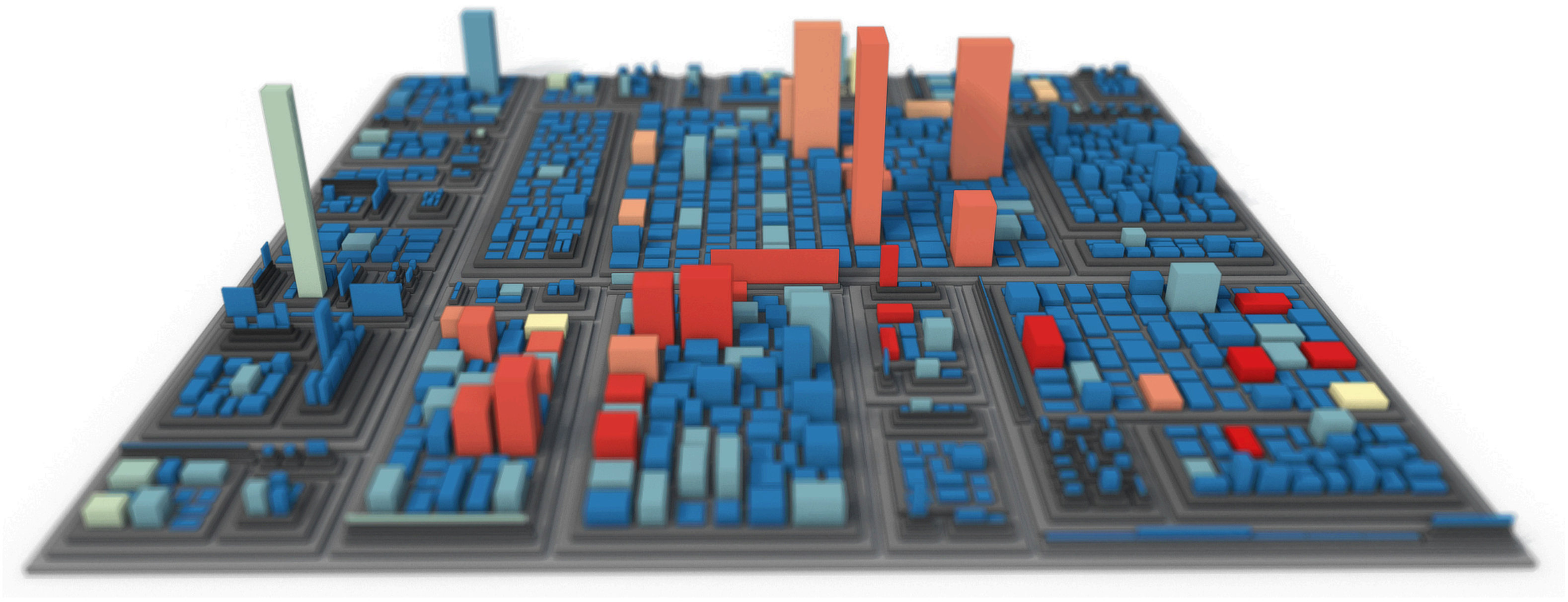


Figure 5.3: Image of a treemap created using accumulative rendering for anti-aliasing, depth of field, and soft shadows.

so far, leading us to prefer custom solutions for the high-performance rendering of dynamic graphical elements. This decision, however, comes with its trade-offs, such as spending more time on project setup, deployment, continuous integration, and quality-of-life measures like context creation, camera navigation, or typesetting and rendering text.

The choice between using existing engines or developing custom solutions depends on the specifics of a project. Using WebGL is certainly not the most suitable choice for the scientific exploration of visual variables and studying our perceptual capabilities for visual analytics. However, custom solutions become increasingly attractive if the focus is on handling massive amounts of data, interactive testing, verifying technical feasibility, actual provisioning to experts, and taking advantage of low-level graphics API features that might be unconventional or barely supported.

5.2 Responsive Accumulative Rendering

In 1990 Haeberli and Akeley described "a system architecture that supports real-time generation of complex images, efficient generation of extremely high-quality images, and a smooth trade-off between the two" and introduced the *accumulation buffer* [95]. We revived the idea of distributing sampling over multiple consecutive frames to allow sampling-based, real-time rendering techniques in applications not dependent on continuous rendering [L6]. This systematic, extensible schema allows developers to effectively handle the increasing implementation complexity of advanced, sophisticated, real-time rendering techniques while improving responsiveness and reducing required hardware resources.

The approach is motivated by the following observations related to 3D systems and application development:

- Many interactive data visualizations do not require continuous, high-quality image generation. Less frequent user interactions and data changes may allow for relaxed real-time imaging constraints.
- Business and industry applications often face stricter hardware and API constraints, resulting in a slow adoption of state-of-the-art, real-time rendering techniques.
- The single-frame design, necessitated by continuous rendering, not only increases hardware requirements but causes rendering techniques to become increasingly complex to implement and adopt.

By using *multi-frame sampling*, multiple frames are rendered and accumulated instead of rendering a single frame in response to an update request. Every accumulation result can be immediately displayed while the frame quality progressively increases. It distributes rendering cost over time while providing precise quality control primarily by the number of subsequent frames to be accumulated. Furthermore, it increases the responsiveness of our visualization, reduces resource requirements, and simplifies the implementation of various rendering effects. Favoring responsiveness and progressively overlaying high-quality features seems to gain popularity, as can be seen in Sketchfab and 3ds Max by Autodesk [5.4].

Multi-frame sampling allows us to use relatively simple rendering techniques to produce state-of-the-art effects. We demonstrated our approach for a variety of rendering techniques (Figure 5.3), i.e., *anti-aliasing* (AA), *screen-space ambient occlusion* (SSAO), OIT, *depth-of-field* (DoF) [L6], and physically-based environment and area lighting [185]. We also used it in three.js [L7, R11] and successfully applied it for progressive, tile-based rendering for volumetric brain tumor segmentation on MRI [140]. Approximating glossy screen space reflections and utilizing massive lighting to highlight cuboids with local light impact (emissive lighting) should also be possible. Overall, the multi-frame approach reduces memory usage, decreases rendering cost per frame (lowering response times), allows for better maintainable implementations, and provides simpler easy-to-understand parameterizations.

5.2.1 Composition of a Progressive Frame

An integral part of today's hardware-accelerated, real-time rendering technologies is built on *sampling* as the "process of rendering images is inherently a sampling task." [2] Sampling is generally used to approximate continuous characteristics and signals, e.g., reducing aliasing artifacts caused by insufficient depictions of continuous domains. For single-frame rendering, sampling is limited to a single frame. Increasing the number of samples improves the resulting image quality but also increases the rendering costs per frame in terms of time and memory.

Our approach distributes samples over a well-defined number of intermediate frames n_{MF}. Each frame generated during multi-frame sampling

[5.4] sketchfab.com and autodesk.com/3ds-max

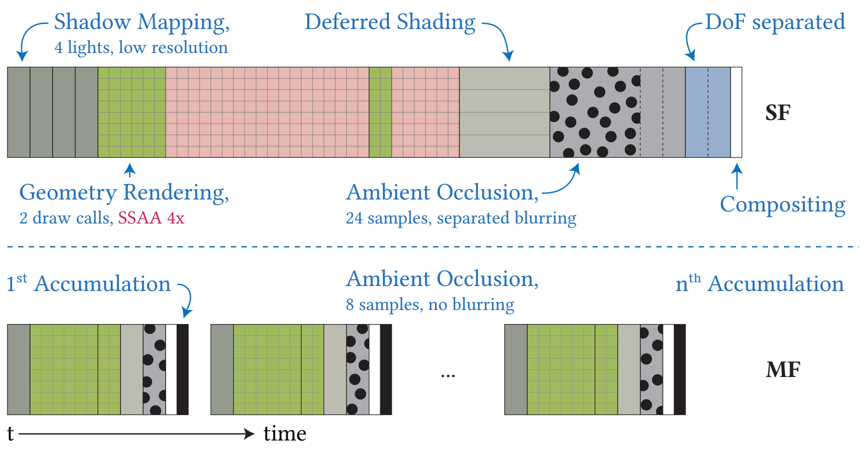


Figure 5.4: Illustration of the composition of a single-frame **SF** and an equivalent multi-frame **MF**. For **MF**, a simplified shadow pass and 4 SSAO samples per **IF** are sufficient; DoF and AA are inherent due to camera and NDC shifting. The additional impact of SSAA is indicated by red squares and could be mitigated to some extent [L6].

computes a unique sample or set of samples based on a *kernel*. Consecutive frames are accumulated until n_{MF} frames are computed, and the rendering pauses. On any update request, the accumulation process is restarted.

Assumptions. The application of multi-frame sampling in 3D systems and applications is based on the following assumptions:

- The underlying rendering uses sampling as one of its elements.
- Rendering update requests are less frequent, and responsiveness is favored over intermediate frame quality.
- A converging image quality is not disruptive for the use cases or usability of 3D systems and applications.

Instead of rendering a single frame **SF** in response to changed inputs (e.g., mapped data, camera movement), multiple intermediate frames (**IF**) are rendered and accumulated into a multi-frame **MF** (Figure 5.4). For it, a multi-frame number n_{MF} is defined, denoting the number of intermediate frames to be rendered for a full multi-frame. A progressive rendering control flow should (1) (re)start rendering of a intermediate frames immediately when any input changes, (2) update the accumulated result after every intermediate frame, and (3) stop rendering when n_{MF} frames were rendered, continue displaying the final result.

Implementation. To transform a given single-frame, sampling-based technique into a multi-frame technique, we proceed as follows:

1. We identify segments that are processed repeatedly. A parameterization that controls an iteration per frame (e.g., the number of samples and lights) often indicates such segments. These iterations are unrolled and distributed over consecutive frames.
2. We have to verify that (a) an increase in the number of segment executions results in better quality and (b) each segment's result can be accumulated throughout multiple consecutive frames.

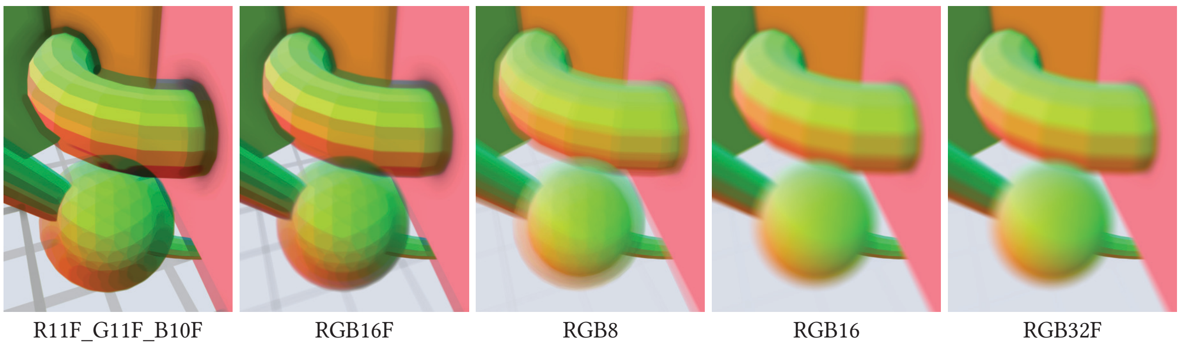


Figure 5.5: Accumulation of 1024 frames (AA and DoF) using various texture formats. F.l.t.r., the quality is increasing.

3. We adapt the original technique such that it supports an appropriate sampling characteristic: The sampling type (single or multiple samples per frame) and the spatiotemporal distribution of samples.

When multi-frame sampling is used with multiple techniques simultaneously, depending on their assembly, there might be combinations that require special attention, for example, stochastic per-fragment discarding combined with screen-space ambient occlusion.

5.2.2 **Progressive Frame Accumulation**

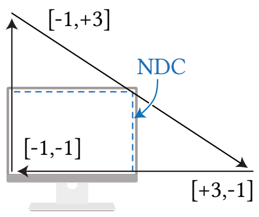


Figure 5.6: NDC-filling triangle used to avoid redundant fragment processing.

The accumulation of consecutive frames is implemented using hardware blending. Alternatively, the accumulation can be executed as a post-processing pass, e.g., using a *NDC-filling triangle*[5.5] with a fragment shader (Figure 5.6). The color c of the nth frame is blended with the accumulated average, e.g., `mix(a, c, 1.0/n)` in GLSL. This works with a single accumulation buffer as long as no adjacent fragments are processed.

On update requests, multi-frame rendering is set up for a multi-frame number of 1, and accumulation is just blitting this frame, i.e., accumulation is skipped, and the frame is rendered into the accumulation buffer directly. Since the scene and its underlying data is assumed to be static during accumulation, the time per frame is roughly constant for subsequent frames. Thus, sampling characteristics can be adapted ad-hoc for subsequent frames to converge to a targeted frame time.

The accumulation buffer's texture format should support sufficient accuracy since the weight for frame averaging gets subsequently smaller. *Web Graphics Library*[5.6] (WebGL2) provides `RGB8` by default which may limit the accuracy and quality of accumulation; small color changes get lost in the accumulation of later frames (Figure 5.5). Thus, a comparatively small n_{MF} should be applied, or formats such as `RGB32F` should be considered.

[5.5] github.com/cginternals/webgl-operate/[...]/ndcfillingtriangle.ts
[5.6] Khronos Group. *WebGL 2.0 Specification.* khronos.org/registry/webgl/[...]/2.0. 2019.

5.3 Progressive Sampling Strategies

We can often reduce rendering techniques to their core concept when we unfold them over multiple frames. Neither caching, sorting, nor other optimization strategies are required. Some techniques are virtually free since they are inherent to multi-frame rendering, namely AA and DoF. The final rendering quality, especially the convergence speed and its "temporal tranquility"—the stability and smoothness of an image or animation over time—strongly depend on a well-designed spatiotemporal sampling strategy, a *kernel*. Its characteristics include (1) the number of samples for targeted quality, (2) spatial or value distribution, (3) sample regularity and completeness for finite accumulation, (4) temporal convergence constraints to the sample sequence, and (5) additional per-fragment randomization.

We avoid GPU-based pseudo-randomness and, instead, precompute kernels for their specific multi-frame number n_{MF}; accumulating additional frames on top of that is futile. Especially when using low multi-frame numbers, this may lead to temporal clustering. The presented techniques have been implemented using the open-source, header-only libraries *OpenGL Mathematics Library*[5.7] (glm) and *glkernel* [R10]. They allow a dynamic computation of kernels of required characteristics at run-time.

We differentiate between techniques using either one or multiple samples per frame. The kernel can be provided using *uniforms* for a single sample per frame. When multiple samples per frame are used, it can be encoded using textures or buffers. However, there are minor challenges:

1. When using multiple multi-frame techniques simultaneously, sampling parameterization must be orchestrated with all effects in mind, as they all share the same multi-frame number.
2. Some combinations of techniques may impose additional sampling constraints, e.g., fragment-based dithering combined with SSAO.
3. Image-based retrieval of ID (e.g., picking) and Depth (e.g., coordinates) becomes ambiguous since IDs and depths cannot be accumulated meaningfully.

The latter issue, we include as a constraint in our kernel designs and make the first frame the single source for id and depth-related readbacks. The IDs are rendered only in the first frame, and we use two depth buffers, one for the first frame and the other for all consecutive frames.

5.3.1 Multi-frame Anti-Aliasing

Without taking specific countermeasures, image synthesis based on rasterization depicts a continuous domain and, thus, usually contains aliasing artifacts like jagged edges and moiré patterns. Anti-aliasing is commonly applied to mitigate these artifacts, e.g., supersampling and multisampling: Color or depth buffers are rendered at a higher resolution than the output resolution. While they provide good results for single-frame-constrained applications, they take up much processing power and

[5.7] Christophe Riccio. *GLM - OpenGL Mathematics Library.* glm.g-truc.net. 2019.

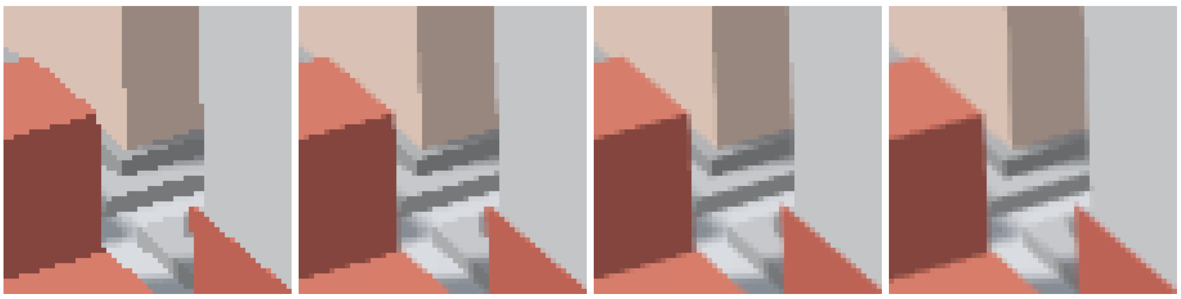


Figure 5.7: The results of accumulated sub-pixel view frustum shifting for, f.l.t.r., 1, 4, 8, and 64 multi-frames.

memory. Several sampling strategies for post-processing have been created, e.g., AMD's *Morphological Anti-Aliasing* (MLAA), Nvidia's *Fast Approximate Anti-Aliasing*[5.8] (FXAA), and Intels *Conservative Morphological Anti-Aliasing*[5.9] (CMAA). Their performance and quality vary, and they all provide a comparably low memory footprint. With temporal anti-aliasing, another type of anti-aliasing was introduced: Nvidia's *Temporal Approximate Anti-Aliasing* (TXAA) and subsequently *Multi-Frame Sampled Anti-Aliasing*[5.10] (MFAA) result in better quality and increased performance compared to *Multisample Anti-Aliasing* (MSAA). Temporal anti-aliasing uses varying sampling patterns on multiple consecutive frames, albeit limited (two subsequent frames), as they are still designed for single-frame rendering. However, all of these will likely become obsolete with the recent shift towards neural graphics technologies—namely DLSS, FidelityFX, and XeSS by Nvidia, AMD, and Intel, respectively—for temporal upscaling or generation of synthetic intermediate frames.

Approach. A sampling offset in $[-0.5, +0.5]$ is semi-randomly chosen per frame and transformed into a subpixel offset. It then added to the vertices' xy-coordinates in NDC, effectively shifting the complete NDC space (Listing 5.1). Shifting the camera's position and center in world space does not work due to the parallax effect.

```
layout(location = 0) in vec3 a_vertex;

uniform mat4 u_modelToNDC;  // aka u_modelViewProjection
uniform vec2 u_ndcOffset;   // per-frame offset in [-0.5,+0.5],
                            // pre-mult. by 1 / viewport size
void main()
{
    vec4 position_ndc = u_modelToNDC * vec4(a_vertex, 1.0);
    position_ndc.xy += u_ndcOffset * vec2(position_ndc.w);
    gl_Position = position_ndc;
}
```

Listing 5.1: Vertex shader (GLSL) implementing sub-pixel shifting of the view-frustum.

[5.8] Timothy Lottes. *FXAA*. developer.download.nvidia.com/[...]/FXAA_WhitePaper. 2009.

[5.9] Filip Strugar and Adam Lake. *Conservative Morphological Anti-Aliasing 2.0*. software.intel.com/en-us/articles/conservative-morphological-anti-aliasing-20. 2018.

[5.10] Nvidia. *Multi-Frame Sampled Anti-Aliasing Delivers Better Performance To Maxwell Gamers*. nvidia.com/[...]/multi-frame-sampled-anti-aliasing-[...]/. 2015.

Sampling Characteristics. Pseudo-randomly chosen offsets within a square work surprisingly well. The convergence can be sped up using uniform, shuffled samples or common sampling patterns [2]. For our implementation, we use shuffled Poisson-disk sampling to generate a uniform distribution of offsets for a specific number of frames (Listing 5.2). This prevents clustering and improves convergence, especially for many samples. To mitigate noticeable shifts during the first few frames, we constrain the first sample to the center of the pixel. Sorting all offsets by their length (i.e., the distance to the pixel center) is not recommended: Although it reduces the subtle shifting further, it also results in temporal clustering. Finally, we use a tile-based Poisson-disk sampling to avoid clustering at pixel edges and corners of adjacent pixels (Figure 5.8):

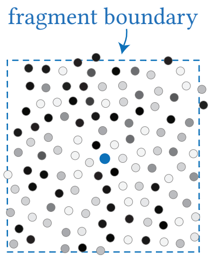


Figure 5.8: Illustration of a anti-aliasing kernel for $n_{\mathrm{MF}} = 64$. Each dot depicts a sample, the blue one being the first. Subsequent samples are colored from black to white.

```cpp
// 3D array of glm::vec2 values with extent: 64x1x1 (glkernel)
auto aaSamples = glkernel::kernel2{ 64 };
glkernel::sample::poisson_square(aaSamples, -.5f, .5f);
glkernel::shuffle::random(aaSamples, 1); // from 1 to last

// ...
// while rendering a intermediate frame:
const auto ndcOffset = aaSamples[accumCount] / viewport;
program.setUniform("u_ndcOffset", ndcOffset);
```

Listing 5.2: Generation of a spatiotemporal Anti-Aliasing kernel using *glkernel* in C++.

Performance & Remarks. Accumulating only a few frames usually results in decent anti aliasing. Our strategy takes about 16 frames to create optimal results and about 64 frames until improvement by subsequent frames becomes negligible (Figure 5.7). In comparison, pseudo-random sampling takes about 1.3 times longer to yield comparable quality and is less predictable due to clustering. In addition, we can create kernels for view-frustum shifting that, for example, blur the image by using increased offsets with Gaussian distribution.

5.3.2 Multi-frame Transparency

Rendering transparent objects is usually avoided as much as possible. The most relevant obstacle to using transparency is its implementation complexity: especially for web-based rendering clients, modern strategies such as OIT cannot easily be implemented due to limited graphics APIs and device capabilities (e.g., *k-buffers* not feasible). One solution is to use stochastic dithering combined with multi-frame sampling [L7]. Other approaches either achieve high performance by neglecting correctness and thus quality, or produce credible results by using many rendering passes or super sampling while lowering rendering performance.

Screen-door transparency applies a threshold pattern to small groups of fragments [174]; within each group, fragments with an opacity value lower than their associated threshold are discarded. Drawbacks comprise highly visible patterns and insufficient accuracy. Stochastic transparency improves on that by applying random patterns per pixel using multisampling, but still produces slightly noisy output for a single frame [69]. The

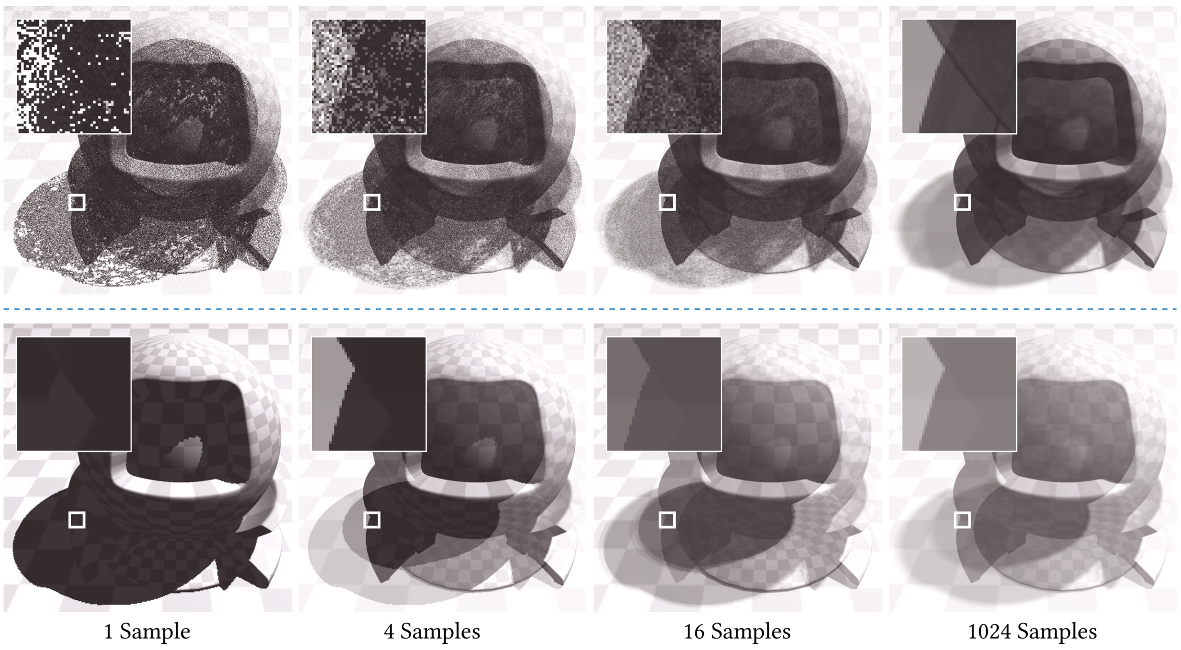


Figure 5.9: Convergence for per-fragment (top) and per-object (bottom) transparency thresholding. For per-fragment thresholding, back-face culling is on. Note the two distinct shadows resulting from the inner and outer spheres.

suggested multiple passes per frame can be transformed to a 1:1 frame mapping for multi-frame rendering of fast converging transparency.

Approach. Transparent fragments are discarded based on a random opacity threshold per fragment or object at a time. For per-fragment thresholding, more accurate and convincing results are achieved with back-face culling disabled.

Sampling Characteristics. A mapping of n distinct opacity values in the range $[0, 1]$ to associated bitmasks is precomputed on the CPU and provided to the GPU (Figure 5.10). Additionally, random, fragment-specific offsets can be used to shuffle the threshold access between adjacent fragments over time. For object-based transparency, no objects should be discarded within the first frame. Thus, all objects are initially opaque and gradually converge toward their opacity. For all consecutive frames, the object-based bit masking is skipping the draw call or discarding fragments based on an object ID (Figure 5.9).

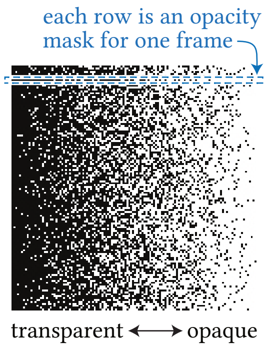


Figure 5.10: Binary transparency mask for $n_{\mathrm{MF}} = 128$: For 128 steps of opacity, 128 uniformly distributed bits are computed (columns). The nth intermediate frame uses the nth row as a mask.

Performance & Remarks. Stochastic transparency usually requires full multisampling within a single pass with up to 16 coverage samples per fragment, requiring extreme amounts of memory. In contrast, multi-frame transparency requires no additional memory at all. The amount of frames required for low-noise transparency depends on the current scene's depth complexity and camera angle. While the direct use of more advanced techniques like stochastic transparency might lead to shorter convergence times, we prefer the more basic approach for its low memory footprint,

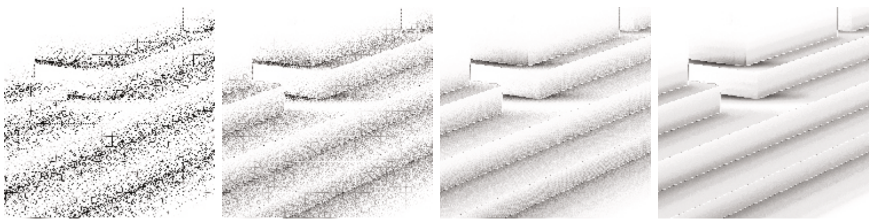


Figure 5.11: The results of accumulated SSAO for, f.l.t.r., 1, 4, 16, and 64 multi-frames, with eight samples per frame.

minimal performance overhead per frame, and implementation simplicity. In contrast to per-object transparency, the per-fragment approach is challenging to combine, for example, with SSAO, as the g-buffers become noisy and lack coherent surfaces. Per-object transparency is inaccurate and can neither account for back faces nor concave objects.

5.3.3 Multi-frame Screen-Space Ambient Occlusion

The complex interactions of real-world objects and lights poses a significant challenge even for today's most sophisticated real-time rendering engines. SSAO is commonly used to capture local light obscurance. Since Crytek's initial concept, numerous variants and enhancements have been proposed, e.g., *Improved Horizon-based Ambient Occlusion*[5.11] (HBAO+), *screen-space directional occlusion [195]* (SSDO).

Approach. We use the Screen Space Ambient Obscurance [168] with an optimized kernel and comparable quality to *Horizon-based Ambient Occlusion [19]* (HBAO). The kernel is randomly rotated per fragment to mitigate banding artifacts caused by reusing the kernel for all fragments. Using our multi-frame approach, noise resolves within a few frames. By accumulating SSAO in a dedicated target, blurring can be separated from the accumulation and reduced with increased sampling. This mitigates noise in the beginning and smoothly converges to a crisp result.

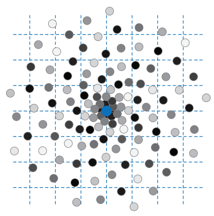


Figure 5.12: Illustration of a SSAO kernel for $n_{\mathrm{MF}} = 16$ and 8 samples per frame. Each dot depicts a sample, the blue one being the first. Subsequent samples are colored from black to white.

Sampling Characteristics. The kernel uses a spiral-shaped pattern projected to a local tangent plane to produce the sample offsets (Figure 5.12). We sort the kernel samples in alternating order to ensure a more steady improvement in image quality for subsequent frames. For n frames and m samples per frame the set of samples S_i for frame i is $S_i = \bigcup_{j=0}^{m-1}\{s(jn+i)\}$, with $s(k)$ providing the kth sample of the original kernel. This sequence ensures that sampling different scales in every intermediate frame.

Performance & Remarks. The number of frames required for a crisp and noise-free result depends mainly on the desired ambient occlusion radius. We found about 480 samples for moderate settings to provide a nearly artifact-free result, i.e., 60 frames when using eight samples per frame (Figure 5.11). Since most screen-space-based ambient occlusion techniques are similar, techniques such as SSDO, also accounting for local light bleeding, should be just as suitable for multi-frame rendering.

[5.11] Louis Bavoil. *HBAO+*. geforce.com/hardware/technology/hbao-plus/technology. 2013.

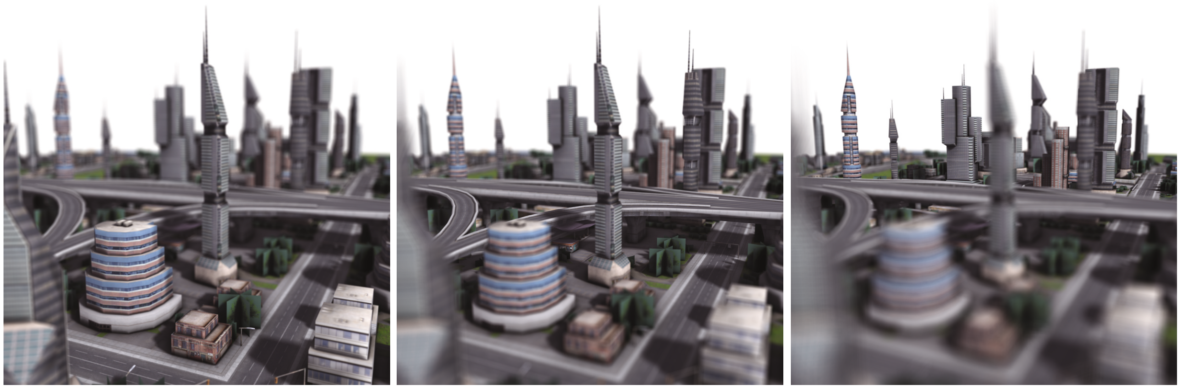


Figure 5.13: Rendering of a small 3D city using multi-frame for depth-of-field, anti-aliasing, screen-space ambient occlusion, and soft shadows, without causing boundary discontinuities. The focus shifts from front (left) to back (right).

5.3.4 Multi-frame Depth of Field

Depth of field is an effect that can guide a user's attention to a particular region within a scene. The effect blurs objects depending on their distance to a chosen focal plane or point, which usually lies on an object or region of interest. DoF is often implemented as post-processing, mixing the sharp focus field with one or two (near and far field) blurry color buffers per fragment, based on the fragment's distance to the focal point or plane [41]. More advanced techniques are also available, usually reducing boundary discontinuities and intensity leakage artifacts and accounting for partial occlusion using multiple focus layers [213]. Though multi-layer approaches can be adapted to multi-frame rendering, we present a minimal approach favoring rendering speed over convergence time while still enabling high-quality DoF (Figure 5.13).

Approach. For DoF, we use a random two-dimensional vector on a unit disc as a per-frame sample. This vector indicates for each point in a scene where on its *circle of confusion* (CoC) it should be rendered on the image plane. With subsequent sampling, each point gradually covers its circle of confusion. Similar to our AA approach, the sample vector is added to the vertices' xy-coordinates in a vertex shader; this time, however, in view space before applying the projection matrix (Listing 5.3). It is scaled with the vertices' z-distance to the chosen focal plane. Additional post-processing passes per frame, e.g., separated blurring, are not required.

```glsl
// point in circle of confusion (opt. pre-multiplied by scale)
uniform vec2 u_cocOffset;
// z-distance to the camera at which objects are in focus
uniform float u_focalDistance;

void main()
{
    vec4 position_eye = u_modelView * vec4(a_vertex, 1.0);
    position_eye.xy +=
        u_cocOffset * (position_eye.z + u_focalDistance);
    gl_Position = u_projection * position_eye;
}
```

Listing 5.3: Vertex shader (GLSL) implementing CoC shifting in camera space.

Sampling Characteristics. Poisson-disk samples are sorted by their distance to the center. The center is used as the first sample to omit camera shaking. Additionally, arbitrary Bokeh shapes can be created by masking the samples with the desired shape (Listing 5.4):

```cpp
auto cocSamples = glkernel::kernel2{ 128 };
glkernel::sample::poisson_square(cocSamples, -1.f, 1.f);
// opt: filter samples by position using a 'bokeh' bitmask
glkernel::mask::by_value(cocSamples, bitmask);
glkernel::sort::distance(cocSamples, 0.f, 0.f);
```

Listing 5.4: Generation of a spatiotemporal DoF kernel using *glkernel* in C++.

Performance & Remarks. The number of samples needed for a high-quality image depends on the largest CoC present in the scene, which is proportional to the desired effect scale and the distance of a point to the focal plane. Multiplying the maximum radius in pixels by ten adequately provides a proper multi-frame number. Full convergence may take up to a few seconds, with the depth of field gradually increasing over time, interestingly often not consciously perceived by users.

5.4 3D-embedded Treemaps using treemap.ts

In the following, we show an excerpt of how we can use our *treemap.ts* [R12] library, and *webgl-operate* [R4] to create an interactive 3D-embedded treemap using WebGL and an HTMLCanvasElement.

Initialization. We start by creating a treemap visualization, providing access to a specialized treemap renderer. Alongside, we create a webgl-operate canvas on our HTML canvas element, which creates a controller that delegates multi-frame rendering to the assigned renderer. Thus, we need to assign the visualization's renderer to our canvas which kicks off the rendering controller so we have everything running (Listing 5.5):

```typescript
import { Configuration, Navigation, Renderer, Visualization
    gloperate, // webgl-operate re-export
} from 'treemap';

const visualization = new Visualization();
const renderer = visualization.renderer as Renderer;

const canvas = new gloperate.Canvas('canvas');
canvas.framePrecision = gloperate.Wizard.Precision.half;
canvas.controller.multiFrameNumber = 64;
canvas.renderer = renderer; // initiates rendering loop (raf)

const config = new Configuration();
visualization.configuration = config;
```

Listing 5.5: Initial setup for creating treemaps using *treemap.ts* and *webgl-operate*.

We also created a configuration, which provides the API for configuring all treemap related data, and assigned it to the visualization. The visualization reacts to configuration changes, maintains an *treemap description*, and delegates the rendering of it to the renderer.

Topology. Next, we create a tree structure using tuples (Listing 5.6). The structure created in the listing matches the treemaps shown in Figure 3.23:

```
config.topology = {
    edges: [ // [ parentID, nodeID ], root ID is always 0
        [0,  1], [1, 10], [1, 11], [1, 12], [1, 13],
        [1,  3], [3, 30], [3, 31], [3, 32], [3, 33],
        [0,  2], [2,  4], [4, 40], [4, 41], [4, 42], [4, 43],
        [2, 20], [2, 21], [2, 22], [2, 23],
    ],
    format: 'tupled', // or 'interleaved'
};
```

Listing 5.6: Configuration of the topology of a treemap using *treemap.ts*.

Layout. For the layout configuration, we specify the algorithm to be used, e.g., `'strip'` [217], optionally enable flattening of directories, and configure padding (Listing 5.7). For padding, most notably, we are interested in accessory padding, which provides us the additional space for labeling:

```
config.layout = {
    algorithm: 'strip', weight: 'bufferView:Weight',
    flattenDirectories: true,
    parentPadding:    { type: 'absolute', value: 0.02 },
    siblingMargin:    { type: 'relative', value: 0.20 },
    accessoryPadding: { type: 'absolute',
        direction: 'bottom', value: [0.0, 0.04, 0.03],
        relativeAreaThreshold: 0.4, targetAspectRatio: 2.0, },
};
```

Listing 5.7: Configuration of the layout of a treemap using *treemap.ts*.

Attributes. Before we can continue with attribute mapping, we need attributes and actual data (attribute values). The configuration supports *buffers* and *buffer views*, which allows for composing complex data transformation graphs. Thereby, (1) backtracking is used to transform only data that is actually used by the visualization, and (2) caching is integrated to avoid duplicate buffer transformations. In the following listing (Listing 5.8), we use it, for example, to compute attribute values for the inner nodes of the `'Weight'` buffer view, which is assigned as a weight attribute used in the layout, or create a normalized view based on `Gamma`:

```
config.buffers = [
  { identifier: 'Alpha', type: 'uint8',
    data: new Uint8Array( [ 0, 0, 1, 1, 1, ..., 1, 1 ]) },
  { identifier: 'Beta', type: 'uint8',
    data: new Uint8Array(21) },
  { identifier: 'Gamma', type: 'uint8',
    data: new Uint8Array( [ 0, 0, 1, 2, ..., 15, 16 ])  },
  { identifier: 'Delta', type: 'float32',
    data: new Float32Array(21) },
  { identifier: 'Epsilon', type: 'uint8',
    data: new Uint8Array(21) } ];

config.bufferViews = [
  { identifier: 'Weight', source: 'buffer:Alpha',
      transformations: [{ type: 'propagate-up',
          operation: 'sum' }] },
    { identifier: 'Beta-Transformed', source: 'buffer:Beta',
      transformations: [{ type: 'range-transform',
          sourceRange: [0, 64], targetRange: [0, 0.75] }] },
    { identifier: 'Gamma-Normalized', source: 'buffer:Gamma',
      transformations: [{ type: 'normalize',
          operation: 'min-to-max' }] } ];
```

Listing 5.8: Configuration of the attributes used by a treemap using *treemap.ts*.

Mapping. Now we can continue with the mapping to color and height; we configure two stacked layers of cuboids and, for each of them, assign an attribute to color and height (Listing 5.9). Lastly, we assign each layer a color map to look up colors in:

```
config.colors = [
  { identifier: 'emphasis',  space: 'hex', value: '#00b0ff' },
  { identifier: 'auxiliary', space: 'hex',
        values: ['#00aa5e', '#71237c'] },
  { identifier: 'inner', space: 'hex',
        values: ['#e8eaee', '#eef0f4'] },
  { identifier: 'leaf',  space: 'hex',
        values: ['#ffffff', '#fed500', '#fe8500', '#e62325'] },
];

config.geometry = {
    parentLayer: { showRoot: false },
    leafLayers: [
        {   colorMap: 'color:leaf',
            colors: 'bufferView:Gamma-Normalized',
            height: 'bufferView:Beta-Transformed' },
        {   colorMap: 'color:leaf',
            colors: 'buffer:Epsilon',
            height: 'buffer:Delta' },
    ],
    heightScale: 0.5,
    levitate: 6.0, // unit: device-independent pixel
};
```

Listing 5.9: Configuration of colors and the attribute mapping using *treemap.ts*.

Labeling. In the following, names are randomly selected from the botanical tree catalog and assigned to the inner nodes using a callback (Listing 5.10). The callback allows developers to fetch labels on demand:

```
const trees = ['Abies Grandis', ..., 'Viburnum Opulus'];
const names = trees.sort(() => Math.random() - 0.5).slice(0,4);
const labels = new Map<number, string>([
    [1, names[0]], [2, names[1]], [3, names[2]], [4, names[3]],
    /* ... leaf node labels can also be assigned here */ ]);

config.labels.callback = (idsToLabel: Set<number>,
  callback: Visualization.NameSetCallback) => callback(labels);
```

Listing 5.10: Setup of a callback to gather label data for a treemap using *treemap.ts*.

Randomize Heights. Data mapped to visual variables can be changed dynamically. The following listing shows how the `'Delta'` buffer, mapped to the height of the upper cuboid layer, is filled with random values (80% of which are 0.0). After that, this specific alteration is reported, and the renderer is invalidated (Listing 5.11):

```
const delta = config.buffers.find(
    (buffer) => buffer.identifier === 'Delta');
delta?.data = Float32Array.from({ length: 21 },
    () => Math.random() > 0.8 ? (Math.random() * 0.25) : 0.0);

config.altered.alter('geometry');
renderer.invalidate();
```

Listing 5.11: Example for a changing attribute values used by a treemap using *treemap.ts*.

Node Events. Lastly, we show how highlighting of nodes can be implemented. For it, we subscribe to respective asynchronous treemap observables, e.g., `nodeSelect$` (Listing 5.12). This allows us to be notified when a cuboid is selected by the user and adjust the configuration to emphasize or highlight the selected node:

```
const emphasize = (event: Navigation.NodeEvent) => {

    // initializes and returns config.geometry.emphasis
    const emphasis = getOrCreateEmphasis();
    const index = emphasis.highlight.indexOf(event.node);

    if (index > -1) {
        emphasis.highlight.splice(index, 1);
    } else {
        emphasis.highlight.push(event.node);
    }
    renderer.invalidate(); };

renderer.navigation.nodeSelect$.subscribe(emphasize);
```

Listing 5.12: Example for subscribing to node events of a treemap using *treemap.ts*.

The configuration uses (1) strict schema checking, (2) falls back to defaults whenever empty or invalid configurations are made, and (3) allows for granular alteration reporting and checking. The latter allows the visualization to keep the number of mapping changes minimal. The same alteration mechanism is used for the treemap description to enable the renderer to decide with custom granularity when to react to specific changes.

When data is fetched from custom endpoints, we typically use specialized adapters that process the incoming data and create a treemap configuration. This allowed us to keep the API of *treemap.ts* use case agnostic and successfully integrate it into different visualization domains.

More capabilities are exposed by the treemap.ts API, such as procedural textures, aggregation, height threshold, and animated transitions, all of which are not shown here. Not all concepts and techniques presented in this work are integrated into the *treemap.ts*. But instead, for example, in *Arboretum* [R14], forks or spikes [R8], other libraries, and demos [R13].

6 Software Visualization using 3D-Embedded Treemaps

The contents of this chapter are based on the following original publications:

D. Limberger, B. Wasty, J. Trümper, and J. Döllner. "Interactive Software Maps for Web-based Source Code Analysis". In: *Proc. ACM Web3D.* 2013 [L1]

D. Limberger, W. Scheibel, M. Trapp, and J. Döllner. "Advanced Visual Metaphors and Techniques for Software Maps". In: *Proc. ACM VINCI.* 2019 [L14]

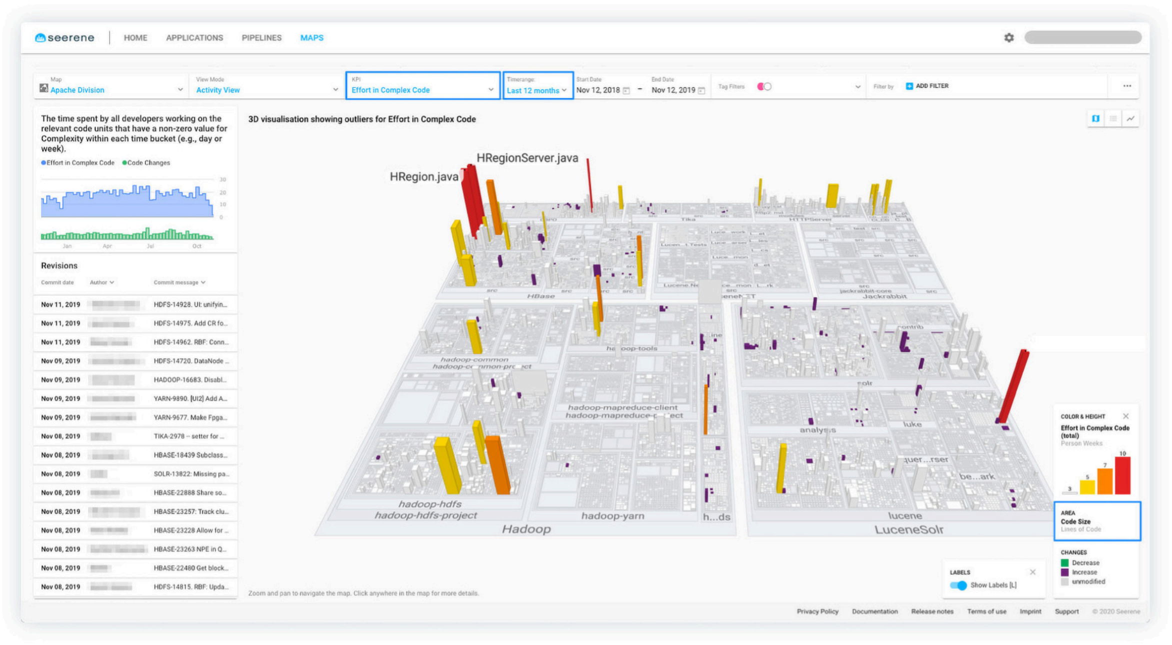
D. Limberger, W. Scheibel, J. Döllner, and M. Trapp. "Visual Variables and Configuration of Software Maps". In: *Journal of Visualization* (2022) [L23]

The *software development life cycle* (SDLC) is a continuous cycle involving planning, designing, developing, testing, deploying, maintaining, and refactoring software over many years. Maintenance "is typically a significant portion of life cycle costs" [212] and is associated with debt [3, 38, 70] which accumulates if not addressed upfront. Code comprehension is a critical part of the software maintenance process, involving cognitive processes that allow programmers to understand a codebase's structure, functionality, and behavior. Code comprehension enables programmers to perform various tasks such as adaptive, perfective, and corrective maintenance, code reuse, and code leverage. These tasks require programmers to identify, locate, modify, test, and document code changes affecting software quality, performance, and functionality.

Figure 6.1: An interactive software engineering dashboard visualizing the latest development activities and metrics by interactive software maps [L1].

Up-to-date knowledge of the software project, its processes, and source code is vital in software engineering. Visualization is essential in communicating data, forecasts, and reasoning to programmers and other stakeholders [234]. It can help raise awareness of the various factors involved in technical debt since its "causes are [...] mostly invisible (i.e., architectural debt, structural debt, test debt, documentation debt, code complexity, low internal quality, coding style violations, and code smells)." [113] "While practitioners are aware of the concept of technical debt, this knowledge is implicit, and hence, underutilized." [113] 3D-embedded software maps can not only (1) provide visual access to SWSE data but (2) expose technical dept and (3) facilitate making this knowledge explicit (Figure 6.1).

We successfully integrated 3D-embedded software maps in (1) a desktop visualization tool targeting experts and, more recently, (2) a digital en-



Map: Apache Division
View Mode: Activity View
KPI: Effort in Complex Code
Timerange: Last 12 months
Start Date: Nov 12, 2018 – End Date: Nov 12, 2019
Tag Filters
Filter by

The time spent by all developers working on the relevant code units that have a non-zero value for Complexity within each time bucket (e.g., day or week).

● Effort in Complex Code ● Code Changes

3D visualisation showing outliers for Effort in Complex Code

Revisions

Commit date	Author	Commit message
Nov 11, 2019	[illegible]	HDFS-14928. UI: unifyin...
Nov 11, 2019	[illegible]	HDFS-14975. Add CR fo...
Nov 11, 2019	[illegible]	HDFS-14962. RBF: Conn...
Nov 09, 2019	[illegible]	HDFS-14720. DataNode ...
Nov 09, 2019	[illegible]	HADOOP-16683. Disabl...
Nov 09, 2019	[illegible]	YARN-9890. [UI2] Add A...
Nov 09, 2019	[illegible]	YARN-9677. Make Fpga...
Nov 08, 2019	[illegible]	TIKA-2978 -- setter for ...
Nov 08, 2019	[illegible]	HBASE-18439 Subclass...
Nov 08, 2019	[illegible]	SOLR-13822: Missing pa...
Nov 08, 2019	[illegible]	HBASE-22888 Share so...
Nov 08, 2019	[illegible]	HBASE-23257: Track clu...
Nov 08, 2019	[illegible]	HBASE-23228 Allow for ...
Nov 08, 2019	[illegible]	HBASE-23263 NPE in Q...
Nov 08, 2019	[illegible]	HBASE-22480 Get block...
Nov 08, 2019	[illegible]	HDFS-14815. RBF: Upda...

Zoom and pan to navigate the map. Click anywhere in the map for more details.

COLOR & HEIGHT
Effort in Complex Code (total)
Person Weeks
3 5 7 10

AREA
Code Size
Lines of Code

CHANGES
🟩 Decrease
🟪 Increase
⬜ unmodified

LABELS
Show Labels [L]

Figure 6.2: 3D-embedded software map integrated into a Digital Engineering Platform showing efforts in complex code over the last 12 months (activity view). The same attribute is mapped to height and color for emphasis, and both labeling and aggregation are enabled. Image courtesy of Seerene.

gineering platform for software mining and analysis (Figure 6.2). Since the collected software system and software engineering data is mostly inherently tree-structured (cf. chapter 2), it could be visualized using the web-based, 3D-embedded software map we developed. Our software map is one of the few core tools for visual root cause analysis in the 'digital boardroom' of the platform and is based on a variant of our treemap library. It facilitates locating and comprehending higher-level metrics within the context of the respective software systems, down to the source code units. These collaborations provided insights into how practitioners approach visual software analytics and how much they utilize 3D-embedded software maps. We will briefly outline fundamental software metrics and highlight how these are used in practice through *map themes*. For the subsequent assembly of software maps, we give an overview of which visual variables can be used in conjunction, discuss two-state mapping, outline constraint-based camera navigation, and show results of an evaluation of our treemap's rendering performance.

6.1 Static Source Code Metrics

Seemingly simple metrics, such as the number of lines of code, are the genuine attributes we associate in a 1:1 mapping to source code units depicted by our visualization. They can be mined using static source code analysis tools and characterize the individual code units. When tracked over time, these metrics establish the groundwork for root-cause analysis and higher-level metrics employed for decision support. Higher-level metrics, however, abstract their measurement basis and obfuscate the underlying data and computations into a new discrete measure. By correlating and directly superimposing multiple 'raw' measurements, we not only (1) avoid obstructing their values but, furthermore, (2) allow for direct exploration of the relationships between characteristics of code units while (3) maintaining their 1:1 association to code units. The following metrics form the basis for these correlation mappings:

Lines of Code LoC Probably the most prominent group of measures for the size of a software system.[6.1] It includes specializations on *source lines* (SLOC), *comment lines* (CLOC), *non-comment lines* (NCLOC), *effective lines* (ELOC), *logical lines* (LLOC), *blank lines* (BLOC), *'real' lines* (RLOC), or *number of statements* (NOS).

Density of Comments (DC) The ratio of CLOC to LOC.

Nesting Level (NL) Measure the level of containment of a logical source code block, with variations like NLMean (mean NL over multiple code units) and NL#+ (e.g., NL4+ measures the number of lines in NL4 and higher) are related to cognitive effort for comprehension.

Cyclomatic Complexity (CC) Indicator for maintenance efforts, counting the number of linearly independent paths of the control flow through a code unit [165] scoped to functions, methods, classes, or modules.

There are other metrics, such as the Halstead Complexity measures [99], the maintainability index (MI) [54], metrics for object-oriented software engineering (MOOSE) [49], or metrics for object-oriented design (MOOD) [1]. 3D-embedded software maps, in general, allow us to visualize any metrics for the source code hierarchy. The simultaneous display of multiple higher-level metrics most likely results in incomprehensible mappings. Moreover, the more complex a metric becomes, the more difficult it is to measure that metric reliably and quickly.

Since our map themes exclusively map LOC derivatives to the area, we rely on its availability. In cases where no source code mining service is available, we (1) use *Tokei*[6.2] for basic static statistics, i.e., the number of files and lines-of-code measurements and (2) use local reporters usually provided by testing frameworks to measure test coverage. Furthermore, *git-log*[6.3] has powerful log filtering and formatting facilities that can be used to track activities and changes directly from within the repository.

6.2 Map Themes for Visual Software Analytics

Software maps are a general-purpose interactive user interface for information display in software analytics. A catalog of *map themes*—sometimes also referred to as *perspective* or *view* [21]—compiles commonly used attribute selections and mappings to visual variables relevant for domain-specific tasks. Even though not described in detail here, every map theme also

[6.1] For a rough orientation of source code sizes in LOC, the visualization "Millions of Lines of Code" by David McCandless. [166] is well suited. It is also available online: informationisbeautiful.net/visualizations/million-lines-of-code/ (2015).

[6.2] Erin Power. *Tokei.* github.com/XAMPPRocky/tokei. 2016.

[6.3] git-scm.com/docs/git-log

accounts for color scale specification, custom attribute transformations, and otherwise relevant parameters for the specific visual variables used.

The following map themes are used for visual exploration to support common software engineering challenges. All themes map LLOC to area:

Technical Dept maps NL to color and CC to height. Allows us to reveal and monitor technical debts inherent to individual code units.

Risk of Knowledge Drain maps the *number of active developers* (per node) to color and a difficult-to-comprehend measure such as NL or CC to height. This allows to identify complex code units known only by a few developers and reveal knowledge distribution.

Code Comprehension maps *change frequency* to color and code complexity, e.g., NL4+ or CC, to height. In addition, *test coverage* is mapped to sketchiness and superimposed to emphasize difficult-to-comprehend modules that have been significantly altered but rarely tested.

Development Hot Spots maps the *number of weeks with changes* to color and CC to height. This is similar to a heat map and allows for highlighting where development efforts are directed.

Complexity Hotspot maps CC to color and NLMean to height. Allows to locate source code files with a disproportional nesting level compared to the implemented business logic.

Test Coverage maps *test coverage* to color, and any complexity metric, e.g., CC to height. In addition, change in test coverage could be mapped to shininess (coverage increased) and roughness (coverage decreased).

A map theme can use a nonlinear mapping such as logarithmic scaling to emphasize or account for certain aspects of the depicted data. In our implementation, this is done by interposing *buffer views* onto the attributes with respective transformations.

6.3 Assembling Map Themes

When assembling a map theme, we usually gather all input characteristics (data types, data resolution, etc.) and specify the use case and related tasks as clearly as possible. The resulting software maps differ in their used metaphors and visual variables. For example, does the task consist of stepwise exploration or subsequent queries that could be addressed within the visualization directly? Is the visualization integrated into a multiple-linked-view setup, preceded by other investigations, or part of a broader analytics process? At what time and at which frequency the user needs to complete the task or use the visualization, respectively?

In most cases, our software maps, first and foremost, assist the user in identifying regions or nodes of interest. Then, additional visual variables are employed for subsequent exploration of the relevant data for those nodes. Tables 3.1 and 6.1 are intended to facilitate the identification of promising visual variables and combinations most appropriate for the task.

Visual Variable	Area (Foot Print)	Color	Height	Transparency	Light Emission	Stacking	Stacking (global)	Segments	Shape Type	Shape Parameter	In-situ	Contour Width	Contour Color	Stippling	Sketchy Contour	Pattern	Noise	Shading	Hatching	NL-Contour	Color Weaving
Area (Foot Print)	–																				
Color	✓	–																			
Height	✓	✓	–																		
Transparency	✓	◦	✓	–																	
Light Emission	✓	◦	✓	◦	–																
Stacking	✓	✓	✓	✓	✓	–															
Stacking (global)	✓	✓	✓	✓	✓		–														
Segments	✓	✓	✓	✓	✓	✓	✓	–													
Shape Type	✓	✓	✓	✓	◦	◦	◦	◦	–												
Shape Parameter	✓	✓	✓	✓	✓	◦	◦	◦		–											
In-situ	✓	✓	✓	✓	✓			◦	◦	◦	–										
Contour Width	✓	✓	✓	◦	◦	✓	✓	◦	◦	◦	◦	–									
Contour Color	✓	◦	✓	◦	◦	✓	✓	✓	◦	◦	◦	✓	–								
(Contour) Stippling	✓	◦	✓	◦	◦	✓	✓	◦	◦	◦	◦	✓	✓	–							
Sketchy Contour	✓	✓	✓	◦	◦	✓	✓	◦	◦	◦	◦	✓	◦	◦	–						
Surface Pattern	✓	✓	✓	◦	◦	✓	✓	✓	✓	✓	◦	✓	✓	◦	◦	–					
Surface Noise	✓	✓	✓	◦	◦	✓	✓	✓	✓	✓	◦	✓	✓	◦	◦		–				
Surface Shading	✓	✓	✓	◦	◦	✓	✓	◦	✓	✓	◦	✓	✓	✓	◦	◦	◦	–			
(Surface) Hatching	✓	✓	✓	◦	◦	✓	✓	✓	✓	✓	◦	✓	✓	◦	✓				–		
NL-Contour	✓	◦	✓	◦	◦	✓	✓		◦							✓	✓	✓	✓	–	
Color Weaving	✓	◦	✓	◦	◦	◦	✓	✓	✓	✓	◦	◦	◦	◦	◦	◦				◦	–
Height Threshold	✓	✓	✓	◦	✓				◦	◦	◦	✓	✓	✓	◦	✓	✓	✓	✓		✓

Table 6.1: This symmetric is-combinable-with matrix captures our assessment for technical compatibility of visual variables. However, using four or more visual variables simultaneously should always be carefully considered regardless of their compatibility. Legend: ✓ – well combinable | ◦ – difficult to combine.

Whether or not the visual variables can be used depends most certainly on the limitations of the targeted devices and used frameworks. As stated before, transparency, emissive lighting, and similar, technically demanding visual variables are highly convenient but difficult to implement for interactive use.

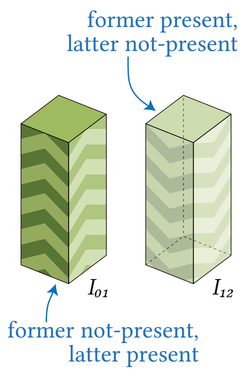


Figure 6.3: In-situ templates that help resolve *two-state ghosting*.

Two-State Mapping. Since software metrics are typically mined continuously or regularly, we enable users to optionally select two distinct attribute revisions for the mapping instead of one. However, each map theme should include a respective specification for two-state mappings. The design space can sometimes be difficult to grasp here, so we presented and discussed our preferred in-situ templates for simultaneous two-state attribute mapping using color and height, illustrated in Figure 3.18. Selecting two or more attribute revisions also introduces the problem of nodes that are only present in one of the revisions; *two-state ghosting*. This can also be resolved using our preferred in-situ templates, namely,

I_{01} or I_{12}, depending if the node is present in the former or latter state. By assuming R_1 to be the node that is present, and either R_0 (former) or R_2 (latter) being not present, one can use I_{01} (latter color, former height zero, arrows pointing upwards,) or I_{12} (transparent, former color, latter height zero, arrows pointing downwards) accordingly (cf. Figure 6.3).

Constraint-based Navigation. Interaction techniques for 2D and 3D-embedded treemaps typically cover navigational tasks (e.g., pan, zoom, rotate), selection and filtering tasks (e.g., implicit or explicit queries specified via GUI elements) as well as detail-providing interactions (e.g., tooltips on hovering, linking cuboids to external resources). For a comprehensive overview of navigation and interaction techniques, we refer to Jankowski and Hachet [121]. We use a world-in-hand metaphor for controlling the virtual 3D camera control, supporting panning, zooming, and rotation. However, we add a few simple, but essential constraints, assuming a mouse is the primary input device for camera control:

1. The virtual camera's center is constrained to $y = 0$ (ground floor) and the extent of the software map's root node, i.e., a unit square. This sounds simple but is most effective in keeping users from losing sight of the software map ever.

2. During panning, the initial scene intersection point at the mouse position is constrained to the mouse position.

3. Rotation is constrained to the camera's center. The camera's altitude must be within a minimum angle and a minimum angular distance to the up vector. This prevents the software map from being seen from below or upside down.

4. While zooming, the camera distance is reduced towards the intersection point at the mouse position.

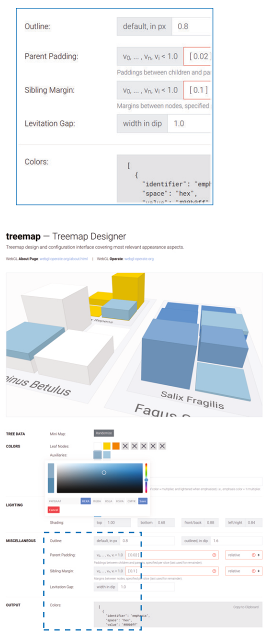

Outline: default, in px 0.8
Parent Padding: $v_0, \ldots, v_n, v_i < 1.0$ [0.02
Paddings between children and pa
Sibling Margin: $v_0, \ldots, v_n, v_i < 1.0$ [0.1]
Margins between nodes, specified
Levitation Gap: width in dip 1.0
Colors:
```
[
  {
    "identifier": "emph
    "space": "hex",
```
treemap — Treemap Designer
Treemap design and configuration interface covering most relevant appearance aspects.
WebGL **About Page**: webgl-operate.org/about.html | WebGL **Operate**: webgl-operate.org

Figure 6.4: Screenshot of our *Treemap Designer* [R7] that guides the treemaps assembly by creating map theme configurations.

Map Theme Designer. Map themes can become quite complex, especially considering the comprehensive parameterization exposed by the *treemap.ts* API (cf. section 5.4) and the number of visual variables (cf. Table 6.1), each often with its variations. We have developed a *Treemap Designer* [R7] that guides visualization designers, i.e., users of *treemap.ts*, in assembling treemaps. During configuration, (1) a live preview of an interactive 3D-embedded treemap reacts immediately to changes, and (2) a JSON-based map theme configuration is generated that can be used or shared with others. The tool guides users through grouped configuration options that mirror the configuration structure, provides meaningful defaults and parameter descriptions, constrains parameter value ranges, avoids misleading configurations, and communicates best practices. We only created the tool's first iteration (Figure 6.4) and suggest including presets and integrating more visual variables in the future.

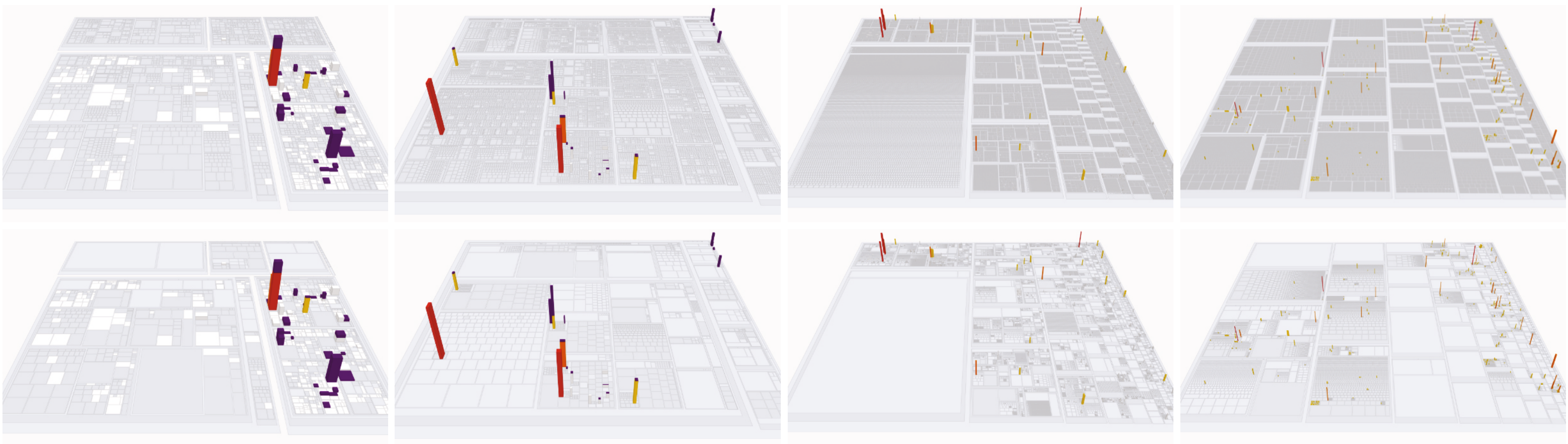


Figure 6.5: Renderings of the four different-sized datasets used in our performance evaluation, with 3 524, 18 551, 448 322, and 500 071 nodes from left to right, respectively, with (bottom) and without (top) aggregation enabled.

6.4 Rendering Performance Evaluation

We conducted a rendering performance test on various devices and browsers to evaluate the responsiveness of treemaps created with *treemap.ts*. We test four different-sized non-synthetic datasets, in three browsers (thereby both WebGL versions), on three different devices. Each dataset was measured with both aggregation off and on (Figure 6.5). A virtual camera with a locked altitude was rotated around the treemap multiple times. The camera position was updated every time a full multi-frame of 64 intermediate frames was finished. The refresh rate measures are enlisted in Table 6.2.

OS	Browser	CPU	GPU	WebGL	**Refresh Rate in Hz at 1280 × 720px**							
Windows 10	C	i7	960M	2	1 940	227	661	940	45	23	35	106
	C	i5	520	2	458	117	237	334	21	15	16	59
	C	i7	530	2	585	166	287	499	22	19	17	81
	E	i7	530	1	704	133	304	427	23	13	17	55
macOS	C	m3	615	2	485	280	244	571	21	37	15	144
	S	m3	615	1	566	286	267	641	21	13	12	150
				Aggregation	–	✓	–	✓	–	✓	–	✓
				Number of Nodes	3 524		18 551		448 322		500 071	
				# Inner Nodes	694		4 230		140 422		70 861	
				# Leaf Nodes	2 830		14 321		307 900		429 210	

Table 6.2: Responsiveness of treemaps depicting four different-sized data sets, each without and with aggregation and with attributes mapped to area, color, and height, using *treemap.ts*. The measurements were made for different browsers, i.e., Chrome/73.0.3683.86 (C), Edge/18.17763 (E), and Safari/605.1.15 (S), on different devices, and WebGL versions, using our multi-frame rendering. Each measurement is the average refresh rate of 6 400 intermediate frames (excluding warmup and AA only), i.e., 100 converged multi-frames with $n_{\mathrm{MF}} = 64$. While 8 Hz+ is quasi-interactive, 60 Hz is targeted.

The performance difference between the aggregated and non-aggregated data sets with 448 322 and 500 071 nodes is that the former has extreme hierarchical depth and a higher number of inner nodes, making aggregation less effective. It can further be noticed that aggregation significantly negatively impacts smaller treemaps due to the additional draw calls needed and the renderer already being highly optimized.

These performance measurements serve as a rough orientation and can be subject to variability. The tests indicate that multi-frame rendering is highly responsive, with aggregation further enhancing performance in some cases. Although not included in the screenshots to ensure anonymity, all tests had dynamic labeling of inner nodes and a few leaf nodes enabled. The results indicate ample room for including additional visual variables on demand. Overall, our multi-frame rendering approach is highly responsive across various devices and browsers.

7 Conclusions and Future Work

The contents of this chapter are based on the following original publications:

W. Scheibel, J. Hartmann, **D. Limberger**, and J. Döllner. "Visualization of Tree-structured Data using Web Service Composition". In: *Springer VISIGRAPP - Extended and Revised Papers.* 2019 [L16]

C. Fiedler, W. Scheibel, **D. Limberger**, M. Trapp, and J. Döllner. "Survey on User Studies on the Effectiveness of Treemaps". In: *Proc. ACM VINCI.* 2020 [L17]

7.1 Conclusions

To conclude, we summarize the findings of this thesis's major chapters 3, 4, and 5, regarding their associated thesis statements on *expressiveness*, *scalability*, and *responsiveness*, respectively.

Conclusions regarding expressiveness $\mathcal{H}_E$. To increase the data mapping capabilities of 2D treemaps for the simultaneous, unambiguous display of data, we embedded 2D treemaps in a 3D attribute space and provided versatile visual variables. We explored visual variables for 3D-embedded treemaps and showed that sketchy contours and surface hatching can visually encode uncertainty, imprecision, and vagueness. Physically-based materials, such as rustiness, roughness, shininess, and glow, can emphasize activity and encode trends. We introduced in-situ templates for two-state mappings to display two attribute values simultaneously while visually differentiating original and comparative states. Additionally, we showcased that procedural textures can convey data changes through animated transitions, and secondary patterns enable their unambiguous reading direction. Lastly, value-added adaptations for 3D-embedded treemaps, such as height-based filtering using reference surfaces, extend the tools available for interactive exploration.

Conclusions regarding scalability $\mathcal{H}_S$. To enhance the readability of 3D-embedded treemaps, we presented a technique for aggregating nodes through dynamic degree-of-interest scoring and score propagation, along with the visual display of aggregates using aggregation operators for color and height. The addition of nesting level contouring, animated state transitions, and color weaving adheres to the established aggregation guidelines. All combined help to reduce or avoid visual clutter and facilitate interactive exploration of large datasets. We showcased dynamic labeling with descriptive label placement and discussed their hardware-accelerated rendering and usefulness for 3D-embedded map legends. Additionally, we showcased a partial 3D-embedding approach using a node-local tilt operator that allows users to toggle between the 2D and 3D-embedded display of inner-nodes.

Conclusions regarding responsiveness $\mathcal{H}_R$. To enable high-quality and responsive rendering of 3D-embedded treemaps, we explored rendering rectangles and cuboids using WebGL in conjunction with a progressive rendering approach. To ensure high-quality and responsive rendering, we described progressive sampling strategies, including multi-frame anti-aliasing, transparency, screen-space ambient occlusion, and depth of field. We, finally, detailed the use of our *treemap.ts* library to create and configure 3D-embedded treemaps for interactive, visual exploration of large tree-structured data in the browser.

7.2 Outlook and Challenges

The concepts and techniques presented and discussed in this work have limitations; some challenges have yet to be addressed, marked for future investigation, or, most likely, need a thorough evaluation. Instead of focusing on minor aspects, technical details, or low-hanging fruits, we suggest one primary idea for each main contributing section that we would like to explore more.

Chapter 3 | Evaluation Tooling A survey on the effectiveness of treemaps by Fiedler et al. [L17] recently reviewed 69 user studies related to treemaps: "Due to pitfalls and shortcomings in design, conduct, and reporting of the user studies, there is little that can be reliably derived or accepted as a generalized statement. Fundamental open questions include configuration, compatible tasks, use cases, and perceptional characteristics of treemaps."[L17] Ideas for appropriate toolings that guide experiments in information visualization have been presented [74]. We would like to see an increase in free and open tools for a more unified and streamlined evaluation of findings. For example, a benchmark for tree visualizations, methodologies [43] augmented through tools, and (more) public repositories for collectively gathering results could aid in evaluating visual variables and facilitate informed visualization design.

Chapter 4 | Interactable Labels We created and open-sourced hardware-accelerated typesetting, placement, and text rendering for virtual 3D environments. Even though UTF-8 is difficult, typesetting is limited, and per-glyph coloring and style mixing are not supported, we found ourselves craving to directly interact with the labels by selecting the text, changing it, and copying it, all within the 3D

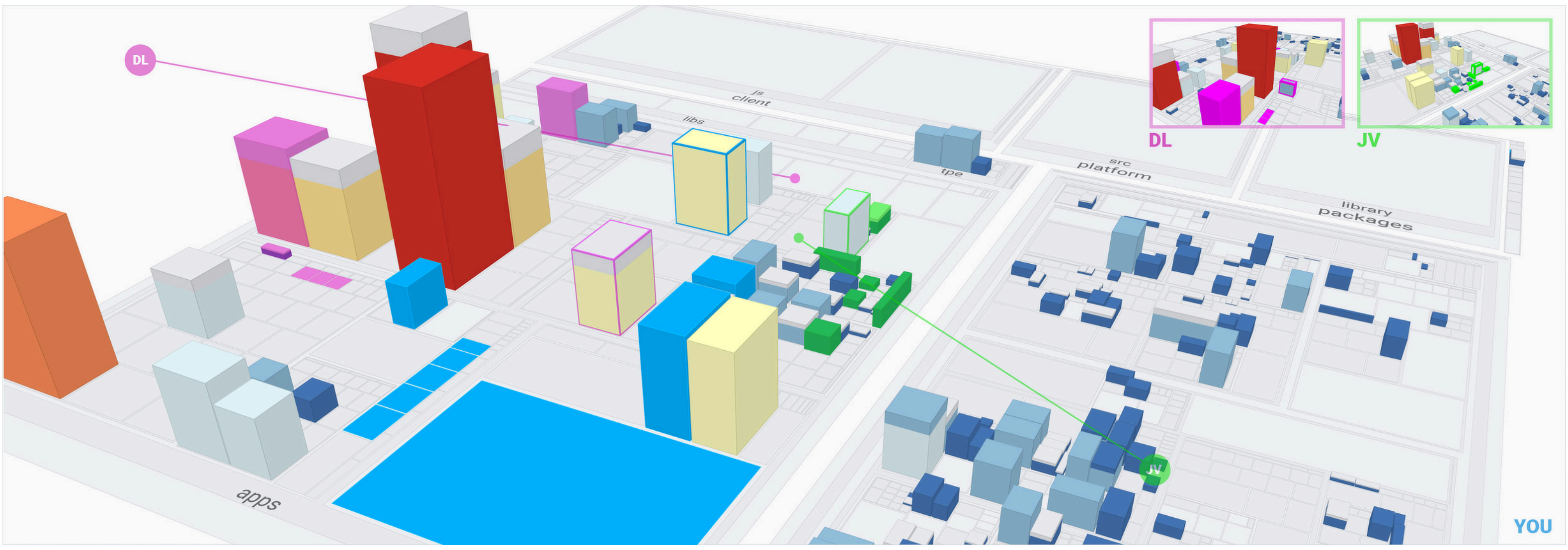


Figure 7.1: Illustration of a collaborative visual analytics space based on a 3D-embedded treemap. It shows the view of a user that collaborates with two other users (DL and JV), their views being overlayed (top-right), their virtual camera positions and look-at directions visualized, and unique colors are used for use associated selections and highlights.

scene. Augmenting 3D visualizations with essential, fundamental text editing capabilities would certainly enable novel visual analytics workflows.

Chapter 5 | Collaborative Exploration Sharing documents and working collaboratively and simultaneously is typical for many workloads today and was recently explored, e.g., for raster and vector images [17, 18]. When presenting interesting results or communicating exciting insights in visual analytics, we should be able to collaboratively explore within the same 3D-virtual environment instead of using screen sharing. We would like to see investigations on how to allow for awareness of the exploration of collaborators, what they interact with, how to adjust the visualization mapping, and collaboratively emphasize and share insights (Figure 7.1).

Chapter 6 | AI-Assisted Assembly We briefly outlined the most basic design aid for configuring 3D-embedded treemaps. We want to see assembly services similar to AI-based writing assistants reviewing spelling, grammar, punctuation, clarity, or even engagement and delivery. An AI-assisted visualization assembly service, e.g., specialized for *D3.js*,[7.1] would check the visualization mapping for common mistakes and make suggestions that support the task, e.g., promoting more appropriate color scales, proposing attribute transformations, or advising for additional visual variables for details on demand. This might help to make visualizations accessible that go beyond everyday diagrams.

[7.1] Mike Bostock. *D3.js – Data-Driven Documents*. d3js.org. 2012.

7.3 Closing Remarks

This work has demonstrated how hierarchical data can be visualized interactively using an advanced treemap approach. The embedding in 3D is of significant advantage, conceptually and technically, because it can improve both the design scope and the implementation. At the same time, it becomes clear that very precise considerations are required when it comes to the design of individual visualization features. Especially here, various specialized scientific contexts can provide valuable considerations and methods, such as cartographic visualization and real-time rendering. With the 3D-embedded treemaps presented here, it becomes clear that hierarchical data—one of the most important categories in the age of 'Big Data'—can be visualized in a scalable, readable, precise, and interactive way.

List of Publications

This thesis is based on the following original publications:

[L1] **Daniel Limberger**, Benjamin Wasty, Jonas Trümper, and Jürgen Döllner. "Interactive Software Maps for Web-based Source Code Analysis". In: *Proceedings of the 18th International Conference on 3D Web Technology*. Web3D '13. ACM, 2013, pp. 91–98. ISBN: 978-1-450321-33-4. DOI: 10.1145/2466533.2466550.

[L2] Hannes Würfel, Matthias Trapp, **Daniel Limberger**, and Jürgen Döllner. "Natural Phenomena as Metaphors for Visualization of Trend Data in Interactive Software Maps". In: *Computer Graphics and Visual Computing*. CGVC '15. The Eurographics Association, 2015, pp. 69–76. ISBN: 978-3-905674-94-1. DOI: 10.2312/cgvc.20151246.

[L3] **Daniel Limberger** and Jürgen Döllner. "Real-time Rendering of High-quality Effects using Multi-frame Sampling". In: *ACM SIGGRAPH 2016 Posters*. SIGGRAPH '16. ACM, 2016, 79:1–1. ISBN: 978-1-450343-71-8. DOI: 10.1145/2945078.2945157.

[L4] **Daniel Limberger**, Carolin Fiedler, Sebastian Hahn, Matthias Trapp, and Jürgen Döllner. "Evaluation of Sketchiness as a Visual Variable for 2.5 D Treemaps". In: *20th International Conference Information Visualisation*. IV '16. IEEE, 2016, pp. 183–189. ISBN: 978-1-467389-42-6. DOI: 10.1109/iV.2016.61.

[L5] **Daniel Limberger**, Willy Scheibel, Stefan Lemme, and Jürgen Döllner. "Dynamic 2.5D Treemaps using Declarative 3D on the Web". In: *Proceedings of the 21st International Conference on 3D Web Technology*. Web3D '16. ACM, 2016, pp. 33–36. ISBN: 978-1-450344-28-9. DOI: 10.1145/2945292.2945313.

[L6] **Daniel Limberger**, Karsten Tausche, Johannes Linke, and Jürgen Döllner. "Progressive Rendering using Multi-frame Sampling". In: *GPU Pro 7: Advanced Rendering Techniques* (2016), pp. 155–173. DOI: 10.1201/b21261.

[L7] **Daniel Limberger**, Marcel Pursche, Jan Klimke, and Jürgen Döllner. "Progressive High-quality Rendering for Interactive Information Cartography using WebGL". In: *Proceedings of the 22nd International Conference on 3D Web Technology*. Web3D '17. **Best Paper**. ACM, 2017, 8:1–4. ISBN: 978-1-450349-55-0. DOI: 10.1145/3055624.3075951.

[L8] **Daniel Limberger**, Willy Scheibel, Sebastian Hahn, and Jürgen Döllner. "Reducing Visual Complexity in Software Maps using Importance-based Aggregation of Nodes". In: *Proceedings of the 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*. IVAPP '17. SciTePress, 2017, pp. 176–185. ISBN: 978-9-897582-28-8. DOI: 10.5220/0006267501760185.

[L9] **Daniel Limberger**, Willy Scheibel, Matthias Trapp, and Jürgen Döllner. "Mixed-Projection Treemaps: A Novel Approach Mixing 2D and 2.5D Treemaps". In: *21st International Conference Information Visualisation*. IV '17. IEEE, 2017, pp. 164–169. ISBN: 978-1-538608-31-9. DOI: 10.1109/iV.2017.67.

[L10] **Daniel Limberger**. "Interactive, Adaptive Level-of-Detail in 2.5D Treemaps". U.S. pat. 9953443. Seerene GmbH. Apr. 24, 2018.

[L11] **Daniel Limberger**, Anne Gropler, Stefan Buschmann, Jürgen Döllner, and Benjamin Wasty. “OpenLL: an API for Dynamic 2D and 3D Labeling”. In: *22nd International Conference Information Visualisation*. IV ’18. IEEE, 2018, pp. 175–181. ISBN: 978-1-538672-02-0. DOI: 10.1109/iV.2018.00039.

[L12] **Daniel Limberger**, Karsten Tausche, Johannes Linke, and Jürgen Döllner. “Progressive Rendering using Multi-frame Sampling”. In: *GPU Pro 360 Guide to Rendering* (2018). Reprint of the GPU Pro 7 Article, 2016, pp. 537–553. DOI: 10.1201/9781351261524.

[L13] **Daniel Limberger**, Matthias Trapp, and Jürgen Döllner. “Interactive, Height-Based Filtering in 2.5D Treemaps”. In: *Proceedings of the 11th International Symposium on Visual Information Communication and Interaction*. VINCI ’18. ACM, 2018, pp. 49–55. ISBN: 978-1-450365-01-7. DOI: 10.1145/3231622.3231638.

[L14] **Daniel Limberger**, Willy Scheibel, Matthias Trapp, and Jürgen Döllner. “Advanced Visual Metaphors and Techniques for Software Maps”. In: *Proceedings of the 12th International Symposium on Visual Information Communication and Interaction*. VINCI ’19. ACM, 2019, 11:1–8. ISBN: 978-1-450376-26-6. DOI: 10.1145/3356422.3356444.

[L15] **Daniel Limberger**, Matthias Trapp, and Jürgen Döllner. “In-Situ Comparison for 2.5D Treemaps”. In: *Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*. IVAPP ’19. SciTePress, 2019, pp. 314–321. ISBN: 978-9-897583-54-4. DOI: 10.5220/0007576203140321.

[L16] Willy Scheibel, Judith Hartmann, **Daniel Limberger**, and Jürgen Döllner. “Visualization of Tree-structured Data using Web Service Composition”. In: *Computer Vision, Imaging and Computer Graphics Theory and Applications*. VISIGRAPP ’17. Springer International Publishing, 2019, pp. 227–252. ISBN: 978-3-030415-90-7. DOI: 10.1007/978-3-030-41590-7_10.

[L17] Carolin Fiedler, Willy Scheibel, **Daniel Limberger**, Matthias Trapp, and Jürgen Döllner. “Survey on User Studies on the Effectiveness of Treemaps”. In: *Proceedings of the 13th International Symposium on Visual Information Communication and Interaction*. VINCI ’20. ACM, 2020, 2:1–10. ISBN: 978-1-450387-50-7. DOI: 10.1145/3430036.3430054.

[L18] **Daniel Limberger**, Matthias Trapp, and Jürgen Döllner. “Depicting Uncertainty in 2.5D Treemaps”. In: *Proceedings of the 13th International Symposium on Visual Information Communication and Interaction*. VINCI ’20. ACM, 2020, 28:1–2. ISBN: 978-1-450387-50-7. DOI: 10.1145/3430036.3432753.

[L19] Willy Scheibel, **Daniel Limberger**, and Jürgen Döllner. “Survey of Treemap Layout Algorithms”. In: *Proceedings of the 13th International Symposium on Visual Information Communication and Interaction*. VINCI ’20. ACM, 2020, 1:1–9. ISBN: 978-1-450387-50-7. DOI: 10.1145/3430036.3430041.

[L20] Willy Scheibel, Matthias Trapp, **Daniel Limberger**, and Jürgen Döllner. “A Taxonomy of Treemap Visualization Techniques”. In: *Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*. IVAPP ’20. SciTePress, 2020, pp. 273–280. ISBN: 978-9-897584-02-2. DOI: 10.5220/0009153902730280.

[L21] **Daniel Limberger**, Willy Scheibel, Jan Dieken, and Jürgen Döllner. “Visualization of Data Changes in 2.5D Treemaps using Procedural Textures and Animated Transitions”. In: *Proceedings of the 14th International Symposium on Visual Information Communication and Interaction*. VINCI ’21. ACM, 2021, 21:1–5. ISBN: 978-1-450386-47-0. DOI: 10.1145/3481549.3481570.

[L22] **Daniel Limberger**, Willy Scheibel, Jan van Dieken, and Jürgen Döllner. “Procedural Texture Patterns for Encoding Changes in Color in 2.5D Treemap Visualizations”. In: *Journal of Visualization* (2022). DOI: 10.1007/s12650-022-00874-3.

[L23] **Daniel Limberger**, Willy Scheibel, Jürgen Döllner, and Matthias Trapp. “Visual Variables and Configuration of Software Maps”. In: *Journal of Visualization* (2022). DOI: 10.1007/s12650-022-00868-1.

List of Repositories

The following software systems, services, libraries, and tools have been developed or co-developed by the author as part of this thesis. For each entry, the year of the last modification (main branch) was used, and only the major contributors are referred to by name. Complete listings of contributors are available online as part of each repository. All repositories were last accessed and verified for availability on 22. March 2023 and are free and open source unless otherwise noted.

[R1] Stefan Buschmann, Willy Scheibel, **Daniel Limberger**, et al. *gloperate – A C++ Library for Defining and Controlling Modern GPU Rendering and Processing Operations.* gloperate.org. 2014 – 2023. Repository:

- github.com/cginternals/gloperate.

[R2] Carolin Fiedler, Willy Scheibel, **Daniel Limberger**, et al. *The Companion Website for the VINCI '20 publication "Survey on User Studies on the Effectiveness of Treemaps".* varg-dev.github.io/treemap-studies. 2018 – 2020. Repository:

- github.com/varg-dev/treemap-studies.

[R3] **Daniel Limberger**. *haeley-colors – Color Math and Color Scale Generation for Real-time Rendering.* github.com/halb3/colors. 2022 – 2023. Repository:

- github.com/halb3/colors.

[R4] **Daniel Limberger** et al. *webgl-operate – A Robust, Application-agnostic Rendering Framework using WebGL.* webgl-operate.org. 2016 – 2023. Repositories:

- github.com/cginternals/webgl-operate and
- gitlab.cginternals.com/libs/haeley.js | not public, predecessor.

[R5] **Daniel Limberger**, Anne Gropler, and Willy Scheibel. *OpenLL – Open Label Library.* openll.org. 2015 – 2018. Repositories:

- github.com/cginternals/openll and
- gitlab.hpi3d.de/treevis/glannotations | not public.

[R6] **Daniel Limberger** and Willy Scheibel. *Font Asset Generator – A Service for the OpenLL Asset Generator.* fonts.varg.dev. 2021. Repositories:

- github.com/varg-dev/font-asset-service, and
- github.com/varg-dev/font-asset-gui.

[R7] **Daniel Limberger** and Willy Scheibel. *Treemap Designer – A Configurator for the Assembly of Treemaps.* treemap.de/treemap-designer.html. 2019. Repositories:

- https://github.com/cgcostume/[...]/treemap-designer.ts | not public.

[R8] **Daniel Limberger**, Willy Scheibel, Stefan Lemme, et al. *2.5D Treemaps using Declarative 3D.* cgcostume.github.io/web3d-treemaps. 2016. Repository:

- github.com/cgcostume/web3d-treemaps.

[R9] **Daniel Limberger**, Willy Scheibel, Roland Lux, et al. *glbinding – A C++ Binding for the OpenGL API.* glbinding.org. 2014 – 2023. Repository:

- github.com/cginternals/glbinding.

[R10] **Daniel Limberger**, Willy Scheibel, Philipp Otto, et al. *glkernel – Sampling Utilities for OpenGL*. kernel.varg.dev/docs. 2015 – 2023. Repositories:

- github.com/cginternals/glkernel and
- github.com/cginternals/glkernel-service.

[R11] **Daniel Limberger** and Maximilian Söchting. *A Multi-frame Sampling Viewer for 3D Scenes using three.js*. cgcostume.github.io/mfsv. 2016. Repositories:

- github.com/cgcostume/mfsv and
- github.com/cgcostume/multiframesampling (C++ and gloperate).

[R12] **Daniel Limberger** and Scheibel Willy. *treemaps.ts – A Library for Rendering 3D-embedded Treemaps using WebGL*. swmap.org. 2016 – 2021. Repositories:

- github.com/cgcostume/treemap | not public.

[R13] Jaqueline Pollak, Carsten Walther, Pascal Lange, **Daniel Limberger**, and Willy Scheibel. *File System Viewer using 2.5D Treemaps*. 2015 – 2016. Repositories:

- gitlab.hpi3d.de/treevis/mp2016d1 | not public and
- gitlab.hpi3d.de/treevis/filesystemviewer | not public.

[R14] Willy Scheibel, **Daniel Limberger**, et al. *arboretum – A Tree Visualization Framework for Rapid Prototyping*. 2013 – 2019. Repositories:

- gitlab.hpi3d.de/treevis/arboretum | not public,
- gitlab.hpi3d.de/treevis/bp2014d2 | not public,
- gitlab.hpi3d.de/treevis/bp2015d2 | not public, and
- gitlab.hpi3d.de/treevis/arboretum-viewer | not public.

[R15] Willy Scheibel, Roland Lux, **Daniel Limberger**, et al. *globjects – A Strict C++ Wrapper of OpenGL Objects*. globjects.org. 2013 – 2023. Repository:

- github.com/cginternals/globjects.

Bibliography

[1] Fernando Brito Abreu and Rogério Carapuça. "Object-Oriented Software Engineering: Measuring and Controlling the Development Process". In: *Proceedings of the 4th International Conference on Software Quality*. Vol. 186. 1994, pp. 1–8.

[2] Tomas Akenine-Möller, Eric Heines, and Naty Hoffman. *Real-Time Rendering, Fourth Edition*. CRC Press, 2018. ISBN: 978-1-351816-15-1.

[3] Nicolli S.R. Alves, Leilane F. Ribeiro, Vivyane Caires, Thiago S. Mendes, and Rodrigo O. Spínola. "Towards an Ontology of Terms on Technical Debt". In: *2014 Sixth International Workshop on Managing Technical Debt*. MTD '14. 2014, pp. 1–7. DOI: 10.1109/MTD.2014.9.

[4] Keith Andrews. "Case Study. Visualising Cyberspace: Information Visualisation in the Harmony Internet Browser". In: *Proceedings of Visualization 1995 Conference*. InfoVis '95. IEEE, 1995, pp. 97–104. DOI: 10.1109/INFVIS.1995.528692.

[5] Keith Andrews. "Visual Exploration of Large Hierarchies with Information Pyramids". In: *Proceedings of the 6th International Conference on Information Visualisation*. IV '02. IEEE, 2002, pp. 793–798. DOI: 10.1109/IV.2002.1028871.

[6] Keith Andrews, Josef Wolte, and Michael Pichler. "Information Pyramids®: A New Approach to Visualizing Large Hierarchies". In: *Proc. IEEE Visualization*. Vol. 97. IEEE Vis '97. 1997, pp. 49–52.

[7] Daniel Archambault, Helen Purchase, and Bruno Pinaud. "Animation, Small Multiples, and the Effect of Mental Map Preservation in Dynamic Graphs". In: *Transactions on Visualization and Computer Graphics* 17.4 (2011), pp. 539–552. ISSN: 1077-2626. DOI: 10.1109/TVCG.2010.78.

[8] Daniel Atzberger, Tim Cech, Merlin de la Haye, Maximilian Söchting, Willy Scheibel, **Daniel Limberger**, and Jürgen Döllner. "Software Forest: A Visualization of Semantic Similarities in Source Code using a Tree Metaphor". In: *Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*. IVAPP '21. SciTePress, 2021, pp. 112–122. ISBN: 978-9-897584-88-6. DOI: 10.5220/0010267601120122.

[9] David Auber, C. Huet, A. Lambert, B. Renoust, A. Sallaberry, and A. Saulnier. "GosperMap: Using a Gosper Curve for Laying Out Hierarchical Data". In: *IEEE Trans Vis Comput Graph*. TVCG '13 19.11 (2013), pp. 1820–1832. ISSN: 1077-2626. DOI: 10.1109/TVCG.2013.91.

[10] Ketan Babaria. *Using Treemaps to Visualize Gene Ontologies*. Tech. rep. Human Computer Interaction Lab and Institute for Systems Research, University of Maryland, College Park, MD USA, 2001.

[11] Jan Balata, Ladislav Cmolik, and Zdenek Mikovec. "On the Selection of 2D Objects Using External Labeling". In: *Proceedings of the 32nd Annual ACM Conference on Human Factors in Computing Systems*. CHI '14. ACM, 2014, pp. 2255–2258. ISBN: 978-1-450324-73-1. DOI: 10.1145/2556288.2557288.

[12] Gergő Balogh, Attila Szabolics, and Árpád Beszédes. "CodeMetropolis: Eclipse over the City of Source Code". In: *2015 IEEE 15th International Working Conference on Source Code Analysis and Manipulation*. SCAM '15. 2015, pp. 271–276. DOI: 10.1109/SCAM.2015.7335425.

[13] M. Balzer and Oliver Deussen. "Voronoi Treemaps". In: *Proc. IEEE Symp. on Information Visualization*. InfoVis '05. 2005, pp. 49–56. DOI: 10.1109/INFVIS.2005.1532128.

[14] Michael Balzer and Oliver Deussen. "Hierarchy Based 3D Visualization of Large Software Structures". In: *Proceedings of the Conference on Visualization '04*. VIS '04. IEEE Computer Society, 2004, p. 598.4. ISBN: 0-7803-8788-0. DOI: 10.1109/VISUAL.2004.39.

[15] Michael Balzer and Oliver Deussen. "Level-of-detail visualization of clustered graph layouts". In: *6th International Asia-Pacific Symposium on Visualization*. APVIS '07. IEEE, 2007, pp. 133–140. DOI: 10.1109/APVIS.2007.329288.

[16] Michael Balzer, Andreas Noack, Oliver Deussen, and Claus Lewerentz. "Software Landscapes: Visualizing the Structure of Large Software Systems". In: *Eurographics / IEEE VGTC Symposium on Visualization*. The Eurographics Association, 2004. ISBN: 3-905673-07-X. DOI: 10.2312/VisSym/VisSym04/261-266.

[17] Ulrike Bath, Sumit Shekhar, Jürgen Döllner, and Matthias Trapp. "COLiER: Collaborative Editing of Raster Images". In: *International Conference on Cyberworlds*. CW '21. 2021, pp. 33–40. DOI: 10.1109/CW52790.2021.00013.

[18] Ulrike Bath, Sumit Shekhar, Julian Egbert, Julian Schmidt, Amir Semmo, Jürgen Döllner, and Matthias Trapp. "CERVI: Collaborative Editing of Raster and Vector Images". In: *Springer The Visual Computer* 38.12 (2022), pp. 4057–4070. DOI: 10.1007/s00371-022-02522-1.

[19] Louis Bavoil, Miguel Sainz, and Rouslan Dimitrov. "Image-space Horizon-based Ambient Occlusion". In: *ACM SIGGRAPH 2008 Talks*. 2008, 22:1–1.

[20] Benjamin B. Bederson, Ben Shneiderman, and Martin Wattenberg. "Ordered and Quantum Treemaps: Making Effective Use of 2D Space to Display Hierarchies". In: *ACM Transactions on Graphics* 21.4 (2002), pp. 833–854. ISSN: 0730-0301. DOI: 10.1145/571647.571649.

[21] Michael Behrisch et al. "Quality Metrics for Information Visualization". In: *EG Computer Graphics Forum* 37.3 (2018), pp. 625–662. DOI: 10.1111/cgf.13446.

[22] Omar Benomar, Houari A. Sahraoui, and Pierre Poulin. "Visualizing software dynamicities with heat maps". In: *2013 First IEEE Working Conference on Software Visualization*. VISSOFT '13. 2013, pp. 1–10. DOI: 10.1109/VISSOFT.2013.6650524.

[23] J"org Bernhardt, Stefan Funke, Michael Hecker, and Juliane Siebourg. "Visualizing Gene Expression Data via Voronoi Treemaps". In: *2009 Sixth International Symposium on Voronoi Diagrams*. 2009, pp. 233–241. DOI: 10.1109/ISVD.2009.33.

[24] Jacques Bertin. *Sémiologie graphique*. 1967. DOI: 10.1002/zamm.19690490917.

[25] Jacques Bertin. *Semiology of Graphics*. University of Wisconsin Press, 1983. ISBN: 978-1-589482-61-6. DOI: 10.5555/1095597.

[26] Jacques Bertin. *Semiology of Graphics: Diagrams, Networks, Maps*. Esri Press, 2010. ISBN: 978-1-589-48261-6.

[27] Donald Bertucci, Md Montaser Hamid, Yashwanthi Anand, Anita Ruangrotsakun, Delyar Tabatabai, Melissa Perez, and Minsuk Kahng. "Visual Exploration of Large-Scale Image Datasets for Machine Learning with Treemaps". In: *arXiv* abs/2205.06935 (2022). preprint.

[28] Joseph Bethge, Sebastian Hahn, and Jürgen Döllner. "Improving Layout Quality by Mixing Treemap-Layouts Based on Data-Change Characteristics". In: *Vision, Modeling & Visualization*. The Eurographics Association, 2017. ISBN: 978-3-038680-49-9. DOI: 10.2312/vmv.20171261.

[29] Thomas Bladh, David A Carr, and Jeremiah Scholl. "Extending tree-maps to three dimensions: A comparative study". In: *Asia-Pacific Conference on Computer Human Interaction*. APCHI '04. Springer. 2004, pp. 50–59. DOI: 10.1007/978-3-540-27795-8_6.

[30] Thomas Bladh, David A. Carr, and Matjaž Kljun. “The Effect of Animated Transitions on User Navigation in 3D Tree-maps”. In: *9th International Conference on Information Visualisation*. IV ’05. 2005, pp. 297–305. DOI: 10.1109/IV.2005.122.

[31] Renaud Blanch and Eric Lecolinet. “Browsing Zoomable Treemaps: Structure-Aware Multi-Scale Navigation Techniques”. In: *IEEE Trans Vis Comput Graph*. TVCG ’07 13.6 (2007), pp. 1248–1253. ISSN: 1077-2626. DOI: 10.1109/TVCG.2007.70540.

[32] Sandro Boccuzzo and Harald Gall. “CocoViz: Towards Cognitive Software Visualizations”. In: *Proceedings of the 4th International Workshop on Visualizing Software for Understanding and Analysis*. VISSOFT ’07. IEEE, 2007, pp. 72–79. DOI: 10.1109/VISSOF.2007.4290703.

[33] Johannes Bohnet and Jürgen Döllner. “Monitoring Code Quality and Development Activity by Software Maps”. In: *Proceedings of the 2nd Workshop on Managing Technical Debt*. MTD ’11. ACM, 2011, pp. 9–16. ISBN: 978-1-450305-86-0. DOI: 10.1145/1985362.1985365.

[34] Rita Borgo, Alfie Abdul-Rahman, Farhan Mohamed, Philip W. Grant, Irene Reppa, Luciano Floridi, and Min Chen. “An Empirical Study on Using Visual Embellishments in Visualization”. In: *IEEE Trans Vis Comput Graph* 18.12 (2012), pp. 2759–2768. ISSN: 1077-2626. DOI: 10.1109/TVCG.2012.197.

[35] Nadia Boukhelifa, Anastasia Bezerianos, Tobias Isenberg, and Jean-Daniel Fekete. “Evaluating Sketchiness as a Visual Variable for the Depiction of Qualitative Uncertainty”. In: *IEEE Trans Vis Comput Graph* 18.12 (2012), pp. 2769–2778. ISSN: 1077-2626. DOI: 10.1109/TVCG.2012.220.

[36] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. “LOF: Identifying Density-Based Local Outliers”. In: *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’00. ACM, 2000, pp. 93–104. ISBN: 1-581132-17-4. DOI: 10.1145/342009.335388.

[37] Cynthia A. Brewer. “Chapter 7 - Color Use Guidelines for Mapping and Visualization”. In: *Visualization in Modern Cartography*. Vol. 2. Modern Cartography Series. Academic Press, 1994, pp. 123–147. DOI: 10.1016/B978-0-08-042415-6.50014-4.

[38] Nanette Brown et al. “Managing Technical Debt in Software-Reliant Systems”. In: *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*. FoSER ’10. ACM, 2010, pp. 47–52. ISBN: 978-1-450304-27-6. DOI: 10.1145/1882362.1882373.

[39] Jeffrey Browne, Bongshin Lee, Sheelagh Carpendale, Nathalie Riche, and Timothy Sherwood. “Data Analysis on Interactive Whiteboards Through Sketch-based Interaction”. In: *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*. ITS ’11. ACM, 2011, pp. 154–157. ISBN: 978-1-450308-71-7. DOI: 10.1145/2076354.2076383.

[40] Mark Bruls, Kees Huizing, and Jarke J. van Wijk. “Squarified Treemaps”. In: *Data Visualization 2000: Proceedings of the Joint EUROGRAPHICS and IEEE TCVG Symposium on Visualization*. 2000, pp. 33–42. ISBN: 978-3-709167-83-0. DOI: 10.1007/978-3-7091-6783-0_4.

[41] Mike Bukowski, Padraic Hennessy, Brian Osman, and Morgan McGuire. “The Skylanders SWAP Force Depth-of-Field Shader”. In: *GPU Pro 4: Advanced Rendering Techniques*. An A K Peters Book. Taylor & Francis, 2013, pp. 175–184. ISBN: 978-1-466567-43-6.

[42] Stefan Buschmann, Matthias Trapp, and Jürgen Döllner. “Real-Time Animated Visualization of Massive Air-Traffic Trajectories”. In: *2014 International Conference on Cyberworlds*. 2014, pp. 174–181. DOI: 10.1109/CW.2014.32.

[43] Alma Cantu, Olivier Grisvard, Thierry Duval, and Gilles Coppin. “Identifying the Relationships Between the Visualization Context and Representation Components to Enable Recommendations for Designing New Visualizations”. In: *21st International Conference Information Visualisation*. IV ’17. IEEE Computer Society, 2017, pp. 20–28. DOI: 10.1109/iV.2017.55.

[44] M. Sheelagh T. Carpendale. *Considering Visual Variables as a Basis for Information Visualisation*. Tech. rep. University of Calgary, 2003. doi: 10.11575/PRISM/10182.

[45] Yi-Jun Chang and Hsu-Chun Yen. "Constrained floorplans in 2D and 3D". In: *Theoretical Computer Science* 607 (2015), pp. 320–336. issn: 0304-3975. doi: 10.1016/j.tcs.2015.07.063.

[46] Abon Chaudhuri and Han-Wei Shen. "A Self-adaptive Treemap-based Technique for Visualizing Hierarchical Data in 3D". In: *IEEE Pacific Visualization Symposium*. Vol. 00. IEEE Computer Society, 2009, pp. 105–112. doi: 10.1109/PACIFICVIS.2009.4906844.

[47] Yi Chen, Xiaomin Du, and Xiaoru Yuan. "Ordered Small Multiple Treemaps for Visualizing Time-Varying Hierarchical Pesticide Residue Data". In: *Springer The Visual Computer* 33.6 (2017), pp. 1073–1084. issn: 1432-2315. doi: 10.1007/s00371-017-1373-x.

[48] Ed H. Chi. "A Taxonomy of Visualization Techniques Using the Data State Reference Model". In: *Proc. IEEE Symp. on Information Visualization*. InfoVis '00. IEEE Computer Society, 2000. isbn: 0769508049. doi: 10.5555/857190.857691.

[49] Shyam R. Chidamber and Chris F. Kemerer. "A Metrics Suite for Object Oriented Design". In: *IEEE Trans. Softw. Eng.* 20.6 (1994), pp. 476–493. issn: 0098-5589. doi: 10.1109/32.295895.

[50] Junghong Choi, Oh-hyun Kwon, and Kyungwon Lee. "Strata Treemaps". In: *ACM SIGGRAPH 2011 Posters*. SIGGRAPH '11. ACM, 2011, 87:1–87:1. isbn: 978-1-450309-71-4. doi: 10.1145/2037715.2037813.

[51] Jon Christensen, Joe Marks, and Stuart Shieber. "An Empirical Study of Algorithms for Point-feature Label Placement". In: *ACM Trans. Graph.* 14.3 (1995), pp. 203–232. doi: 10.1145/212332.212334.

[52] Mei C. Chuah. "Dynamic Aggregation with Circular Visual Designs". In: *Proc. IEEE Symp. on Information Visualization*. InfoVis '98. 1998, pp. 35–43. isbn: 0-818690-93-3. doi: 10.1109/INFVIS.1998.729557.

[53] E. Clarkson, J. Foley, and K. Desai. "ResultMaps: Visualization for Search Interfaces". In: *IEEE Trans Vis Comput Graph*. TVCG '09 15 (2009), pp. 1057–1064. issn: 1077-2626. doi: 10.1109/TVCG.2009.176.

[54] Don Coleman, Dan Ash, Bruce Lowther, and Paul Oman. "Using Metrics to Evaluate Software System Maintainability". In: *Computer* 27.8 (1994), pp. 44–49. issn: 0018-9162. doi: 10.1109/2.303623.

[55] Alexandre Coninx, Georges-Pierre Bonneau, Jacques Droulez, and Guillaume Thibault. "Visualization of Uncertain Scalar Data Fields using Color Scales and Perceptually Adapted Noise". In: *Proc. ACM SIGGRAPH Symposium on Applied Perception in Graphics and Visualization*. APGV '11. ACM, 2011, pp. 59–66. isbn: 978-1-450308-89-2. doi: 10.1145/2077451.2077462.

[56] Michael Correll, Dominik Moritz, and Jeffrey Heer. "Value-Suppressing Uncertainty Palettes". In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. CHI '18. ACM, 2018, pp. 1–11. isbn: 978-1-450356-20-6. doi: 10.1145/3173574.3174216.

[57] Christoph Csallner, Marcus Handte, Othmar Lehmann, and John Stasko. "FundExplorer: Supporting the Diversification of Mutual Fund Portfolios using Context Treemaps". In: *Proc. IEEE Symp. on Information Visualization*. InfoVis '03. 2003, pp. 203–208. doi: 10.1109/INFVIS.2003.1249027.

[58] Qingguang Cui, Matthew Ward, Elke Rundensteiner, and Jing Yang. "Measuring Data Abstraction Quality in Multiresolution Visualizations". In: *IEEE Trans Vis Comput Graph* 12.5 (2006), pp. 709–716. doi: 10.1109/TVCG.2006.161.

[59] Veronika Dashuber and Michael Philippsen. "Trace Visualization within the Software City Metaphor: A Controlled Experiment on Program Comprehension". In: *Proceedings of the 2021 Working Conference on Software Visualization*. VISSOFT '21. IEEE, 2021, pp. 55–64. doi: 10.1109/VISSOFT52517.2021.00015.

[60] Will Dobbie. "GPU text rendering with vector textures". In: (2016). wdobbie.com/post/gpu-text-rendering-with-vector-textures.

[61] Jürgen Döllner and Maike Walther. “Real-Time Expressive Rendering of City Models”. In: *2013 17th International Conference on Information Visualisation*. IEEE Computer Society, 2003, pp. 245–250. DOI: 10.1109/IV.2003.1217986.

[62] S. Dübel, M. Röhlig, Heidrun Schumann, and Matthias Trapp. “2D and 3D presentation of spatial data: A systematic review”. In: *2014 IEEE VIS International Workshop on 3DVis (3DVis)*. 2014, pp. 11–18. DOI: 10.1109/3DVis.2014.7160094.

[63] Steve Dübel, Martin Röhlig, Christian Tominski, and Heidrun Schumann. “Visualizing 3D Terrain, Geo-Spatial Data, and Uncertainty”. In: *Informatics* 4 (2017). ISSN: 2227-9709. DOI: 10.3390/informatics4010006.

[64] Charles Dupin. “Carte figurative de l’instruction populaire de la France, pl. I.” In: 2 (1826). gallica.bnf.fr/ark:/12148/btv1b530830640, Accessed via Bibliothèque national de France on 22. March 2023.

[65] Geoffrey Ellis and Alan Dix. “A Taxonomy of Clutter Reduction for Information Visualisation”. In: *IEEE Trans Vis Comput Graph* 13.6 (2007), pp. 1216–1223. ISSN: 1077-2626. DOI: 10.1109/TVCG.2007.70535.

[66] Niklas Elmqvist and Jean-Daniel Fekete. “Hierarchical Aggregation for Information Visualization: Overview, Techniques, and Design Guidelines”. In: *IEEE Trans Vis Comput Graph* 16.3 (2010), pp. 439–454. ISSN: 1077-2626. DOI: 10.1109/TVCG.2009.84.

[67] Niklas Elmqvist and P. Tsigas. “A Taxonomy of 3D Occlusion Management for Visualization”. In: *IEEE Trans Vis Comput Graph*. TVCG ’08 14 (2008), pp. 1095–1109. ISSN: 1077-2626. DOI: 10.1109/TVCG.2008.59.

[68] Eric Enderton, Erik Sintorn, Peter Shirley, and David Luebke. “Stochastic Transparency”. In: *Proc. ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D ’10. ACM, 2010, pp. 157–164. ISBN: 978-1-605589-39-8. DOI: 10.1145/1730804.1730830.

[69] Eric Enderton, Erik Sintorn, Peter Shirley, and David Luebke. “Stochastic Transparency”. In: *Proc. of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D ’10. ACM, 2010, pp. 157–164. DOI: 10.1145/1730804.1730830.

[70] John Estdale. “Delaying Maintenance Can Prove Fatal”. In: *Software Quality Management XXVII: International Experiences and Initiatives in IT Quality Managementy*. Software Quality Management. Solent University, 2019, pp. 95–106. ISBN: 978-1-999654-92-4.

[71] Jean-Daniel Fekete and Catherine Plaisant. “Excentric Labeling: Dynamic Neighborhood Labeling for Data Visualization”. In: *Proceedings ACM SIGCHI Conference on Human Factors in Computing Systems*. CHI ’99. ACM, 1999, pp. 512–519. ISBN: 0-201485-59-1. DOI: 10.1145/302979.303148.

[72] Jean-Daniel Fekete and Catherine Plaisant. “Interactive Information Visualization of a Million Items”. In: *Symposium on Information Visualization*. InfoVis ’02. IEEE, 2002, pp. 117–124. ISBN: 978-0-769517-51-3. DOI: 10.1109/INFVIS.2002.1173156.

[73] Cong Feng, Minglun Gong, Oliver Deussen, and Hui Huang. “Treemapping via Balanced Partitioning”. In: *Proceedings of the International Conference on Computational Visual Media*. CVM ’19. 2019. DOI: 10.3724/SP.J.2096-5796.21.00040.

[74] Carolin Fiedler. “Development of a Framework for Controlled Experiments in Information Visualisation”. Masters’s Thesis. Hasso Plattner Institute, University of Potsdam, 2018.

[75] Florian Fittkau, E. Koppenhagen, and W. Hasselbring. “Research Perspective on Supporting Software Engineering via Physical 3D Models”. In: *IEEE 3rd Working Conference on Software Visualization*. VISSOFT ’15. 2015, pp. 125–129. DOI: 10.1109/VISSOFT.2015.7332422.

[76] Florian Fittkau, Alexander Krause, and Wilhelm Hasselbring. “Exploring software cities in virtual reality”. In: *2015 IEEE 3rd Working Conference on Software Visualization*. VISSOFT ’15. 2015, pp. 130–134. DOI: 10.1109/VISSOFT.2015.7332423.

[77] Florian Fittkau, Alexander Krause, and Wilhelm Hasselbring. "Hierarchical Software Landscape Visualization for System Comprehension: A Controlled Experiment". In: *IEEE 3rd Working Conference on Software Visualization*. VISSOFT 15. 2015, pp. 36–45. DOI: 10.1109/VISSOFT.2015.7332413.

[78] Michael Friendly. "A Brief History of the Mosaic Display". In: *Journal of Computational and Graphical Statistics* 11.1 (2002), pp. 89–107. DOI: 10.1198/106186002317\-375631.

[79] Sarah F. Frisken, Ronald N. Perry, Alyn P. Rockwood, and Thouis R. Jones. "Adaptively Sampled Distance Fields: A General Representation of Shape for Computer Graphics". In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '00. ACM Press/Addison-Wesley Publishing Co., 2000, pp. 249–254. ISBN: 1-581132-08-5. DOI: 10.1145/344779.344899.

[80] George W. Furnas. "Generalized Fisheye Views". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '86. ACM, 1986, pp. 16–23. ISBN: 0-897911-80-6. DOI: 10.1145/22627.22342.

[81] Simone Garlandini and Sara Fabrikant. "Evaluating the Effectiveness and Efficiency of Visual Variables for Geographic Information Visualization". In: 2009, pp. 195–211. ISBN: 978-3-642038-31-0. DOI: 10.1007/978-3-642-03832-7_12.

[82] Simon Garnier, Noam Ross, Bob Rudis, Marco Sciaini, Antônio Pedro Camargo, and Cédric Scherer. *viridis(Lite) - Colorblind-Friendly Color Maps for R*. sjmgarnier.github.io/viridis/. 2021. DOI: 10.5281/zenodo.4679424.

[83] Mohammad Ghoniem, Maël Cornil, Bertjan Broeksema, Mickaël Stefas, and Benoît Otjacques. "Weighted Maps: Treemap Visualization of Geolocated Quantitative Data". In: *Proc. International Society for Optical Engineering*. Vol. 9397. 2015. DOI: 10.1117/12.2079420. URL: http://dx.doi.org/10.1117/12.2079420.

[84] Mark Giereth, Harald Bosch, and Thomas Ertl. "A 3D Treemap Approach for Analyzing the Classificatory Distribution in Patent Portfolios". In: *Proc. IEEE Symp. on Visual Analytics Science and Technology*. VAST '08. 2008, pp. 189–190. DOI: 10.1109/VAST.2008.4677380.

[85] Michael Gleicher, Danielle Albers, Rick Walker, Ilir Jusufi, Charles D. Hansen, and Jonathan C. Roberts. "Visual Comparison for Information Visualization". In: *Information Visualization* 10.4 (2011), pp. 289–309. ISSN: 1473-8716. DOI: 10.1177/1473871611416549.

[86] Alison Gopnik and Clark Glymour. "Causal Maps and Bayes Nets: A Cognitive and Computational Account of Theory-formation". In: 2002, pp. 117–132. ISBN: 978-0-521812-29-0. DOI: 10.1017/CBO9780511613517.007.

[87] Jochen Görtler, Christoph Schulz, Daniel Weiskopf, and Oliver Deussen. "Bubble Treemaps for Uncertainty Visualization". In: *IEEE Trans Vis Comput Graph*. TVCG '17 24.1 (2018), pp. 719–728. DOI: 10.1109/TVCG.2017.2743959.

[88] Chris Green. "Improved Alpha-tested Magnification for Vector Textures and Special Effects". In: *ACM SIGGRAPH 2007 Courses*. SIGGRAPH '07. ACM, 2007, pp. 9–18. ISBN: 978-1-450318-23-5. DOI: 10.1145/1281500.1281665.

[89] Saul Greenberg, Sheelagh Carpendale, Nicolai Marquardt, and Bill Buxton. *Sketching User Experiences: The Workbook*. Morgan Kaufmann Publishers Inc., 2011. ISBN: 978-0-123819-59-8. DOI: 10.5555/2208198.

[90] Brendan Gregg. "The Flame Graph". In: *Communications* 59.6 (2016), pp. 48–57. ISSN: 0001-0782. DOI: 10.1145/2909476.

[91] John A. Guerra-Gómez, Cati Boulanger, Sanjay Kairam, and David A. Shamma. "Identifying Best Practices for Visualizing Photo Statistics and Galleries Using Treemaps". In: *Proceedings of the International Working Conference on Advanced Visual Interfaces*. AVI '16. ACM, 2016, pp. 60–63. ISBN: 978-1-450341-31-8. DOI: 10.1145/2909132.2909280.

[92] John Alexis Guerra-Gómez, M. L. Pack, Catherine Plaisant, and Ben Shneiderman. "Visualizing Change over Time Using Dynamic Hierarchies: TreeVersity2 and the StemView". In: *IEEE Trans Vis Comput Graph*. TVCG '13 19 (2013), pp. 2566–2575. ISSN: 1077-2626. DOI: 10.1109/TVCG.2013.231.

[93] Stefan Gustavson. “Procedural Textures in GLSL”. In: *OpenGL Insights*. CRC Press, 2012, pp. 105–120. ISBN: 978-1-439893-76-0.

[94] Robert B Haber and David A McNabb. “Visualization idioms: A conceptual model for scientific visualization systems”. In: *Visualization in Scientific Computing* 74 (1990), pp. 74–93.

[95] Paul Haeberli and Kurt Akeley. “The Accumulation Buffer: Hardware Support for High-Quality Rendering”. In: *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’90. ACM, 1990, pp. 309–318. ISBN: 0-897913-44-2. DOI: 10.1145/97879.97913.

[96] Haleh Hagh-Shenas, Sunghee Kim, Victoria Interrante, and Christopher G. Healey. “Weaving Versus Blending: a Quantitative Assessment of the Information Carrying Capacities of Two Alternative Methods for Conveying Multivariate Data with Color”. In: *Transactions on Visualization and Computer Graphics* 13.6 (2007), pp. 1270–1277. DOI: 10.1109/TVCG.2007.70623.

[97] Sebastian Hahn, Joseph Bethge, and Jürgen Döllner. “Relative Direction Change - A Topology-based Metric for Layout Stability in Treemaps”. In: *Proceedings of the 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 3: IVAPP, (VISIGRAPP 2017)*. INSTICC. SciTePress, 2017, pp. 88–95. ISBN: 978-9-897582-28-8. DOI: 10.5220/0006117500880095.

[98] Sebastian Hahn and Jürgen Döllner. “Hybrid-Treemap Layouting”. In: *EuroVis 2017 - Short Papers*. EuroVis ’17. The Eurographics Association, 2017, pp. 79–83. ISBN: 978-3-038680-43-7. DOI: 10.2312/eurovisshort.20171137.

[99] Maurice H. Halstead. *Elements of Software Science (Operating and Programming Systems Series)*. Elsevier Science Inc., 1977. DOI: 10.5555/540137.

[100] Frank van Ham and Jarke J. van Wijk. “Beamtrees: Compact Visualization of Large Hierarchies”. In: *Palgrave Information Visualization* 2.1 (2003), pp. 31–39. DOI: 10.1057/palgrave.ivs.9500036.

[101] M. C. Hao, Umeshwar Dayal, Daniel A. Keim, and Tobias Schreck. “Importance-driven Visualization Layouts for Large Time Series Data”. In: *Proc. IEEE Symp. on Information Visualization*. 2005, pp. 203–210. DOI: 10.1109/INFVIS.2005.1532148.

[102] Ming C. Hao, Umeshwar Dayal, Daniel A. Keim, and Tobias Schreck. “Multi-Resolution Techniques for Visual Exploration of Large Time-Series Data”. In: *Eurographics/IEEE VGTC Symposium on Visualization*. EUROVIS ’07. 2007, pp. 27–34. DOI: 10.2312/VisSym/EuroVis07/027-034.

[103] Simon Harper, Eleni Michailidou, and Robert Stevens. “Toward a Definition of Visual Complexity As an Implicit Measure of Cognitive Load”. In: *ACM Trans. Appl. Percept.* 6.2 (2009), 10:1–18. ISSN: 1544-3558. DOI: 10.1145/1498700.1498704.

[104] Mark Harrower and Cynthia A. Brewer. “ColorBrewer.org: An Online Tool for Selecting Colour Schemes for Maps”. In: *Taylor & Francis The Cartographic Journal* 40.1 (2003), pp. 27–37. DOI: 10.1179/000870403235002042.

[105] Helwig Hauser and Heidrun Schumann. “Visualization Pipeline”. In: *Encyclopedia of Database Systems*. Springer US, 2009, pp. 3414–3416. ISBN: 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_1133.

[106] Christopher G. Healey and James T. Enns. “Building Perceptual Textures to Visualize Multidimensional Datasets”. In: *Proc. of VIS*. VIS ’98. IEEE Computer Society Press, 1998, pp. 111–118. DOI: 10.1109/VISUAL.1998.745292.

[107] Christopher G. Healey and James T. Enns. “Large Datasets at a Glance: Combining Textures and Colors in Scientific Visualization”. In: *IEEE TVCG*. TVCG ’99 5.2 (1999), pp. 145–167. ISSN: 1077-2626. DOI: 10.1109/2945.773807.

[108] Rinse van Hees and Jurriaan Hage. “Stable and Predictable Voronoi Treemaps for Software Quality Monitoring”. In: *Elsevier Information and Software Technology* 87 (2017), pp. 242–258. DOI: https://doi.org/10.1016/j.infsof.2016.10.003.

[109] Roland Heilmann, Daniel A. Keim, Christian Panse, and Mike Sips. “RecMap: Rectangular Map Approximations”. In: *Proc. IEEE Symp. on Information Visualization*. InfoVis ’03. 2004, pp. 33–40. DOI: 10.1109/INFVIS.2004.57.

[110] Alexandre Henrique Ichihara Pires, Rodrigo Santos do Amor Divino Lima, Carlos Gustavo Resque dos Santos, Bianchi Serique Meiguins, and Anderson Gregório Marques Soares. “A summarization glyph for sets of unreadable visual items in treemaps”. In: *2020 24th International Conference Information Visualisation (IV)*. IV ’20. 2020, pp. 242–247. DOI: 10.1109/IV51561.2020.00047.

[111] Danny Holten. “Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data”. In: *IEEE Trans Vis Comput Graph.* TVCG ’06 12.5 (2006), pp. 741–748. ISSN: 1077-2626. DOI: 10.1109/TVCG.2006.147.

[112] Danny Holten, Roel. Vliegen, and Jarke J. van Wijk. “Visual Realism for the Visualization of Software Metrics”. In: *Proc. of VISSOFT.* VISSOFT ’05. IEEE Computer Society, 2005, pp. 1–6. DOI: 10.1109/VISSOF.2005.1684299.

[113] Johannes Holvitie, Sherlock A. Licorish, Rodrigo O. Spínola, Sami Hyrynsalmi, Stephen G. MacDonell, Thiago S. Mendes, Jim Buchan, and Ville Leppänen. “Technical Debt and Agile Software Development Practices and Processes: An Industry Practitioner Survey”. In: *Inf. Softw. Technol.* 96.C (2018), pp. 141–160. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2017.11.015.

[114] Michael. S. Horn, Matthew Tobiasz, and Chia Shen. “Visualizing Biodiversity with Voronoi Treemaps”. In: *2009 Sixth International Symposium on Voronoi Diagrams.* 2009, pp. 265–270. DOI: 10.1109/ISVD.2009.22.

[115] Eduard Imhof. “Positioning Names on Maps”. In: *The American Cartographer* 2.2 (1975), pp. 128–144. DOI: 10.1559/152304075784313304.

[116] Tobias Isenberg. “Evaluating and Validating Non-Photorealistic and Illustrative Rendering”. In: *Image and Video based Artistic Stylisation.* Vol. 42. Springer, 2013, pp. 311–331. DOI: 10.1007/978-1-4471-4519-6_15.

[117] Takayuki Itoh, Yasumasa Kajinaga, Yumi Yamaguchi, and Yuko Ikehata. “Hierarchical Data Visualization Using a Fast Rectangle-Packing Algorithm”. In: *Transactions on Visualization and Computer Graphics.* TVCG ’04 10 (2004), pp. 302–313. ISSN: 1077-2626. DOI: 10.1109/TVCG.2004.1272729.

[118] Takayuki Itoh, Hiroki Takakura, Atsushi Sawada, and Koji Koyamada. “Hierarchical Visualization of Network Intrusion Detection Data”. In: *Computer Graphics & Applications* 26.2 (2006), pp. 40–47. ISSN: 0272-1716. DOI: 10.1109/MCG.2006.34.

[119] ŁLukasz Izdebski and Dariusz Sawicki. “Easing Functions in the New Form Based on Bézier Curves”. In: *Computer Vision and Graphics.* Springer International Publishing, Sept. 2016, pp. 37–48. ISBN: 978-3-319464-17-6. DOI: 10.1007/978-3-319-46418-3_4.

[120] Mahipal Jadeja and Rahul Muthu. “Labeled object treemap: A new graph-labeling based technique for visualizing multiple hierarchies”. In: *Annals of Pure and Applied Mathematics* 13 (2017), pp. 49–62. DOI: 10.22457/apam.v13n1a6.

[121] Jacek Jankowski and Martin Hachet. “Advances in Interaction with 3D Environments”. In: *Comput. Graph. Forum* 34.1 (2015), pp. 152–190. ISSN: 0167-7055. DOI: 10.1111/cgf.12466.

[122] Jensen Huang as interviewed by Tom Simonite. “Nvidia CEO: Software Is Eating the World, but AI Is Going to Eat Software”. In: *Technology Review* (May 2017). technologyreview.com/2017/05/12/151722.

[123] Mikael Jern, Jakob Rogstadius, and Tobias Åström. “Treemaps and Choropleth Maps Applied to Regional Hierarchical Statistical Data”. In: *13th International Conference Information Visualisation.* 2009, pp. 403–410. DOI: 10.1109/IV.2009.97.

[124] Brian Scott Johnson. “Treemaps: Visualizing Hierarchical and Categorical Data”. UMI Order No. GAX94-25057. PhD thesis. College Park, MD, USA: University of Maryland, 1993.

[125] Brian Scott Johnson and Ben Shneiderman. “Tree-Maps: A Space-filling Approach to the Visualization of Hierarchical Information Structures”. In: *Proc. Conference on Visualization.* VIS ’91. IEEE, 1991, pp. 284–291. ISBN: 0-8186-2245-8.

[126] John B. Johnston. “Structure of multiple activity algorithms”. In: *Proc. Symp. on Operating Systems Principles.* SOSP ’69. ACM, 1969, pp. 80–82.

[127] John B. Johnston. “The Contour Model of Block Structured Processes”. In: *SIGPLAN Not.* 6.2 (1971), pp. 55–82. ISSN: 0362-1340. DOI: 10.1145/1115880.1115883.

[128] Walter-Alexander Jungmeister and David Turo. *Adapting Treemaps to Stock Portfolio Visualization.* Tech. rep. University of Maryland, 1992.

[129] Konstantinos G. Kakoulis and Ioannis G. Tollis. “Labeling Algorithms”. In: *Handbook of Graph Drawing and Visualization.* Chapman and Hall/CRC, 2013. ISBN: 978-1-584884-12-5.

[130] Benjamin Karran, Jonas Trümper, and Jürgen Döllner. “SYNCTRACE: Visual Thread-Interplay Analysis”. In: *First IEEE Working Conference on Software Visualization.* VISSOFT ’13. 2013, pp. 1–10. ISBN: 978-1-479914-57-9. DOI: 10.1109/VISSOFT.2013.6650534.

[131] Karl G. Karsten. *Charts And Graphs.* Retrieved from archive.org/details/in.ernet.dli.2015.13852. Prentice-Hall, Inc, 1925.

[132] Daniel A. Keim, Gennady Andrienko, Jean-Daniel Fekete, Carsten Gorg, Jorn Kohlhammer, and Guy Melançon. “Visual analytics: Definition, process, and challenges”. In: *Information Visualization. Lecture Notes in Computer Science* 4950 (2008), pp. 154–176. DOI: 10.1007/978-3-540-70956-5_7.

[133] Can Keskin and Volker Vogelmann. “Effective Visualization of Hierarchical Graphs With the Cityscape Metaphor”. In: *Proceedings of the 1997 Workshop on New Paradigms in Information Visualization and Manipulation.* NPIV ’97. ACM, 1997, pp. 52–57. ISBN: 1-58113-051-1. DOI: 10.1145/275519.275531.

[134] Yongjin Kim, Jingyi Yu, Xuan Yu, and Seungyong Lee. “Line-art Illustration of Dynamic and Specular Surfaces”. In: *Transactions on Graphics* 27.5 (), 156:1–156:10. DOI: 10.1145/1409060.1409109.

[135] Gordon Kindlmann and Carlos Scheidegger. “An Algebraic Process for Visualization Design”. In: *IEEE Trans Vis Comput Graph* 20.12 (2014), pp. 2181–2190. ISSN: 1077-2626. DOI: 10.1109/TVCG.2014.2346325.

[136] Claire Knight and Malcom Munro. “Virtual but Visible Software”. In: *Conference on Information Visualization. An International Conference on Computer Visualization and Graphics.* IV ’2000. IEEE. 2000, pp. 198–205. ISBN: 0-769507-43-3. DOI: 10.1109/IV.2000.859756.

[137] Aimi Kobayashi, Kazuo Misue, and Jiro Tanaka. “Edge Equalized Treemaps”. In: *16th International Conference on Information Visualisation.* IV ’12. 2012, pp. 7–12. DOI: 10.1109/IV.2012.12.

[138] Nicholas Kong, Jeffrey Heer, and Maneesh Agrawala. “Perceptual Guidelines for Creating Rectangular Treemaps”. In: *IEEE Trans Vis Comput Graph* 16.6 (2010), pp. 990–998. ISSN: 1077-2626. DOI: 10.1109/TVCG.2010.186.

[139] Bastian König, **Daniel Limberger**, Jan Klimke, Benjamin Hagedorn, and Jürgen Döllner. “RoomCanvas: A Visualization System for Spatiotemporal Temperature Data in Smart Homes”. In: *EuroVis 2021 - Short Papers.* EuroVis ’21. EG, 2021, pp. 13–17. ISBN: 978-3-038681-43-4. DOI: 10.2312/evs.20211048.

[140] Jonas Kordt, Paul Brachmann, **Daniel Limberger**, and Christoph Lippert. “Interactive Volumetric Region Growing for Brain Tumor Segmentation on MRI Using WebGL”. In: *Proceedings of the 26th International Conference on 3D Web Technology.* Web3D ’21. ACM, 2021, 2:1–8. ISBN: 978-1-450390-95-8. DOI: 10.1145/3485444.3487640.

[141] Joseph B. Kruskal and James M. Landwehr. “Icicle plots: Better displays for hierarchical clustering”. In: *Taylor & Francis The American Statistician* 37 (1983), pp. 162–168.

[142] Adrian Kuhn, Peter Loretan, and Oscar Nierstrasz. “Consistent Layout for Thematic Software Maps”. In: *2008 15th Working Conference on Reverse Engineering.* WCRE ’08. IEEE, 2008, pp. 209–218. DOI: 10.1109/WCRE.2008.45.

[143] Daniel O. Kutz. “Examining the Evolution and Distribution of Patent Classifications”. In: *Proc. 8th International Conference on Information Visualisation.* IV04. 2004, pp. 983–988. DOI: 10.1109/IV.2004.1320261.

[144] Sehi L'Yi, Jaemin Jo, and Jinwook Seo. "Comparative Layouts Revisited: Design Space, Guidelines, and Future Directions". In: *IEEE Trans Vis Comput Graph* 27.2 (2021), pp. 1525–1535. DOI: 10.1109/TVCG.2020.3030419.

[145] Adam Lake, Carl Marshall, Mark Harris, and Marc Blackstein. "Stylized Rendering Techniques for Scalable Real-Time 3D Animation". In: *Proceedings of the 1st International Symposium on Non-Photorealistic Animation and Rendering*. NPAR '00. ACM, 2000, pp. 13–20. ISBN: 978-1-581132-77-9. DOI: 10.1145/340916.340918.

[146] Guillaume Langelier, Houari Sahraoui, and Pierre Poulin. "Exploring the Evolution of Software Quality with Animated Visualization". In: *Proceedings of the 2008 IEEE Symposium on Visual Languages and Human-Centric Computing*. VL-HCC '08. IEEE Computer Society, 2008, pp. 13–20. ISBN: 978-1-424425-28-0. DOI: 10.1109/VLHCC.2008.4639052.

[147] Guillaume Langelier, Houari Sahraoui, and Pierre Poulin. "Visualization-based Analysis of Quality for Large-scale Software Systems". In: *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*. ASE '05. ACM, 2005, pp. 214–223. ISBN: 1-58113-993-4. DOI: 10.1145/1101908.1101941.

[148] Eric Lengyel. "GPU-Centered Font Rendering Directly from Glyph Outlines". In: *Journal of Computer Graphics Techniques*. JCGT '17 6.2 (2017), pp. 31–47. ISSN: 2331-7418.

[149] Jie Liang, Quang Vinh Nguyen, Simeon Simoff, and Mao Lin Huang. "Divide and Conquer Treemaps". In: *Journal of Visual Languages & Computing* 31 (2015), pp. 104–127. ISSN: 1045-926X. DOI: 10.1016/j.jvlc.2015.10.009.

[150] Peter Liggesmeyer, Jens Heidrich, Jürgen Münch, Robert Kalcklösch, Henning Barthel, and Dirk Zeckzer. "Visualization of Software and Systems as Support Mechanism for Integrated Software Project Control". In: *Proc. of HCI*. 2009, pp. 846–855. DOI: 10.1007/978-3-642-02574-7_94.

[151] Daniel Müller (now Limberger). "Computergenerierte Bleistiftzeichnungen von 3D-Stadtmodellen". daniellimberger.de/[../...]Computergenerierte Bleistiftzeichnungen von 3D-Stadtmodellen.pdf. Bachelor's Thesis. Hasso Plattner Institute, University of Potsdam, 2009.

[152] Shixia Liu, Nan Cao, and Hao Lv. "Interactive Visual Analysis of the NSF Funding Information". In: *2008 IEEE Pacific Visualization Symposium*. 2008, pp. 183–190. DOI: 10.1109/PACIFICVIS.2008.4475475.

[153] Zhicheng Liu and John Stasko. "Mental Models, Visual Reasoning and Interaction in Information Visualization: A Top-down Perspective". In: *Transactions on Visualization and Computer Graphics* 16.6 (2010), pp. 999–1008. DOI: 10.1109/TVCG.2010.177.

[154] Haik Lorenz, Matthias Trapp, Jürgen Döllner, and Markus Jobst. "Interactive Multi-Perspective Views of Virtual 3D Landscape and City Models". In: *The European Information Society*. Springer, 2008, pp. 301–321. DOI: 10.1007/978-3-540-78946-8_16.

[155] Hao Lü and James Fogarty. "Cascaded Treemaps: Examining the Visibility and Stability of Structure in Treemaps". In: *Proceedings of Graphics Interface 2008*. GI '08. Canadian Information Processing Society, 2008, pp. 259–266. ISBN: 978-1-568814-23-0. DOI: 10.5555/1375714.1375758.

[156] Martin Luboschik, Axel Radloff, and Heidrun Schumann. "Using Non-Photorealistic Rendering Techniques for the Visualization of Uncertainty". In: *Poster at IEEE Conference on Information Visualization*. InfoVis '10. Poster. 2010.

[157] Martin Luboschik and Heidrun Schumann. "Discovering the covered: Ghost-views in information visualization". In: *Proceedings of the 16th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*. WSCG '08. 2008, pp. 113–118. ISBN: 978-80-86943-15-2. DOI: 11025/10927.

[158] Shiyong Ma and Zhen Zhang. "OmicsMapNet: Transforming Omics Data to Take Advantage of Deep Convolutional Neural Network for Discovery". In: *arXiv Computing Research Repository* (2018).

[159] Stefan Maass and Jürgen Döllner. “Embedded labels for line features in interactive 3D virtual environments”. In: *Proceedings of the 5th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa.* AFRIGRAPH ’07. ACM, 2007, pp. 53–59. ISBN: 978-1-595939-06-7. DOI: 10.1145/1294685.1294695.

[160] Alan M. MacEachren. “The Evolution of Thematic Cartography / A Research Methodology and Historical Review”. In: *Cartographica: The International Journal for Geographic Information and Geovisualization* 16 (1979), pp. 17–33.

[161] Alan M. MacEachren, Robert E. Roth, James O’Brien, Bonan Li, Derek Swingley, and Mark Gahegan. “Visual semiotics & uncertainty visualization: An empirical study”. In: *IEEE Trans Vis Comput Graph.* TVCG ’12 18.12 (2012), pp. 2496–2505. DOI: 10.1109/TVCG.2012.279.

[162] Andrian Marcus, Louis Feng, and Jonathan I. Maletic. “3D Representations for Software Visualization”. In: *Proceedings of the 2003 ACM Symposium on Software Visualization.* SoftVis ’03. ACM, 2003, 27–ff. ISBN: 1-58113-642-0. DOI: 10.1145/774833.774837.

[163] Fernando Marson and Soraia Raupp Musse. “Automatic Real-time Generation of Floor Plans Based on Squarified Treemaps Algorithm”. In: *ACM International Journal of Computer Games Technology* (2010), 7:1–10. DOI: 10.1155/2010/624817.

[164] Vivian C McAlister. “Datum isn’t; data are”. In: *Canadian journal of surgery. Journal canadien de chirurgie* 59.4 (2016), pp. 220–221. ISSN: 0008-428X. DOI: 10.1503/cjs.009316.

[165] Thomas J. McCabe. “A Complexity Measure”. In: *IEEE Transactions on Software Engineering* 2.4 (1976), pp. 308–320. DOI: 10.1109/TSE.1976.233837.

[166] David McCandless. “Knowledge Is Beautiful”. In: 2013.

[167] David McCandless. *The Beauty of Data Visualization.* ted.com/talks/david_mccandless_the_beauty_of_data_visualization. Aug. 2010.

[168] Morgan McGuire, Brian Osman, Michael Bukowski, and Padraic Hennessy. “The Alchemy Screen-Space Ambient Obscurance Algorithm”. In: *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics.* HPG ’11. ACM, 2011, pp. 25–32. ISBN: 978-1-450308-96-0. DOI: 10.1145/2018323.2018327.

[169] Barbara J. Meier. “Painterly Rendering for Animation”. In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques.* SIGGRAPH ’96. ACM, 1996, pp. 477–484. DOI: 10.1145/237170.237288.

[170] Maysam Mirahmadi and Abdallah Shami. “A Novel Algorithm for Real-time Procedural Generation of Building Floor Plans”. In: *arXiv Computing Research Repository* abs/1211.5842 (2012).

[171] Kazuo Misue, Peter Eades, Wei Lai, and Kozo Sugiyama. “Layout Adjustment and the Mental Map”. In: *Elsevier Journal of Visual Languages & Computing* 6.2 (1995), pp. 183–210. ISSN: 1045-926X. DOI: https://doi.org/10.1006/jvlc.1995.1010.

[172] Pascal Molli, Hala Skaf-Molli, and Christophe Bouthier. “State Treemap: an Awareness Widget for Multi-Synchronous Groupware”. In: *Proceedings of the 7th International Workshop on Groupware.* CRIWG ’01. IEEE, 2001, pp. 106–114. DOI: 10.1109/CRIWG.2001.951823.

[173] David Moreno-Lumbreras, Roberto Minelli, Andrea Villaverde, Jesús M. González-Barahona, and Michele Lanza. “CodeCity: On-Screen or in Virtual Reality?” In: *Working Conference on Software Visualization.* VISSOFT ’21. 2021, pp. 12–22. DOI: 10.1109/VISSOFT52517.2021.00011.

[174] Jurriaan D. Mulder, Frans C. A. Groen, and Jarke J. van Wijk. “Pixel Masks for Screen-Door Transparency”. In: *Proceedings of the Conference on Visualization.* VIS ’98. IEEE Computer Society Press, 1998, pp. 351–358. ISBN: 1-581131-06-2. DOI: 10.5555/288216.288309.

[175] Tanja Munz, Michael Burch, Toon van Benthem, Yoeri Poels, Fabian Beck, and Daniel Weiskopf. “Overlap-Free Drawing of Generalized Pythagoras Trees for Hierarchy Visualization”. In: *2019 IEEE Visualization Conference.* VIS ’19. 2019, pp. 251–255. DOI: 10.1109/VISUAL.2019.8933606.

[176] Tamara Munzner. *Visualization Analysis and Design*. AK Peters Visualization Series. CRC Press, 2014. ISBN: 978-1-498759-71-7. DOI: 10.1201/b17511.

[177] I. Nassi and Ben Shneiderman. “Flowchart Techniques for Structured Programming”. In: *SIGPLAN Notices* 8.8 (1973). ACM, pp. 12–26. ISSN: 0362-1340. DOI: 10.1145/953349.953350.

[178] Quang Vinh Nguyen and Mao Lin Huang. “Improvements of Space-optimized Tree for Visualizing and Manipulating Very Large Hierarchies”. In: *Selected Papers from the 2002 Pan-Sydney Workshop on Visualisation - Volume 22*. VIP ’02. Australian Computer Society, Inc., 2002, pp. 75–81. ISBN: 1-920682-01-5. DOI: 10.5555/1164094.1164106.

[179] Marc Nienhaus and Jürgen Döllner. “Sketchy Drawings”. In: *Proc. ACM 3rd International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa*. AFRIGRAPH ’04. ACM, 2004, pp. 73–81. ISBN: 978-1-581138-63-4. DOI: 10.1145/1029949.1029963.

[180] Arlind Nocaj and Ulrik Brandes. “Computing Voronoi Treemaps: Faster, Simpler, and Resolution-independent”. In: *EG Computer Graphics Forum* 31.3pt1 (2012), pp. 855–864. DOI: 10.1111/j.1467-8659.2012.03078.x.

[181] Steffen Oeltze-Jafra and Bernhard Preim. “Survey of Labeling Techniques in Medical Visualizations”. In: *Proceedings of the 4th Eurographics Workshop on Visual Computing for Biology and Medicine*. VCBM ’14. Vienna, Austria: Eurographics Association, 2014, pp. 199–208. ISBN: 978-3-905674-62-0. DOI: 10.2312/vcbm.20141192.

[182] S. Okajima and Y. Okada. “Treecube+3D-ViSOM: Combinational Visualization Tool for Browsing 3D Multimedia Data”. In: *11th International Conference on Information Visualization*. 2007, pp. 40–45. DOI: 10.1109/IV.2007.117.

[183] OpenAI. “ChatGPT”. In: (Nov. 2022). openai.com/blog/chatgpt.

[184] OpenAI and GitHub. “GitHub Copilot”. In: (2021). copilot.github.com.

[185] Philipp Otto, **Daniel Limberger**, and Jürgen Döllner. “Physically-based Environment and Area Lighting using Progressive Rendering in WebGL”. In: *Proceedings of the 25th International Conference on 3D Web Technology*. Web3D ’20. ACM, 2020, 15:1–9. ISBN: 978-1-450381-69-7. DOI: 10.1145/3424616.3424697.

[186] Thomas Panas, Rebecca Berrigan, and John Grundy. “A 3D Metaphor for Software Production Visualization”. In: *Proc. 7th International Conf. on Information Visualization*. 2003, pp. 314–319. DOI: 10.1109/IV.2003.1217996.

[187] Sebastian Pasewaldt, Matthias Trapp, and Jürgen Döllner. “Multiscale Visualization of 3D Geovirtual Environments Using View-Dependent Multi-Perspective Views”. In: *Journal of WSCG* 19.3 (2011), pp. 111–118.

[188] Ken Perlin. “Improving Noise”. In: *ACM Trans. Graph.* 21.3 (2002), pp. 681–682. ISSN: 0730-0301. DOI: 10.1145/566654.566636.

[189] Emil Praun, Hugues Hoppe, Matthew Webb, and Adam Finkelstein. “Real-time Hatching”. In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’01. ACM, 2001, pp. 579–584. DOI: 10.1145/383259.383328.

[190] Zheng Qin, Michael D. McCool, and Craig S. Kaplan. “Real-time Texture-mapped Vector Glyphs”. In: *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*. I3D ’06. ACM, 2006, pp. 125–132. ISBN: 1-595932-95-X. DOI: 10.1145/1111411.1111433.

[191] P. Samuel Quinan, Lace M. K. Padilla, Sarah H. Creem-Regehr, and Miriah Meyer. “Examining Implicit Discretization in Spectral Schemes”. In: *Computer Graphics Forum* 38.3 (2019), pp. 363–374. DOI: 10.1111/cgf.13695.

[192] Erwin Raisz. “Rectangular Statistical Cartograms of the World”. In: *Journal of Geography* 35.1 (1936), pp. 8–10. DOI: 10.1080/00221343608987880.

[193] Erwin Raisz. “The Rectangular Statistical Cartogram”. In: *AGS Geographical Review* 24.2 (1934), pp. 292–296. ISSN: 00167428. DOI: 10.2307/208794.

[194] Jun Rekimoto and Mark Green. “The Information Cube: Using Transparency in 3D Information Visualization”. In: *Proceedings of the Third Annual Workshop on Information Technologies & Systems*. WITS ’93. 1993, pp. 125–132.

[195] Tobias Ritschel, Thorsten Grosch, and Hans-Peter Seidel. “Approximating Dynamic Global Illumination in Image Space”. In: *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*. I3D ’09. ACM, 2009, pp. 75–82. ISBN: 978-1-605584-29-4. DOI: 10.1145/1507149.1507161.

[196] Richard C. Roberts, Chao Tong, Robert S. Laramee, Gary A. Smith, Paul Brookes, and Tony D’Cruze. “Interactive Analytical Treemaps for Visualisation of Call Centre Data”. In: *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*. The Eurographics Association, 2016. ISBN: 978-3-03868-026-0. DOI: 10.2312/stag.20161370.

[197] A. H. Robinson, J. B. Harley, D. Woodward, and G. M. Lewis. *Early Thematic Mapping in the History of Cartography*. University of Chicago Press, 1982. ISBN: 978-0-226722-85-6.

[198] René Rosenbaum and Bernd Hamann. “Progressive Presentation of Large Hierarchies Using Treemaps”. In: *5th International Symposium on Advances in Visual Computing*. Springer Berlin Heidelberg, 2009, pp. 71–80. ISBN: 978-3-642105-20-3. DOI: 10.1007/978-3-642-10520-3_7.

[199] Ruth Rosenholtz, Yuanzhen Li, Jonathan Mansfield, and Zhenlan Jin. “Feature Congestion: A Measure of Display Clutter”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’05. ACM, 2005, pp. 761–770. ISBN: 978-1-581139-98-3. DOI: 10.1145/1054972.1055078.

[200] Robert E. Roth. “Visual Variables”. In: *International Encyclopedia of Geography: People, the Earth, Environment and Technology*. John Wiley & Sons, Ltd, 2016. ISBN: 978-1-118786-35-2. DOI: 10.1002/9781118786352.wbieg0761.

[201] Dominik Sacha, Andreas Stoffel, Florian Stoffel, Bum Chul Kwon, Geoffrey Ellis, and Daniel A. Keim. “Knowledge Generation Model for Visual Analytics”. In: *IEEE Trans Vis Comput Graph*. TVCG ’14 20.12 (2014), pp. 1604–1613. ISSN: 1077-2626. DOI: 10.1109/TVCG.2014.2346481.

[202] Selan dos Santos and Ken Brodlie. “Gaining understanding of multivariate and multidimensional data through visualization”. In: *Computers and Graphics* 28.3 (2004), pp. 311–325. ISSN: 0097-8493. DOI: 10.1016/j.cag.2004.03.013.

[203] Joshua D Scarsbrook, Ryan K L Ko, Bill Rogers, and David Bainbridge. “MetropolJS: Visualizing and Debugging Large-Scale Javascript Program Structure with Treemaps”. In: *Proc. of the 26th Conference on Program Comprehension*. ICPC ’18. ACM, 2018, pp. 389–392. ISBN: 978-1-450357-14-2. DOI: 10.1145/3196321.3196368.

[204] Willy Scheibel, Stefan Buschmann, Matthias Trapp, and Jürgen Döllner. “Attributed Vertex Clouds”. In: *GPU Zen: Advanced Rendering Techniques*. Bowker Identifier Services, 2017. Chap. Geometry Manipulation, pp. 3–21.

[205] Willy Scheibel, Matthias Trapp, and Jürgen Döllner. “Interactive Revision Exploration using Small Multiples of Software Maps”. In: *Proceedings of the 11th Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 2: IVAPP, (VISIGRAPP 2016)*. INSTICC. SciTePress, 2016, pp. 131–138. ISBN: 978-989758-175-5. DOI: 10.5220/0005694401310138.

[206] Willy Scheibel, Christopher Weyand, and Jürgen Döllner. “EvoCells – A Treemap Layout Algorithm for Evolving Tree Data”. In: *Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications – Volume 2: IVAPP*. IVAPP ’18. SciTePress, 2018, pp. 273–280. ISBN: 978-9-897582-89-9. DOI: 10.5220/0006617102730280.

[207] Stefan Schlechtweg and Andreas Raab. “Rendering Line Drawings for Illustrative Purposes”. In: *Computational Visualization: Graphics, Abstraction, and Interactivity*. Springer Verlag, 1998, pp. 65–89. ISBN: 978-3-540637-37-0.

[208] Karen B. Schloss, Connor C. Gramazio, Allison T. Silverman, Madeline L. Parker, and Audrey S. Wang. “Mapping Color to Meaning in Colormap Data Visualizations”. In: *Transactions on Visualization and Computer Graphics* 25.1 (2018), pp. 810–819. DOI: 10.1109/TVCG.2018.2865147.

[209] Hans-Jörg Schulz. “Treevis.net: A Tree Visualization Reference”. In: *IEEE Computer Graphics and Applications* 31.6 (2011), pp. 11–15. DOI: 10.1109/MCG.2011.103.

[210] Hans-Jörg Schulz, Steffen Hadlak, and Heidrun Schumann. “The Design Space of Implicit Hierarchy Visualization: A Survey”. In: *IEEE Trans Vis Comput Graph.* TVCG ’10 17.4 (2011), pp. 393–411. DOI: 10.1109/TVCG.2010.79.

[211] Hans-Jörg Schulz, Martin Luboschik, and Heidrun Schumann. “Interactive Poster: Exploration of the 3D Treemap Design Space”. In: *IEEE Symposium on Information Visualization’07*. Citeseer. 2007.

[212] ISO Central Secretary. *International Standard – Software engineering – Software life cycle processes – Maintenance*. Standard ISO/IEC/IEEE 2021. International Organization for Standardization, 2022.

[213] Kai Selgrad, Christian Reintges, Dominik Penk, Pascal Wagner, and Marc Stamminger. “Real-time Depth of Field Using Multi-layer Filtering”. In: *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games*. i3D ’15. ACM, 2015, pp. 121–127. DOI: 10.1145/2699276.2699288.

[214] Amir Semmo, Matthias Trapp, Jan Eric Kyprianidis, and Jürgen Döllner. “Interactive Visualization of Generalized Virtual 3D City Models Using Level-of-Abstraction Transitions”. In: *Computer Graphics Forum* 31 (2012), pp. 885–894. ISSN: 0167-7055. DOI: 10.1111/j.1467-8659.2012.03081.x.

[215] Ben Shneiderman. “The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations”. In: *IEEE Symposium on Visual Languages*. IEEE Computer Society, 1996, pp. 336–343. DOI: 10.1109/VL.1996.545307.

[216] Ben Shneiderman. “Tree Visualization with Tree-Maps: 2-d Space-Filling Approach”. In: *Transactions on Graphics* 11.1 (1992), pp. 92–99. ISSN: 0730-0301. DOI: 10.1145/102377.115768.

[217] Ben Shneiderman and Martin Wattenberg. “Ordered Treemap Layouts”. In: *Proc. IEEE Symp. on Information Visualization*. 2001, pp. 73–78. DOI: 10.1109/INFVIS.2001.963283.

[218] Aidan Slingsby, Jason Dykes, and Jo Wood. “Configuring Hierarchical Layouts to Address Research Questions”. In: *IEEE Trans Vis Comput Graph.* TVCG ’09 15.6 (2009), pp. 977–984. ISSN: 1077-2626. DOI: 10.1109/TVCG.2009.128.

[219] Aidan Slingsby, Jason Dykes, and Jo Wood. “Using Treemaps for Variable Selection in Spatio-temporal Visualisation”. In: *Palgrave Information Visualization* 7.3-4 (2008), pp. 210–224. DOI: 10.1057/PALGRAVE.IVS.9500185.

[220] Anderson Gregório Marques Soares, Elvis Thermo Carvalho Miranda, Rodrigo Santos do Amor Divino Lima, Carlos Gustavo Resque dos Santos, and Bianchi Serique Meiguins. “Depicting More Information in Enriched Squarified Treemaps with Layered Glyphs”. In: *Information* 11.2 (2020), 123:1–21. ISSN: 2078-2489. DOI: 10.3390/info11020123.

[221] Max Sondag, Wouter Meulemans, Christoph Schulz, Kevin Verbeek, Daniel Weiskopf, and Bettina Speckmann. “Uncertainty Treemaps”. In: *2020 IEEE Pacific Visualization Symposium*. PacificVis ’20. 2020, pp. 111–120. DOI: 10.1109/PacificVis48177.2020.7614.

[222] John Stasko, Richard Catrambone, Mark Guzdial, and Kevin McDonald. *An evaluation of space-filling information visualizations for depicting hierarchical structures*. Tech. rep. 5. GVU Technical Report;GIT-GVU-00-03. 2000, pp. 663–694. DOI: 10.1006/ijhc.2000.0420.

[223] Marcel Steinbeck. “An Arc-based Approach for Visualization of Code Smells”. In: *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering*. SANER 17. 2017, pp. 397–401. DOI: 10.1109/SANER.2017.7884641.

[224] Marcel Steinbeck, Rainer Koschke, and Marc O. Rüdel. “Comparing the EvoStreets Visualization Technique in Two- and Three-dimensional Environments: A Controlled Experiment”. In: *Proceedings of the 27th IEEE / ACM International Conference on Program Comprehension*. ICPC ’19. IEEE Press, 2019, pp. 231–242. DOI: 10.1109/icpc.2019.00042.

[225] Frank Steinbrückner and Claus Lewerentz. “Representing Development History in Software Cities”. In: *Proceedings of the 5th International Symposium on Software Visualization*. SOFTVIS ’10. ACM, 2010, pp. 193–202. ISBN: 978-1-450300-28-5. DOI: 10.1145/1879211.1879239.

[226] Frank Steinbrückner and Claus Lewerentz. “Understanding Software Evolution with Software Cities”. In: *Information Visualization* 12.2 (2013), pp. 200–216. DOI: 10.1177/1473871612438785.

[227] Thomas Strothotte and Stefan Schlechtweg. *Non-Photorealistic Computer Graphics: Modeling, Rendering, and Animation*. Morgan Kaufmann Publishers Inc., 2002. ISBN: 978-1-558607-87-3. DOI: 10.5555/544522.

[228] Suphachai Sutanthavibul, Eugene Shragowitz, and J. Ben Rosen. “An Analytical Approach to Floorplan Design and Optimization”. In: *Proc. of the 27th ACM/IEEE Design Automation Conference*. DAC ’90. ACM, 1991, pp. 187–192. ISBN: 0897913639. DOI: 10.1145/123186.123255.

[229] László Szécsi and Marcell Szirányi. “Recursive Procedural Tonal Art Maps”. In: *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*. WSCG ’14. The Eurographics Association, 2014, pp. 57–66.

[230] Susanne Tak and Andy Cockburn. “Enhanced Spatial Stability with Hilbert and Moore Treemaps”. In: *IEEE Trans Vis Comput Graph*. TVCG ’12 19.1 (2013), pp. 141–148. ISSN: 1077-2626. DOI: 10.1109/TVCG.2012.108.

[231] Justin Talbot, Vidya Setlur, and Anushka Anand. “Four Experiments on the Perception of Bar Charts”. In: *IEEE Trans Vis Comput Graph*. TVCG ’14 20.12 (2014), pp. 2152–2160. DOI: 10.1109/TVCG.2014.2346320.

[232] Yoichi Tanaka, Yoshihiro Okada, and Koichi Niijima. “Interactive Interfaces of Treecube for Browsing 3D Multimedia Data”. In: *Proceedings of the Working Conference on Advanced Visual Interfaces*. AVI ’04. ACM, 2004, pp. 298–302. ISBN: 1-58113-867-9. DOI: 10.1145/989863.989914.

[233] Markus Tatzgern, Denis Kalkofen, Raphael Grasset, and Dieter Schmalstieg. “Hedgehog Labeling: View Management Techniques for External Labels in 3D Space”. In: *2014 IEEE Virtual Reality*. VR ’14. IEEE, pp. 27–32. DOI: 10.1109/VR.2014.6802046.

[234] Alexandru C. Telea, Ozan Ersoy, and Lucian Voinea. “Visual Analytics in Software Maintenance: Challenges and Opportunities”. In: *International Symposium on Visual Analytics Science and Technology*. EuroVAST ’10. The Eurographics Association, 2010. ISBN: 978-3-905673-74-6. DOI: 10.2312/PE/EuroVAST/EuroVAST10/075-080.

[235] Sidharth Thakur and Theresa-Marie Rhyne. “Data Vases: 2D and 3D Plots for Visualizing Multiple Time Series”. In: *5th International Symposium on Advances in Visual Computing*. ISVC ’09. Springer Berlin Heidelberg, 2009, pp. 929–938. ISBN: 978-3-642105-20-3. DOI: 10.1007/978-3-642-10520-3_89.

[236] Sabine Timpf. “Abstraction, Levels of Detail, and Hierarchies in Map Series”. In: *Spatial Information Theory. Cognitive and Computational Foundations of Geographic Information Science*. Springer Berlin Heidelberg, 1999, pp. 125–139. ISBN: 978-3-540483-84-7. DOI: 10.1007/3-540-48384-5_9.

[237] Christian Tominski, Stefan Gladisch, Ulrike Kister, Raimund Dachselt, and Heidrun Schumann. “Interactive Lenses for Visualization: An Extended Survey”. In: *Computer Graphics Forum* 36.6 (2017), pp. 173–200. DOI: 10.1111/cgf.12871.

[238] Christian Tominski and Heidrun Schumann. *Interactive Visual Data Analysis*. AK Peters Visualization Series. CRC Press, 2020. ISBN: 978-1-498753-98-2. DOI: 10.1201/9781315152707. URL: https://ivda-book.de.

[239] Farshad Ghassemi Toosi and Nikola S Nikolov. “Circular Tree Drawing by Simulating Network Synchronisation Dynamics and Scaling”. In: *Graph Drawing*. Springer, 2014, pp. 511–512.

[240] Matthias Trapp, Tassilo Glander, Henrik Buchholz, and Jürgen Döllner. “3D Generalization Lenses for Interactive Focus + Context Visualization of Virtual City Models”. In: *Proceedings of the 12th International Conference Information Visualisation*. 2008, pp. 356–361. DOI: 10.1109/IV.2008.18.

[241] Matthias Trapp, Sebastian Schmechel, and Jürgen Döllner. “Interactive Rendering of Complex 3D-Treemaps with a Comparative Performance Evaluations”. In: *Proceedings of the International Conference on Computer Graphics Theory and Applications.* GRAPP ’13. SciTePress, 2013, pp. 165–175. ISBN: 978-9-898565-46-4. DOI: 10.5220/0004290101650175.

[242] Jonas Trümper and Jürgen Döllner. “Extending Recommendation Systems with Software Maps”. In: *Proceedings of the Third International Workshop on Recommendation Systems for Software Engineering.* RSSE ’12. IEEE Press, 2012, pp. 92–96. ISBN: 978-1-467317-59-7. DOI: 10.1109/RSSE.2012.6233420.

[243] Alexey Tsymbal, Martin Huber, Sonja Zillner, Tamás Hauer, and Shaohua Kevin Zhou. “Visualizing Patient Similarity in Clinical Decision Support”. In: *LWA 2007 LWA 2007, Lernen - Wissen - Adaption.* 2007, pp. 304–311.

[244] Ying Tu and Han-Wei Shen. “Visualizing Changes of Hierarchical Data using Treemaps”. In: *IEEE Trans Vis Comput Graph.* TVCG ’07 13.6 (2007), pp. 1286–1293. ISSN: 1077-2626. DOI: 10.1109/TVCG.2007.70529.

[245] Edward R. Tufte. *The Visual Display of Quantitative Information.* 2nd ed. Graphics Press, 2001. ISBN: 9-78-096139-214-7.

[246] David Turo. “Hierarchical Visualization with Treemaps: Making Sense of Pro Basketball Data”. In: *Conference Companion on Human Factors in Computing Systems.* CHI ’94. ACM, 1994, pp. 441–442. ISBN: 0-89791-651-4. DOI: 10.1145/259963.260441.

[247] David Turo and Brian S. Johnson. “Improving the Visualization of Hierarchies with Treemaps: Design Issues and Experimentation”. In: *Proceedings of the 3rd Conference on Visualization.* VIS ’92. IEEE Computer Society Press, 1992, pp. 124–131. ISBN: 0-8186-2896-0. DOI: 10.5555/949685.949711.

[248] United States Census Office. 11th census, 1890 and Henry Gannett. *Statistical atlas of the United States, based upon the results of the eleventh census.* Retrieved from the Library of Congress, loc.gov/item/07019233/. Washington, Government Printing Office, 1898.

[249] United States Census Office. 12th census, 1900 and Henry Gannett. *Statistical Atlas of the United States 1900.* Retrieved from archive.org/details/ statisticalatlas00unit/. Washington, United States Census Office, 1903.

[250] United States Census Office. 9th census, 1870 and Francis Amasa Walker. *Statistical atlas of the United States based on the results of the ninth census 1870 with contributions from many eminent men of science and several departments of the government.* Retrieved from the Library of Congress, loc.gov/item/05019329/. New York: Julius Bien, lith, 1874.

[251] Timothy Urness, Victoria Interrante, Ivan Marusic, Ellen Longmire, and Bharathram Ganapathisubramani. “Effectively Visualizing Multi-Valued Flow Data using Color and Texture”. In: *Proc. 14th IEEE Visualization 2003.* VIS ’03. IEEE Computer Society, 2003, pp. 115–121. DOI: 10.1109/VISUAL.2003.1250362.

[252] Mikael Vaaraniemi, Martin Freidank, and Rüdiger Westermann. “Enhancing the Visibility of Labels in 3D Navigation Maps”. In: *Progress and New Trends in 3D Geoinformation Sciences.* Springer Berlin Heidelberg, 2013, pp. 23–40. ISBN: 978-3-642297-93-9. DOI: 10.1007/978-3-642-29793-9_2.

[253] Joel Vanderpypen and Laurent Schumacher. “Treemap-Based Burst Mapping Algorithm for Downlink Mobile WiMAX Systems”. In: *2011 IEEE Vehicular Technology Conference.* VTC ’11. 2011, pp. 1–5. DOI: 10.1109/VETECF.2011.6093072.

[254] Eduardo Faccin Vernier, João Luiz Dihl Comba, and Alexandru C. Telea. “A Stable Greedy Insertion Treemap Algorithm for Software Evolution Visualization”. In: *Proceedings of the 31st Conference on Graphics, Patterns and Images.* IEEE, 2018, pp. 158–165. DOI: 10.1109/SIBGRAPI.2018.00027.

[255] Eduardo Faccin Vernier, Max Sondag, João Luiz Dihl Comba, Bettina Speckmann, Alexandru C. Telea, and Kevin Verbeek. “Quantitative Comparison of Time-Dependent Treemaps”. In: *Comput. Graph. Forum* 39.3 (2020), pp. 393–404. DOI: 10.1111/cgf.13989.

[256] Eduardo Faccin Vernier, Alexandru C. Telea, and João Comba. “Quantitative Comparison of Dynamic Treemaps for Software Evolution Visualization”. In: *Proc. IEEE Working Conference on Software Visualization*. Vol. 00. VISSOFT ’18. IEEE, 2018, pp. 96–106. DOI: 10.1109/VISSOFT.2018.00018.

[257] Marcos Viana, André C. Hora, and Marco Tulio Valente. “CodeCity for (and by) JavaScript”. In: *arXiv Computing Research Repository* abs/1705.05476 (2017). URL: http://arxiv.org/abs/1705.05476.

[258] Roel Vliegen, Jarke J. van Wijk, and E. J. van der Linden. “Visualizing Business Data with Generalized Treemaps”. In: *IEEE Trans Vis Comput Graph*. TVCG ’06 12.5 (2006), pp. 789–796. ISSN: 1077-2626. DOI: 10.1109/TVCG.2006.200.

[259] Georg Von Mayr. *Die Gesetzmässigkeit im Gesellschaftsleben*. Vol. 23. De Gruyter Oldenbourg, 1877.

[260] Lukas Wagner, **Daniel Limberger**, Willy Scheibel, Matthias Trapp, and Jürgen Döllner. “A Framework for Interactive Exploration of Clusters in Massive Data using 3D Scatter Plots and WebGL”. In: *Proceedings of the 25th International Conference on 3D Web Technology*. Web3D ’20. ACM, 2020, 31:1–2. ISBN: 978-1-450381-69-7. DOI: 10.1145/3424616.3424730.

[261] Nicholas Waldin, Manuela Waldner, Mathieu Le Muzic, Eduard Gröller, David S. Goodsell, Ludovic Autin, Arthur J. Olson, and Ivan Viola. “Cuttlefish: Color Mapping for Dynamic Multi-Scale Visualizations”. In: *Computer Graphics Forum* 38.6 (2019), pp. 150–164. DOI: 10.1111/cgf.13611.

[262] Weixin Wang, Hui Wang, Guozhong Dai, and Hongan Wang. “Visualization of Large Hierarchical Data by Circle Packing”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’06. ACM, 2006, pp. 517–520. ISBN: 1-59593-372-7. DOI: 10.1145/1124772.1124851.

[263] Yan-Chao Wang, Jigang Liu, Feng Lin, and Hock-Soon Seah. “Generating Orthogonal Voronoi Treemap for Visualization of Hierarchical Data”. In: *Advances in Computer Graphics*. Springer International Publishing, 2020, pp. 394–402. ISBN: 978-3-030618-64-3.

[264] Yue Wang, Soon Tee Teoh, and Kwan-Liu Ma. “Evaluating the Effectiveness of Tree Visualization Systems for Knowledge Discovery”. In: *EuroVis*. Vol. 6. 2006, pp. 67–74.

[265] Colin Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann Series in Interactive Technologies. Morgan Kaufmann, 2020. ISBN: 978-0-128128-75-6.

[266] Martin Wattenberg. “A note on space-filling visualizations and space-filling curves”. In: *Proc. IEEE Symp. on Information Visualization*. InfoVis ’05. 2005, pp. 181–186. DOI: 10.1109/INFVIS.2005.1532145.

[267] Martin Wattenberg. “Visualizing the Stock Market”. In: *CHI ’99 Extended Abstracts on Human Factors in Computing Systems*. CHI EA ’99. ACM, 1999, pp. 188–189. ISBN: 1-58113-158-5. DOI: 10.1145/632716.632834.

[268] Matthew Webb, Emil Praun, Adam Finkelstein, and Hugues Hoppe. “Fine Tone Control in Hardware Hatching”. In: *Proceedings of the 2nd International Symposium on Non-photorealistic Animation and Rendering*. NPAR ’02. ACM, 2002, pp. 53–59. DOI: 10.1145/508530.508540.

[269] Richard Wettel and Michele Lanza. “CodeCity: 3D Visualization of Large-scale Software”. In: *Companion of the 30th International Conference on Software Engineering*. ICSE Companion ’08. ACM, 2008, pp. 921–922. ISBN: 978-1-605580-79-1. DOI: 10.1145/1370175.1370188.

[270] Richard Wettel and Michele Lanza. “Program Comprehension Through Software Habitability”. In: *Proceedings of the 15th IEEE International Conference on Program Comprehension*. ICPC ’07. IEEE, 2007, pp. 231–240. ISBN: 978-0-769528-60-1. DOI: 10.1109/ICPC.2007.30.

[271] Richard Wettel and Michele Lanza. “Visual Exploration of Large-scale System Evolution”. In: *15th Working Conference on Reverse Engineering*. WCRE ’08. IEEE, 2008, pp. 219–228. ISBN: 978-0-769534-29-9. DOI: 10.1109/WCRE.2008.55.

[272] Richard Wettel and Michele Lanza. "Visualizing Software Systems as Cities". In: *4th International Workshop on Visualizing Software for Understanding and Analysis*. VISSOFT '07. IEEE, 2007, pp. 92–99. DOI: 10.1109/VISSOF.2007.4290706.

[273] Richard Wettel, Michele Lanza, and Romain Robbes. "Software Systems As Cities: A Controlled Experiment". In: *Proceedings of the 33rd International Conference on Software Engineering*. ICSE '11. ACM, 2011, pp. 551–560. ISBN: 978-1-450304-45-0. DOI: 10.1145/1985793.1985868.

[274] Jarke J. van Wijk. "Views on Visualization". In: vol. 12. TVCG '06. IEEE, 2006, pp. 421–432. DOI: 10.1109/TVCG.2006.80.

[275] Jarke J. van Wijk and Huub van de Wetering. "Cushion Treemaps: Visualization of Hierarchical Information". In: *Proc. IEEE Symp. on Information Visualization*. InfoVis '99. IEEE. 1999, pp. 73–78. DOI: 10.1109/INFVIS.1999.801860.

[276] Ludwig Wittgenstein. "Tractatus Logico-Philosophicus". In: *London: Routledge, 1981* (1922). Ed. by D.F.Pears.

[277] Francis Wolinski. "Visualization of Diseases at Risk in the COVID-19 Literature". In: abs/2005.00848 (2020). DOI: 10.48550/ARXIV.2005.00848.

[278] Jo Wood, Petra Isenberg, Tobias Isenberg, Jason Dykes, Nadia Boukhelifa, and Aidan Slingsby. "Sketchy Rendering for Information Visualization". In: *IEEE Trans Vis Comput Graph.* TVCG '12 18 (2012), pp. 2749–2758. ISSN: 1077-2626. DOI: 10.1109/TVCG.2012.262.

[279] Yingtao Xie, Tao Lin, Rui Chen, and Zhi Chen. "Toward Improved Aesthetics and Data Discrimination for Treemaps via Color Schemes". In: *Wiley Color Research & Application* (2017). ISSN: 1520-6378. DOI: 10.1002/col.22196.

[280] Yumi Yamaguchi and Takayuki Itoh. "Visualization of Distributed Processes using "Data Jewelry Box" Algorithm". In: *Proceedings Computer Graphics International*. 2003, pp. 162–169. DOI: 10.1109/CGI.2003.1214461.

[281] Ji Soo Yi, Youn ah Kang, John Stasko, and J.A. Jacko. "Toward a Deeper Understanding of the Role of Interaction in Information Visualization". In: *IEEE Trans Vis Comput Graph.* TVCG '07 13.6 (2007), pp. 1224–1231. DOI: 10.1109/TVCG.2007.70515.

[282] E.F.Y. Young, C.C.N. Chu, and Z.C. Shen. "Twin binary sequences: a nonredundant representation for general nonslicing floorplan". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 22.4 (2003), pp. 457–469. ISSN: 0278-0070. DOI: 10.1109/TCAD.2003.809651.

[283] Silvio Zanola, Sara I. Fabrikant, and Arzu Çöltekin. *The Effect of Realism on the Confidence in Spatial Data Quality in Stereoscopic 3D Displays: (refereed Extended Abstract).* Geographisches Institut, 2009.

[284] Jin Zhang. "The Implication of Metaphors in Information Visualization". In: *Visualization for Information Retrieval*. Vol. 23. Springer, 2008, pp. 215–237. ISBN: 978-3-540751-47-2.

[285] Mengjie Zhou, Wenqing Hu, and Tinghua Ai. "Multi-level thematic map visualization using the Treemap hierarchical representation model". In: *Journal of Geovisualization and Spatial Analysis* 4 (2020), pp. 1–11.