Informatics

Immersive Exploration of Hierarchical Networks in VR

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Manuel Eiweck
Matrikelnummer 01633012

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Ass. Dr.techn. Manuela Waldner, MSc.
Mitwirkung: Dipl.-Ing. Dr.techn. Johannes Sorger
Dipl.-Ing. Wolfgang Knecht

Wien, 16 April, 2021

Manuel Eiweck   Manuela Waldner

Informatics

Immersive Exploration of Hierarchical Networks in VR

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Manuel Eiweck
Registration Number 01633012

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Ass. Dr.techn. Manuela Waldner, MSc.
Assistance: Dipl.-Ing. Dr.techn. Johannes Sorger
Dipl.-Ing. Wolfgang Knecht

Vienna, 16th April, 2021

Manuel Eiweck   Manuela Waldner

Erklärung zur Verfassung der Arbeit

Manuel Eiweck

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 16 April, 2021

Manuel Eiweck

Danksagung

Einen besonderen Dank möchte ich den Betreuern dieser Arbeit Johannes Sorger, Wolfgang Knecht sowie Manuela Waldner aussprechen welche mich in der Entwicklungsphase dieser Bachelorarbeit tatkräftig unterstützt haben. Weiters gilt mein Dank ebenfalls meinen Freunden Thomas Jakli und Lars Müller welche mir Feedback zur implementierten Visualisierung gegeben sowie beim Erstellen der Präsentations Website geholfen haben. Außerdem möchte ich mich bei all meinen Studienkollegen und Studienkolleginnen im speziellen Pascal Hann und Aaron Wedral bedanken, ohne denen mein Studium sicherlich nicht so erfolgreich und unterhaltsam gewesen wäre. Zu guter Letzt bedanke ich mich bei meinen Eltern sowie meiner Schwester für die großartige Unterstützung in all diesen Jahren.

Kurzfassung

Unsere Welt wird von Tag zu Tag digitaler und vernetzter. Dadurch wächst die Menge und Komplexität der Daten stätig an. Eine Analyse dieser Daten birgt großes Potential für die Wissenschaft und Industrie. Ein Teilgebiet dieser Daten ist bereits in Form von hierarchischen Netzwerken strukturiert, oder kann mittels Clustering-Algorithmen in hierarchische Ebenen unterteilt werden. Einige Anwendungsgebiete sind beispielsweise die medizinische Forschung. Hier werden Verbindungen, Gruppen und Cluster Zugehörigkeiten von Krankheiten untersucht. In der sozialwissenschaftlichen Forschung kommen hierarchischen Netzwerke in Organigrammen zum Einsatz. Aber auch in Teilbereichen der Informatik, wie Build-, Dependency- und Source Code Versionierungs Management Software, finden sich hierarchische Verbindungen bei Abhängigkeiten von Software Modulen, Versionen und multilayered Software Architektur.

Allerdings wird das strukturierte Analysieren dieser komplexen und großen Datenmengen mit klassischen zweidimensionalen Visualisierungen zunehmend schwieriger. Daher werden neue Methoden und Techniken benötigt, um den Analyseprozess zu optimieren. In dieser Bachelorarbeit untersuchen wir einen neuen Ansatz um hierarchische Netzwerke zu visualisieren. Dabei erweitern wir bisherige zweidimensionale Konzepte von hierarchischen Netzwerken mit einer dritten Dimension und visualisieren das Ergebnis mittels eines Virtual Reality Systems. Virtual Reality bietet viele Vorteile wie beispielsweise einen verbesserten räumlichen Eindruck sowie Interaktionsmöglichkeiten mittels raumfüllenden VR Tracking Systemen. Die Arbeit unterliegt der Annahme, dass es mit den Vorteilen von Virtual Reality gelingen kann, Visualisierungen zu erstellen welche die Fähigkeiten von 3D Informationsvisualisierung besser ausschöpfen können. Dadurch soll es möglich sein größere und komplexere hierarchische Netzwerke besser als mit herkömmlichen zweidimensionalen Visualisierungen zu analysieren.

Abstract

Our world is becoming more digital each year, new parts of our daily life become connected and the amount and complexity of the produced data increases steadily. The analysis of this data enables big opportunities for science and industry. A subset of this data is organized in the form of hierarchical networks or can be transformed by clustering algorithms into hierarchical layers. We see this in multiple application domains for example medical research where connections, group and cluster memberships of diseases are tracked; social science where relationships are mapped in company organization charts; in software engineering in the form of build-, dependency- and source code version management software with hierarchical connections between software modules, versions and layered software architecture.

However, getting insight into this complex data with traditional two-dimensional visualization is getting more difficult as the visual clutter increases significantly with the exponentially growth of data we saw in recent years. Therefore, we need new methods and techniques to facilitate and expedite the analysis process. In this thesis, we investigate a new approach to visualize hierarchical network data by extending already existing concepts of two-dimensional hierarchical network visualizations with a third dimension and applying it to a virtual reality based visualization system. We believe that the capabilities of virtual reality devices, such as improved spatial impression and interaction possibilities by room-scale tracked headsets and controllers allow the visualization to fully utilize the benefits of three-dimensional information visualization. Therefore, it should be possible to analyze even bigger and more complex hierarchical networks than currently possible with conventional two-dimensional visualizations.

Contents

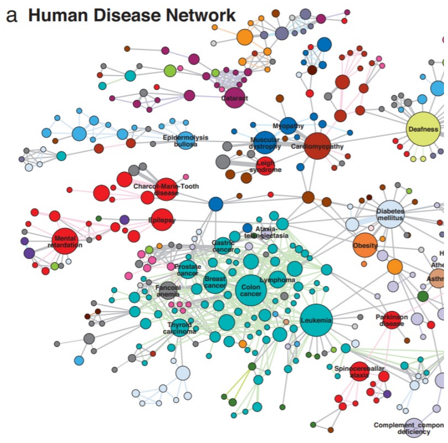
CHAPTER 1

Introduction

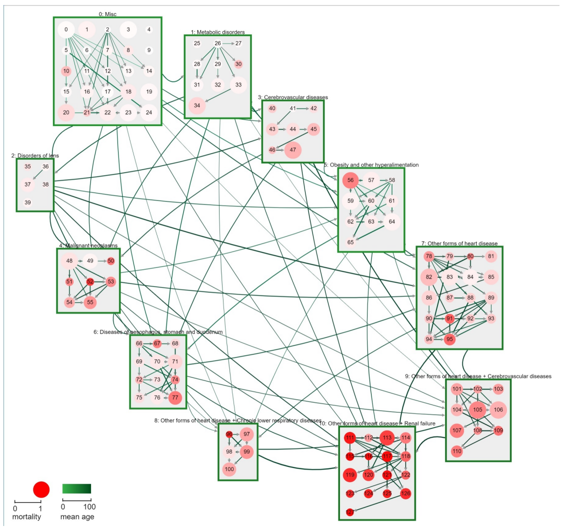
1.1 Motivation

Revealing the structure of hierarchically organized data is a complex task where many approaches already exist. It is still an active research area as data scientists are facing ever greater challenges when analyzing large hierarchical datasets. One example is the Human Disease Network [ZMBS14], which is a hierarchical network of disorders and disease genes with approximately 3000 nodes on two different hierarchy layers. Figure 1.1a shows the representation of the data in a common two-dimensional layout. In this representation, the classes of disorders are coded by color. Another dataset with a similar structure can be seen in Figure 1.1b. Here, the clusters of different diseases are grouped by boxes. Depiction of hierarchical network structure in 2D faces researches with many challenges. The biggest one is visual clutter and the so-called hairball effect (see Figure 1.1c), which reduces the legibility of the network information. Therefore, it is difficult to determine node-node relationships and hierarchical associations, as well as hierarchical relationships, which are crucial in analyzing hierarchical network data. In Figure 1.1a, we can see visual clutter on the green nodes. The graph in Figure 1.1b solves this problem by hiding links between child nodes from different boxes which is also not optimal.

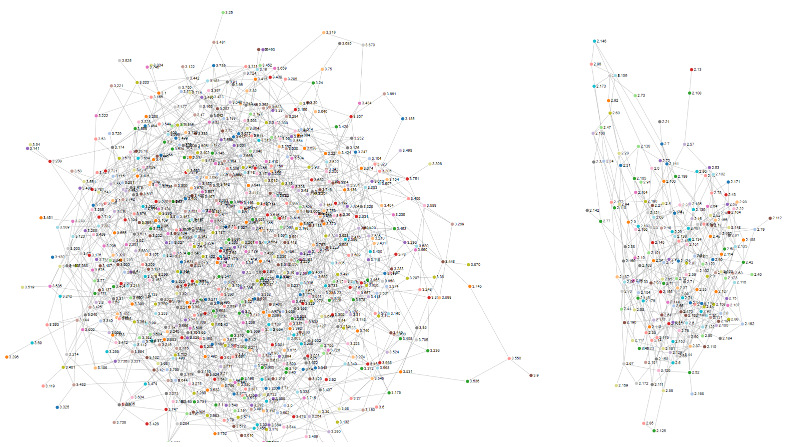
3D information visualization allows us to expand the user experience of traditional 2D graphs. As Brath describes in his paper [Bra14], there are many advantages and opportunities when using three-dimensional visualizations. By using an additional axis, the visualization has more space to distribute all the data and reduce clutter in the first place before applying specialized layouts. Kotlarek et al. [KKM+20] found out, that the mental model of the data gets improved which leads to a better understanding of the network structure when viewing the visualization in an



(a) Human Disease Network [ZMBS14]. Different hierarchical clusters are visualized by different colors.


0: Misc
1: Metabolic disorders
3: Cerebrovascular diseases
5: Obesity and other hyperalimentation
7: Other forms of heart disease
mortality
mean age

(b) A Comorbidity network of diagnoses related to diabetes melitus [SWKA19]. The boxes represent the first hierarchical layer and spheres the second layer.



(c) A part of a generated node-link graph with about 1000 nodes and 2000 links on 4 hierarchical layers. The color of the nodes represents their hierarchical parent node.

Figure 1.1: Example visualizations of hierarchical network datasets in 2D. The hierarchical structure can be data inherent or generated with clustering algorithms.

immersive environment. Greffard et al. [GPK12] compared classical 2D visualization with visualizations that use 3D stereoscopic perception. They found out that, while 2D visualizations work better for small and simple structures, 3D stereoscopic visualizations outperform classical 2D visualizations for large and more complex structures. A similar finding was made by Ware and Franck [WF96], they state that a stereoscopic viewing can increase the size of a visualized graph by a factor of three. Some visualizations also use the additional axis to encode an extra attribute by its position. This can be seen in the “3D space time cube” [Bra14]. Here, the temporal information is encoded on the Z axis. Another common usage is a 3D scatter plot. In three-dimensional layouts, the perspective of the visualization has to be reconsidered. In 2D, the user looks from the outside with a bird’s eye view at the visualization. For 3D, visualizations the user can be placed right inside our graph. This enables us to utilize new navigation and interaction possibilities.

Still, all these opportunities also involve new challenges: navigation inside a hierarchically structured 3D scene, occlusion of elements from a 3D perspective, selection of objects for displaying details and the lack of a reference point in an abstract three-dimensional space. Brath [Bra14] already stated that immersive interfaces could help to overcome these issues. We believe that the recent developments in virtual reality hardware and frameworks offer great potential to address these challenges. We can see the benefits of VR based visualizations in various publications. Marriott et al. [MCH$^+$18] analyzed multiple advantages for 3D immersive visualizations, including an additional visual channel. Bowman and McMahan [BM07] examined the impact of VR techniques, like stereoscopic images, interaction with the virtual world and head movement, can have on users. Recently Kraus et al. [KWO$^+$20] did a study on the effect of immersion for detecting clusters in a scatter plot. Their results show that VR based visualization systems have real world advantages in terms of time needed to get an overview of the data compared to traditional 2D- and 3D information visualization.

1.2 Aim of the Work

The goal of this thesis is to implement a visualization for hierarchical network data that allows users to take the benefits of 3D information visualization and opportunities of virtual reality technology to dive right into the data. Our hypothesis is that the combination of both concepts complement each other well because VR based technology can be used to solve the subsequent challenges from 3D information visualization and therefore enable data scientists to optimize their data analysis tasks. Our aim was to design a customized graph layout that calculates the position and size of each node by the hierarchical structure: child nodes are nested within their parent node. Visual clutter caused by overlapping of nodes and links should be prevented. Furthermore, the exploration of the resulting graph should be possible with room scale virtual reality devices like the HTC Vive, requiring optimized navigation, filtering and brushing methods to enhance the graph exploration user experience.

1.3 Methodology

In order to achieve the proclaimed goal, we first investigated already working solutions. We summarized important techniques and categorized them into different visualization types in Section 3.1. Before improving a visualization, we had to get a clear understanding of our data structure. Therefore, we looked into the theory behind it. Section 2.1 explains the concepts of different graphs data structures, their extensions like trees but also fundamental drawing techniques like node-link diagrams. As we were planning a VR visualization, we also looked into VR technology from an applications point of view in Section 2.2. Furthermore, we examined other related works in the field of VR visualization including navigation and interaction methods as well as VR-specialized graph layouts in Section 3.2.
With our goal in mind, we defined requirements the visualization has to meet. These requirements can be found in Section 4.1. To design our own visualization system, we need a concept for our VR specialized layout which can be found in Section 4.2. Its task is to find the node positions for a distributed hierarchical nested graph with no node overlaps, which is primarily achieved through a customized force system. Furthermore, a fitting render representation had been designed (see Section 4.3) as well as VR optimized interaction (see Section 4.4) and navigation methods (see Section 4.5). A detail description on the implementation process can be found in Chapter 5. The resulting visualization can be seen in Figure 1.2.
For evaluation, we measured performance and technical limits of our solution in Section 6.1. In addition, we gathered informal feedback on the usability in the form of an interview and an online form in Section 6.2. Lastly, in Chapter 7 we discuss possible future work.

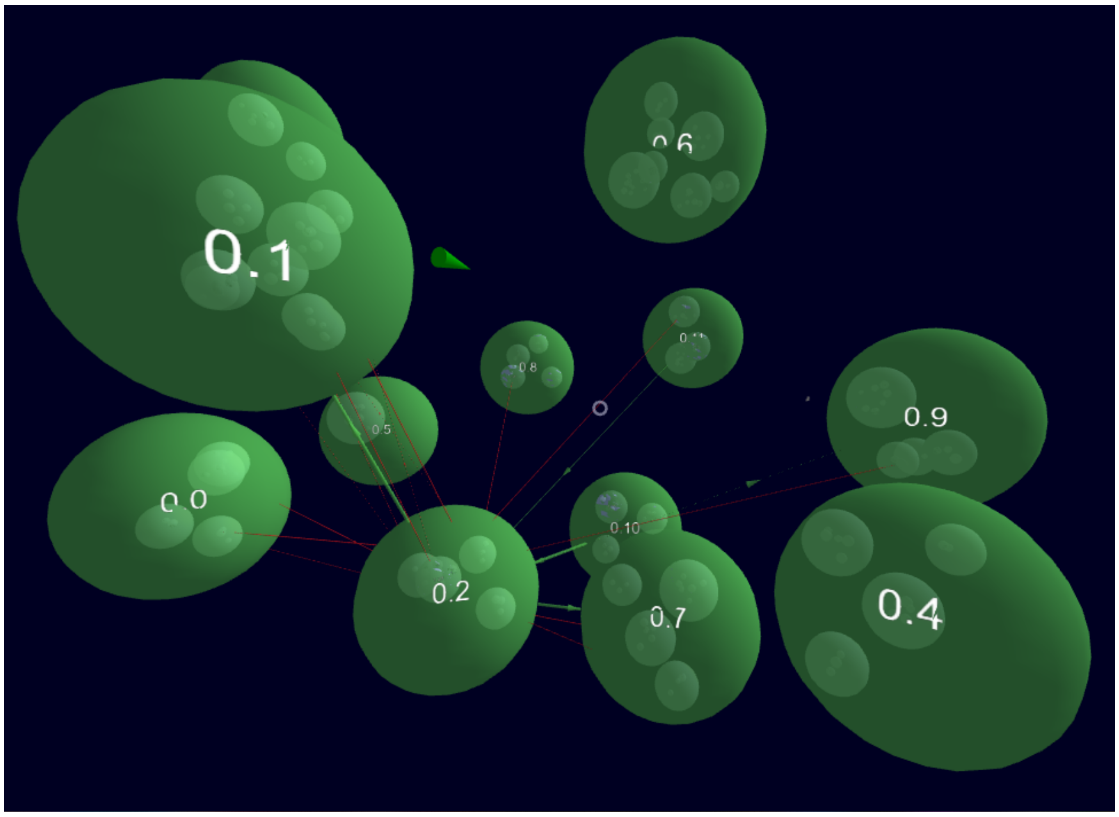


Figure 1.2: Screenshot of the generated node-link graph with about 1000 nodes and 2000 links on 4 hierarchical layers from Figure 1.1c in our hierarchical network visualization. In Figure 1.2 only edges of node 0.2 and their direct childnodes are displayed.

CHAPTER 2

Background

In this chapter we want to give a brief introduction to graph data in general. In addition, early VR approaches and some fundamental concepts and are presented in Section 2.2.

2.1 Graph Theory

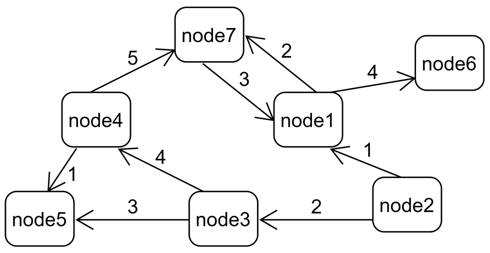
Before we begin discussing different visualization techniques, it is important to understand the data behind the visualization. As we are not limiting ourselves to a specific application domain, the key is to understand the structure of the data. Generally speaking, we are dealing with graph or network data, but as there are multiple terms and terminologies used, we want to give a short summary.

In general, a graph consists of vertices and edges [Die17]. In the context of this thesis, we use the terms ’nodes’ and ’vertices’, as well as ’links’ and ’edges’, and ’network’ and ’graph’, interchangeably. Edges in a graph can either be directed or undirected. Directed edges differentiate between source and target node where undirected edges do not. Additionally, we also distinguish between weighted and unweighted edges. Weighted edges have a sort of numerically comparable attribute. This can be used, for instance, to describe how strong the relation between nodes is. Figure 2.1a shows a directed and weighted graph. Furthermore, multivariate networks [KPW14] combine the data types of networks with multidimensional datasets. Each node and edge can have numerous additional attributes, these can be numerical, ordinal or categorical.

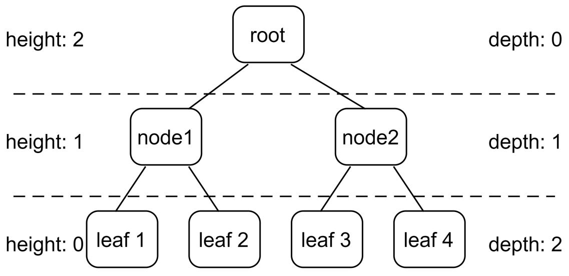
A specialized version of a graph is called tree. It is defined as an acyclic graph where all subcomponents of the graph are connected [Die17]. One node of the tree can be picked as a root node, which is usually drawn on the top. Nodes with only one link

are called leaf nodes and drawn on the bottom (see Figure 2.1b).
In addition, tree nodes also have a height and depth. The height is defined by the largest number of edges between the root node and a leaf node. The height of the root node is the height of the tree itself. The depth of the node is the number of edges along the path to the root node. Note that a tree is able to encode hierarchical relationships between nodes. In the thesis, we use the term ’hierarchical level’ for depth and terminologies like ’leaf1 is a direct child of node1’ or ’node1 is the parent (node) of leaf1’ based on Figure 2.1b.

In addition to the basic graph structures, which are often not sufficient for specific datasets, there are extended forms of graph models [BM99]. For the context of this thesis, clustered graphs are especially interesting. A clustered graph is defined by a recursive structure where each node can be new graph for itself [EF97]. Cluster information can be data inherent, which means that the cluster affiliation is encoded in the data itself. In addition, cluster information can be artificially created by hierarchical clustering algorithms. These clusters, then build hierarchical relationships in the form of tree structures. Therefore, all big graphs can be visualized by hierarchical clustered graphs [Veh15]. The combination of clustered and hierarchical graph data is also referred to as a compound graph.



(a) Weighted and directed graph.



(b) Tree with a height of 2.

Figure 2.1: Schematic depictions of graph data structures.

In conclusion, the most basic graph or network would be a simple unweighted and undirected set of nodes and links without any hierarchical relationship. To be able to represent different types of relationships in our data there exist different extensions of the core concept, such as trees and clustered graphs. In the thesis we focus on visualizing compound graphs.

2.1.1 Graph Drawing

As graph drawing is a subarea of information visualization, common knowledge from this research area can be applied here as well, with Shneiderman’s visual information seeking mantra “Overview first, zoom and filter, then details-on-demand” [Shn96] being one of the most famous. Additionally, the Gestalt Principles can also be used in graph drawing as Kobourov et al. [KMV15] shows. Gestalt Principles can be used to produce more aesthetically pleasing visualizations by defining drawing conventions and even improve the impression of the data. Cluster affiliation of nodes can be expressed by similar colors, shapes or simply by proximity, while the form of links should be consistent without any sharp bends.

For representation of data structures, node-link and space-filling concepts are the most important ones used in this thesis. Space-filling visualizations usually use multiscale approaches to encode hierarchical information (see Figure 3.1). Node-link diagrams tend to utilize force-based or constraint-based layouts[vLKS+11]. Layout techniques are necessary to calculate the node positions. Force based layout techniques work without any domain-specific knowledge, therefore provide a flexible way to determine layouts for node-link graphs [Kob12]. Only structural information about the graph relationships themselves are used. There are a variety of different approaches, but the general concept is always the same. These systems work by providing numerous repelling and attracting forces between nodes and links that update the position of the nodes. Usually these forces are related to Hooke’s law. For example, an approach by Fruchterman et al. [FR91] uses repulsive forces between all nodes and attraction forces for linked nodes. The defined forces are then processed with a given set of nodes and links for either a limited number of iterations, a fixed time, some sort of dynamic condition (for instance the momentum of the nodes fall below a certain threshold) or just infinitely. Large graphs, however, are a common problem as physical force models tend to have many local minima and therefore do not produce optimal results [Kob12]. Instead of defining interactions between nodes and links, constraints try to model the resulting layout directly; for instance, by placing nodes of certain groups in predefined locations of the screen.

Another form of graphs are multilayer visualizations. Their goal is to combine multiple node-link diagrams, so-called layers into one visualization, including the relations between them. Source and target layer of the links are well-defined, they can either be considered as intra-layer when the source and target layer are the same or inter-layer when they are not [GMM+19]. Since the nodes on multiple layers can represent the same entity, there are no strictly defined parent or child nodes in multilayer networks, therefore inter-layer links can be modeled between all layers. When dealing with multiple graphs, there is a group structure that models the relations between nodes of different graphs. Vehlow et al. [Veh15] differentiate group structures, in flat and hierarchical, as well as disjoint and overlapping aspects. In overlapping datasets one node can be in multiple graphs at the same time. For example, a single person can be in a network of family-friends (layer/graph 1) and in a network of social-media friends (layer/graph 2). In disjoint datasets, nodes are only part of one graph. These makes the visualizations more flexible to use.

In our work we are dealing with a hierarchical and disjoint dataset. The hierarchical information can come from various clustering methods or be data inherent.

2.2 VR Technology

Virtual Reality is by far not a new idea. In 1965, Sutherland [Sut65] proposed an article of possibilities of future displays, including the “ultimate display” which allows a user to fully immerse in a virtual reality. Later, in 1992 we saw this concept as an early adoption by Cruz-Neira et al. [CNSD$^+$92] in the form of “The Cave”. It was a system with a head-mounted display which even allowed viewpoint tracking. The name giving “cave” was a setup of multiple screens driven by projectors placed on the walls, ceiling and floor around the user. The additional displays allowed multiple users at once to experience virtual reality.

Today, VR systems usually consist of a head mounted display (also called VR headset) and a pair of controllers, one for each hand. Degree of Freedom (DoF) describes how many geometric aspects are captured and can be used. Headsets with a DoF of three, like the Google Cardboard or Oculus Go, only allow tracking of rotational movement, e.g., looking around. 6-DOF Headsets, like the HTC Vive and Oculus Quest, also allow tracking translational movement and therefore enable the user to walk around in a virtual scene. While rotational tracking can be accomplishment by using gyroscopes inside the headset, translational tracking works either inside-out or outside-in. On inside-out concepts, the cameras and/or sensors are located on the headset itself as seen on the HTC Vive and Oculus Quest. Outside-in techniques instead place the cameras and/or sensors on stationary points in the room. Some examples are the original Oculus Rift and Playstation VR. Some headsets use infrared emitters and sensors to determine the position while others rely on multiple camera feeds. Giving all these different tracking capabilities of headsets, from a software frameworks point of view, VR systems are usually split into room-scale experiences where all tracking features are available, seating or standing experiences with no translational tracking and experiences with no user interaction and only rotational tracking for example a 360-degree video. In this work, we are interested in 6-DOF Headsets that support room-scale as well as seated and standing experiences.

CHAPTER 3

Related Work

In this chapter we want to summarize important techniques related to our work. We separated these techniques into general methods that are used in graph visualization (see Section 3.1) and research for VR visualizations in particular (see Section 3.2).

3.1 2D and 3D Visualizations

Although the focus of this thesis is on visualizing hierarchical networks in virtual reality, many concepts are the same or similar to its 2D or 3D counterparts. First we want to give a brief overview of related 2D and 3D approaches in the field of network visualization. Dynamic network visualization describes a research field where none or only few assumptions about the data are made. These techniques often employ force-based layouts using node-link diagrams as we already summarized in the background Section 2.1.1. Furthermore, we want to discuss approaches dealing with hierarchical and multilayer structured datasets.

3.1.1 Hierarchical Visualizations

As we mentioned in Section 2.1, hierarchical information is encoded in the form of tree visualizations. Schulz [Sch11] presented a good overview of different tree visualization techniques. A rough separation can be made by the representation of edges which is either explicit or implicit.

Explicit approaches

In explicit visualizations, links are directly drawn as shown in Figure 2.1b. This core technique is used in numerous application domains and scientific fields. Based on that concept, researchers have developed many extensions to this approach. LensTree [SQX+06] for example, exchanges the axes and allows collapsing certain parts of the tree to display information structures like computer file directories. A classic strategy is applied by Armando et al. [AOCVMQ17]. They use radial instead of parallel axes to show the hierarchical relation. The root node is placed in the center and child nodes extend to all directions along the radius. Munzer [Mun97] extends the radial axis by a third dimension and displays the tree in the 3D hyperbolic space. Robertson et al. [RMC91] also make use of the 3D space for creating a so called cone-tree and cam-tree. These display the child nodes as a circle below or beside their parents.

Implicit approaches - space filling

Space filling approaches use the concept of nesting, which we also use in our layout. In addition, these approaches can encode a value property of the node by the size of the node (see Figure 3.1). Shneiderman [Shn92] presented this approach 1992 in the form of a tree map where he visualized the required disk space distribution of a file system. In addition to a common circle packing algorithm, Görtler et al. [GSWD18] developed a bubble tree map technique, which optimized the used space and allows encoding of additional properties via various shapes, draw strokes and colors. Sunburst charts are another common space filling technique to display node sizes in tree structures. As for 3D representations, Wang et al. [WWDW06] show a circular tree map extended to the z-axis using cylinders. Instead of cylinders, Balzer and Deussen [BD04] use nested hemispheres to visualize software structures. Other works addressed the performance of layout techniques. For example, Itoh et al. [IYIK04], presented a performance optimized technique for generating a rectangle packing tree map.

The previous presented works show that there are many approaches with various explicit and implicit layouts for tree visualizations. These visualizations focus on mapping hierarchical relationships only. However, with our approach, we also want to support data structures including links between all kind of nodes not only these covering hierarchical relationships, as long as the connected nodes have the same depth. These data structures are not supported by the presented approaches since this would create a cycle in the original data and therefore breaking the definition of a tree structure in the first place.

3.1.2 Multilayer Visualization

As already mentioned in Section 2.1.1, the interesting part of multilayer visualizations is that they enable us to model relationships between multiple separate networks, which are not explicitly supported by the common node-link and tree visualization techniques we discussed before. Domenico et al. [DDPA15] presented their MuxVis

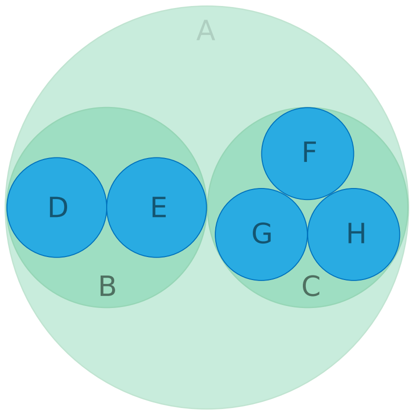
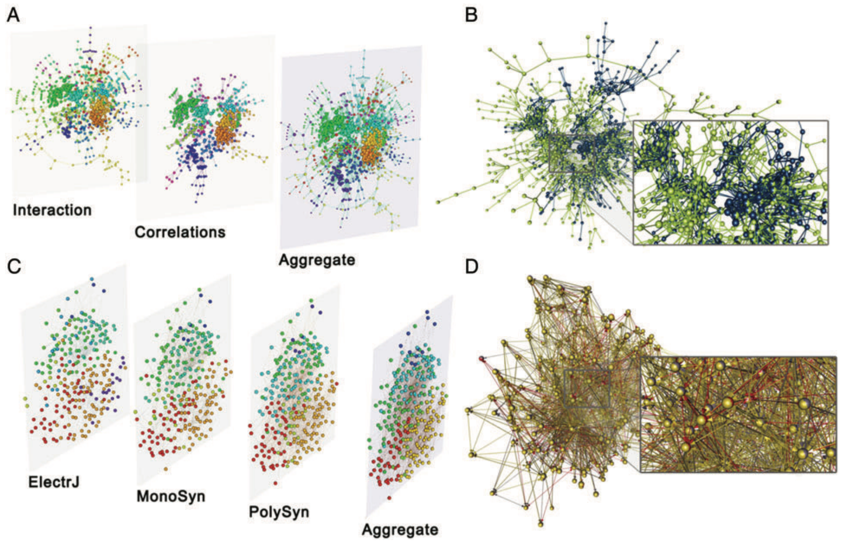
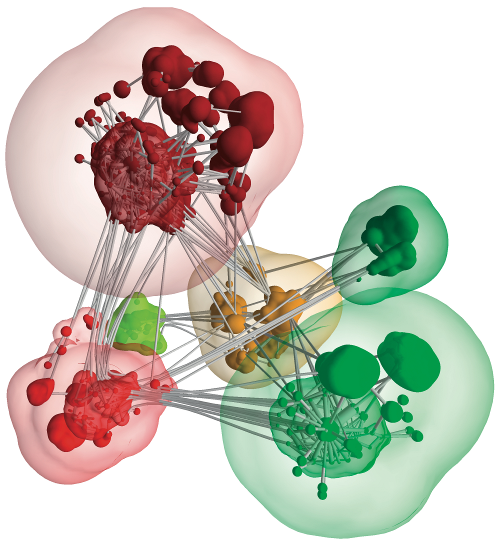


Figure 3.1: Hierarchical circle packing plot [Rib20]

Toolkit, an open-source software which is able to present multilayer datasets. It provides various layouts (see Figure 3.2a), like a classical stacked one-line layer, a multi-line layer where multiple stacked layer planes are displayed beside each other, a force directed method, and lastly a matrix layout. Besides these so-called 2.5D approaches, there are also classical 2D node-links networks, for example Ducruet [Duc17] display multiple networks showing marine traffic in a single graph just coded by color. A combination of multilayer graphs and hierarchical information can be seen in the approach by Eades and Feng [EF97]. Here, the authors group the nodes into nested clusters. These are then displayed with a classical multilayer approach as flat layers below each other. The difference to general multilayer graphs is that each node has exactly one connection to its (parent) node in the higher layer. Later on, Balzer and Deussen [BD07] (see Figure 3.2b) created a similar visualization, but instead of flat layers they used compact 3D surfaces to cluster and group their nodes. Their approach is similar to ours as it also allows nesting multiple levels of nodes and clusters. However, they do not have a strict hierarchical aspect. Jonker et al. [JLG$^+$17] use a level of detail approach where they group parts of the graph into clusters and only display single nodes when zooming into a specific tile. With their 2D layout, they are able to visualize huge networks with multiple hundred thousand nodes and links. Archambault et al. [AMA08] showed a visualization approach combining the implicit space filling technique with a hierarchical multilayer network. Their GrouseFlocks system provides interactive methods to create multiple hierarchical perspectives on the data.



(a) Different supported layout techniques by MuxVis [DDPA15].



(b) Visualization of a node-link dataset with multiple clusters from Balzer and Deussen [BD07].

Figure 3.2: Examples of multilayer visualizations.

3.2 VR Visualizations

3.2.1 Layouts

In VR, we see common 2D and 3D layout approaches like simple node-link diagrams adapted for the use in a 3D virtual scene. Some VR visualizations use a classical force-based layout to calculate their node positions [DCW+20] [SWKA19]. In other visualizations, the position of the nodes comes directly from the data. For example, Yang et al. [YCB+20] use the t-SNE method by Maaten and Hinton [MH08] to map the data attributes directly to x, y and z coordinates and display them as a point cloud scatter plot. In addition to adapted layouts, we also see some new ones specifically designed for the use in VR. Kwon et al. [KMLM16] presented a layout, which displays all nodes on the surface of a sphere. The user stands inside the sphere and is therefore able to freely look around the visualization. Another approach was presented by Halpin et al. [HZBK08]. Here, the authors display the network on a flat 2D plane in front of the viewer. Through interaction, nodes can be extruded from the flat layout and highlighted on a separated layer.

3.2.2 Navigation

The possibilities for navigation techniques in VR scenes, also called locomotion, depend on the target platform and intended experience the application is built for. As already described in Section 2.2, there are seated-, standing- and room-scale experiences differentiated by the used tracking technologies and the required space. In seated experiences, the user is usually in front of a normal office desk setup while wearing the head mounted display. Therefore, real world movement is basically no option. Standing VR setups normally do not restrict movement, but the user's available space to move is extremely limited. Rotating, leaning and hand movement should be supported, but the application can not require the user to walk around. In room-scale VR setups, the available space can differ greatly. For example HTC Vive requires the user to have a minimum space of $2m \times 1.5m$ while a maximum (with 4 base stations) of $10m \times 10m$ is possible. As a general rule of thumb, more available space provides more possibilities because the user can navigate through the scene just by walking. VR treadmills, in theory, provide infinite available space to walk, however they are rare and expensive, therefore not suited for our purposes. The conclusion of a study by Usoh et al. [UAW+99] emphasizes the importance of walking inside a virtual scene. They found that real walking provides a better immersive experience than virtual walking, such as a treadmill would provide and especially better than flying through the scene.

In most scenarios, however, our virtual scene is bigger than the available space in the real world. Therefore, we need techniques to navigate through and scale the scene. For seated experiences, Zielasko et al. [ZWB+17] developed a method that uses a virtual accelerator pedal tracked by a smartphone in the user's pocket and another by leaning the head back and forth. These interactions are then are used to fly through the scene. Similar to the leaning approach, Nguyen-Vo et al. [NVRS18] presented a technique, which enables leaning with the chair while seated. To achieve that, they simply used a swopper chair placed onto a Nintendo Wii Balance board, which is able to track the direction of leaning by the distribution of weight. Drogemuller et al. [DCW+20] summarized common navigation technique used by various VR applications. One of them is free flying using either gaze direction or the direction where the controller is pointed towards. Often, free flying/walking is performed with the left controller while the right can be used for rotating the view without physical rotation, this is especially useful as users often interfere with the cable of the headset while rotating. Another use of the second controller is to describe a direction vector for free flying with the help of the other controller. This is called “two handed flying”. We can see many approaches that use a free flying technique. However, the problem is that many users, especially those new to VR, experience motion sickness in the context of VR also called cybersickness [ZWB+17]. To reduce the effect, a reduction of the field of view while flying with a smooth in and out transition is used by various papers and also often already implemented in common VR frameworks. Beside flying, teleportation is among the most used navigation techniques. Usually, the user can point at a location and then press a button to teleport to that position. This has the advantage of reduced motion sickness because of the reduced animated movement. The challenge of teleportation is to select a suitable point in three-dimensional space without any object nearby. While some approaches solve this with an adjustable teleportation range, Lee et al. [LAK+20] present a method to calculate the traveling distance based on the density of the area. In a sparse environment with fewer objects nearby, the distance is higher than in a dense environment with many objects. Worlds-in-Miniature (WIM) [DCW+20] takes the concept of a “minimap” from 2D visualizations into VR by displaying the entire graph in a small scale onto an object near the user, for example the top of the left or right controller. The user can move and rotate the miniature, then select a point with the other controller, which can be mapped to the position of the real graph and furthermore be used for various interaction possibilities like teleporting. Yang et al. [YCB+20] show that locomotion can also be achieved via zooming and rotating. They use both controllers to scale, rotate and translate the entire graph at the same time, similar to the zooming and panning used in touch screen applications. This allows the user to freely position the graph and therefore implicitly move around.

3.2.3 Interaction

Depending on the application context, there are different goals for interacting with the virtual scene. For visualizations, selection of details, filtering and brushing is important as stated by the information seeking mantra (see Section 2.1.1). To achieve a precise selection in the scene, researchers have come up with different interaction techniques. The most basic approach to select nodes in the graph would be an infinite ray cast based on the controller direction. To display the detail of the selected node or other selected entity, a head-up display or popup can be used. For filtering, Drogemuller et al. [DCW+17] presented an interesting concept of a filter cube (see Figure 3.3). The cube is mapped to an additional Vive tracker. The buttons and sliders on it are used as a virtual input device activated via a virtual click on the button by a controller. The concept of a virtual toolbox or gadgets to interact with the environment however is not new. Similar approaches have been implemented in different VR games and creative applications; for example Google Blocks VR or Tilt Brush by Google. Switching between multiple tools can be achieved by simply grabbing them with a controller or putting them back in their slot or virtual bag. In addition to the classic controller tracked interaction possibilities, recent achievements in hand tracking technology make hand tracking without any additional equipment possible as new VR headsets, for example the Oculus Quest, come with builtin hand tracking. Besides builtin technologies, there are also additional sensors available that can be attached to the VR headsets like the Leap Motion Controller. Yi-Jheng Huang et al. [YFY+17] used this sensor to build a complex hand gesture system especially designed for the manipulation of a node-link graph in VR. Their gestures can be seen in Figure 3.4.

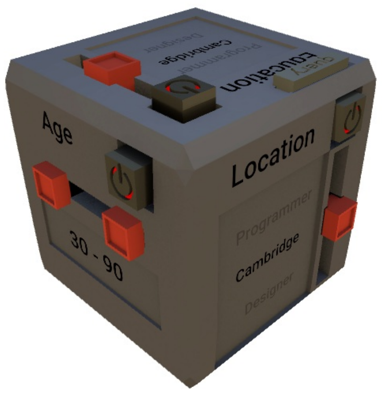
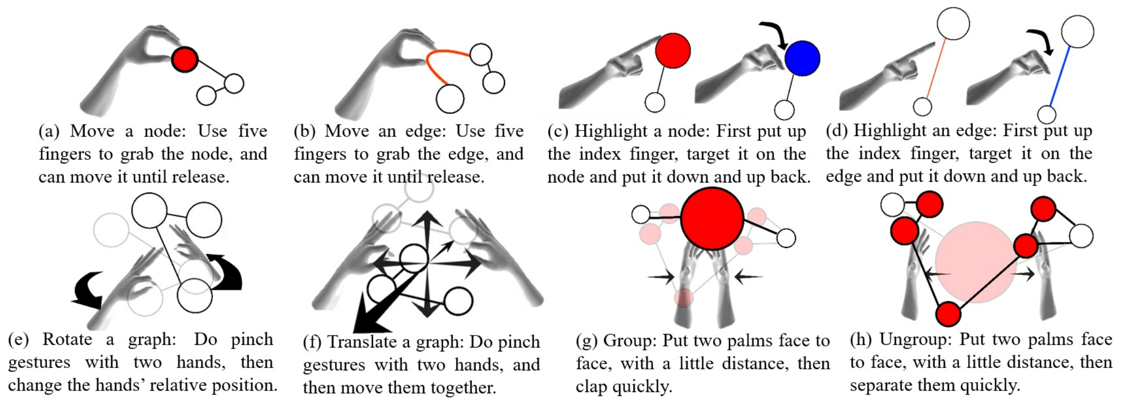


Figure 3.3: Filter cube of Drogemuller et al. [DCW+17].

3.3 Conclusion

In our work we want to visualize a hierarchical and disjoint grouped dataset in VR. Section 3.1 summarizes already existing visualization concepts for hierarchical and disjoint datasets in the 2D and 3D visualization domain. In addition, Section 3.2.1 covers layout concepts specifically designed for the use in VR. However, to our knowledge, currently there are no visualizations, in VR, for hierarchical and disjoint datasets. Therefore, we combined the explicit and implicit visualization techniques from the 2D and 3D domain, thus creating a unique layout optimized for the use in VR and hierarchical and disjoint datasets. For navigation, we use the advantage of walking inside the graph we described in Section 3.2.2 and adapted free-flying and



(a) Move a node: Use five fingers to grab the node, and can move it until release.

(b) Move an edge: Use five fingers to grab the edge, and can move it until release.

(c) Highlight a node: First put up the index finger, target it on the node and put it down and up back.

(d) Highlight an edge: First put up the index finger, target it on the edge and put it down and up back.

(e) Rotate a graph: Do pinch gestures with two hands, then change the hands' relative position.

(f) Translate a graph: Do pinch gestures with two hands, and then move them together.

(g) Group: Put two palms face to face, with a little distance, then clap quickly.

(h) Ungroup: Put two palms face to face, with a little distance, then separate them quickly.

Figure 3.4: Different possibilities of hand gestures by Yi-Jheng Huang et al. [YFY+17].

teleportation techniques for further navigation in the scene. Lastly, we apply the ray cast selection described in Section 3.2.3 to allow selection and interaction.

CHAPTER 4

Proposed Solution

Our initial motivation for this thesis was to enable visualization of hierarchical and disjoint grouped datasets in VR. Data scientists often deal with these datasets and because of the complexity of the data they often have to rely on classical 2D visualization software. However, current 2D approaches quickly reach their limits and run into the problem of visual clutter, especially when visualizing large networks and deep hierarchies. For example the network shown in Figure 1.1c reaches a critical point in visual clutter. The comorbidity network shown in Figure 1.1b hides links and is limited to only one hierarchical layer. Instead of limiting ourselves to one hierarchical layer, we wanted a solution to visualize any number of hierarchical layers and minimize visual clutter. To improve the process of hierarchical network exploration, we wanted to use the capabilities of VR and 3D information visualization. In Section 1.1, we summarized how these techniques can be beneficial for network visualizations.

4.1 Requirements

We defined multiple requirements based on the problems we described in the previous chapters:

R1 The visualization is able to visualize a hierarchical and disjoint network with depth n. Each node can be a super-node or meta-node. Links are possible within nodes of the same parent super-node but also to nodes of different super-nodes with the same hierarchical depth.

R2 The visualization can display larger networks than classical 2D approaches, before reaching a critical point of visual clutter where a meaningful exploration is not possible anymore.

R3 The visualization supports a seated, standing and room scale experience for small and large room sizes.

R4 The visualization fully utilizes the tracking capabilities of the VR Headset to optimize the interaction experience for the user.

R5 The visualization allows a flexible navigation through the graph during the exploration process.

R6 The visualization ensures a clear overview of the entire data by applying appropriated techniques according to Shneiderman's visualization seeking mantra.

R7 The visualization ensures that deep hierarchical networks can be explored using familiar interaction techniques and reduces the problem of a multi scale scene.

R8 The visualization uses the HTC Vive as a target platform, but the concepts should be transferable to other 6-DOF Headsets.

4.2 Layout

To achieve R1, we created a variation of circle packing layout approaches. The core concept is to combine the nested technique of tree maps with multilayer techniques. However instead of flat surfaces as seen in many multilayer network visualizations, we use three-dimensional spheres. By using the third dimension, we can achieve a better distribution of nodes and links and therefore display larger networks than with classical 2D approaches, as required in R2. To dynamically build this layout for any given data as input, we developed a custom force based system. The main goal is to prevent overlapping of nodes while still visualizing the different clusters of hierarchical nodes by proximity. Figure 4.1 shows a sketch of our envisioned layout. The entire data can be seen as a tree structure where each entity in the tree is a graph itself, nested in its parent node. This means that each node can be a super-node or meta-node. Therefore, the number of nodes can grow exponentially with the number of hierarchical layers. Note that the root node is not displayed because it would just create an additional complexity in the visualization without adding any meaningful benefit.

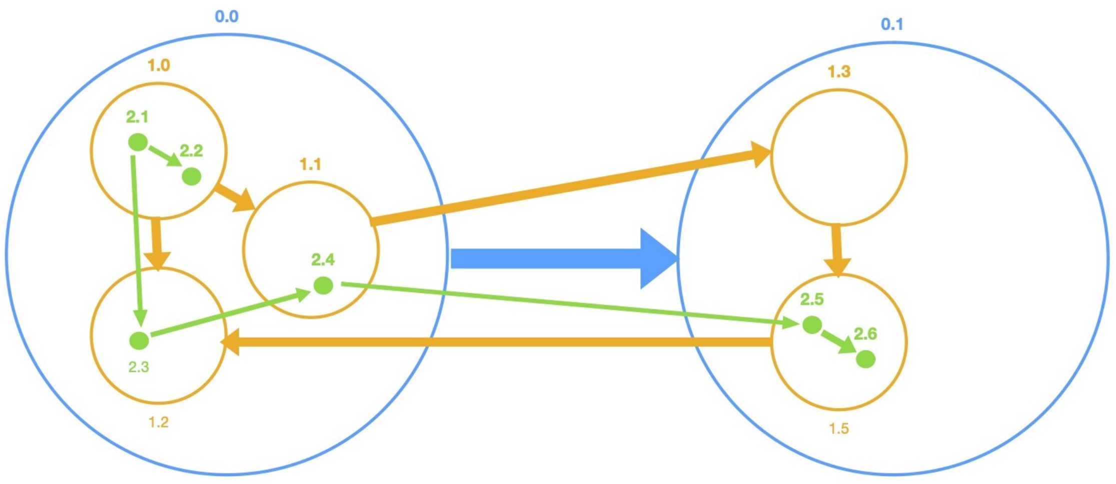


Figure 4.1: A sketch of our layout approach in 2D. Instead of 2D circles, we use 3D spheres in the real application. The size of the nodes is determined by their depth and number of child nodes. The node labels describe their unique ID. This ID consists of two parts: the first number before the dot represents the depth, respectively the layer of the node; the second part is a unique number identifying the node within one layer.

4.2.1 Layout Forces and Constraints

The force-based layout system consists of multiple forces and constraints to automatically calculate the positions of all nodes. We want to achieve an evenly distributed graph while still fulfilling the requirement rules for hierarchical nesting. Firstly, we define a set of force and constraint templates:

- ManyBody-Force: This force is a repulsion force and causes nearby nodes to be pushed away from each other. The strength decreases with the distance of the nodes. This allows the distribution of nodes evenly among the available space.
- Link-Force: It is the counterpart of the ManyBody-Force and pulls nodes connected via a link closer together. Together with the ManyBody-Force, it enables us to model a distributed graph while still clustering connected nodes and minimizing the chance for a link to cross a not involved node.
- Collision-Force: This force is basically a reinforced version of the ManyBody-Force it prevents nodes from overlapping in the case the ManyBody- and Link-Force push nodes into each other.
- Spherical-Constraint: The spherical-constraint is the core concept of our layout we use for nesting nodes. It allows us to “squish” multiple nodes into a sphere

for any given point and radius. The center of the sphere can vary for each simulation step. The constraint works by constantly adding a slightly randomized velocity for each node towards the center of the sphere. If the node is outside the sphere, then the velocity is increased drastically.

- Center-Force: This force helps us to position the entire visualization in the center of our viewpoint, however there is no strict distance or radius like the Spherical-Constraint applies.

It is important to understand that these templates are not applied equally to all nodes. Instead, we apply multiple instances of these forces with different parameters to various subsets of our graph. This separation allows us to prevent interference between different groups of nodes for different parents. Firstly, we distinguish between the top hierarchical layer (from now on called layer 0) and all other subsequent layers 1,2,3 and so on. Each node and link instance is assigned their respectively layer attribute. Layer 0 is treated separately because these nodes have no directly assigned parent node. For Layer 0, the center-force with the coordinates $(x : 0, y : 0, z : 0)$, ManyBody-Force, Collision-Force and lastly Link-Force are applied to all nodes and links with a layer 0 attribute. As for layer $1 - n$, we do not apply forces by layer but instead by parent node. For each parent, we add a Collision-Force, ManyBody-Force and Link-Force for all child nodes and links. In addition, the Spherical-Constraint is also added for all child nodes with the position of the parent node as a center. Note that the position of the parent node can change each simulation iteration, so we also update the center position for the Spherical-Constraints. In conclusion, the total number of forces and constraints in our entire force system (including all layers) is:

$$|forces| + |constraints| \ = 4 \ + 4 \cdot |parent\_nodes| \tag{4.1}$$

4.2.2 Stability of the force system

A big challenge for our customized force system was to equally balance all added forces so that each one performs their specific task and does not influence the effect of other forces. To achieve that, we parameterized each force with a strength parameter. In addition, we also use the concept of an α target from D3-Force [Bos21]. To put it briefly, it is an additional value, which decreases throughout the simulation. This allows the simulation to “cool down” and stop as soon as it reaches the defined α target value. We use the α target to control the number of simulation iterations. Besides balancing the different templates of forces, we also have to decrease the strength recursively for each layer as the nodes and their radius get smaller each layer iteration. To further improve the stability of the layout, we add a small amount of randomness to the applied velocities and strengths. This improves the node distribution and prevents scenarios where multiple nodes receive the same forces and therefore would overlap each other. During our optimization, we stumbled upon the problem that nodes tend to jump rapidly to a far position. This happens because we squish the nodes into the sphere of the parent while still applying the ManyBody- and Collision-Forces. Therefore, in some boundary scenarios, the only "free" position for this node is further away which results in the jumping behavior. However, it defeats the concept of successively fine-tuning the positions throughout the simulation steps. For example, nodes of layer 1-3 would be already positioned well but in the last simulation step the parent node in layer 0 would jump and therefore layer 1-3 nodes are positioned outside the parent layer 0 node again. To circumvent that problem, we do not perform the positioning of all nodes throughout the entire simulation. Instead we split up the amount of simulation steps and perform them for each layer successively (see Figure 4.2): beginning with positioning the layer 0 nodes, then layer 1 nodes and so on. Luckily, we also gain a performance benefit through that strategy. The reason for this is that, usually the number of nodes grows exponentially for each layer. That means, that a simulation steps for layer5 has to process more individual nodes and forces than a simulation step for layer1. By limiting the simulation steps for the larger growing layers we can reduce the total sum of position update operations.

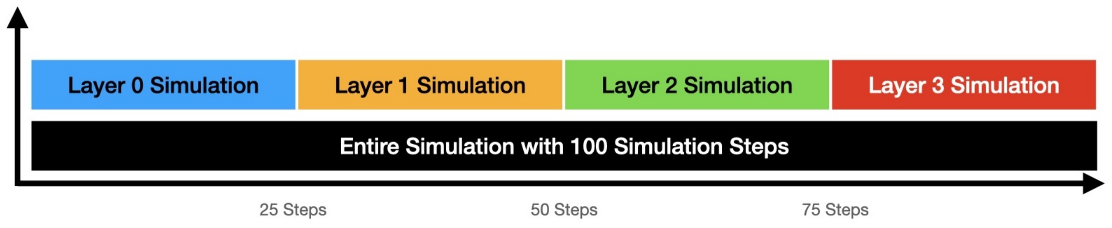


Figure 4.2: Distribution of all simulation steps for a dataset with 4 hierarchical layers. Each layer receives the same amount of simulation steps, in this case 25 of 100.

4.3 Graph Visualization

The goal of the graph visualization was to choose appropriate visual encoding which allow us to reduce visual clutter and improve clarity while still displaying as much data as possible. To represent the graph, we choose spheres for nodes, tubes as links and cones as arrow heads indicating the link direction. With solid textures, the overview ability of the visualization would be poor since our layout approach uses nested layers. Therefore, the user could only see one layer of nodes at the same time. To circumvent that we render all nodes transparently. Then, the user is able to see the nested nodes inside their parent node (see Figure 4.3). However, links inside other nodes are not visible because displaying many links would increase visual clutter and reduce overview. To deal with the large amount of links, we apply

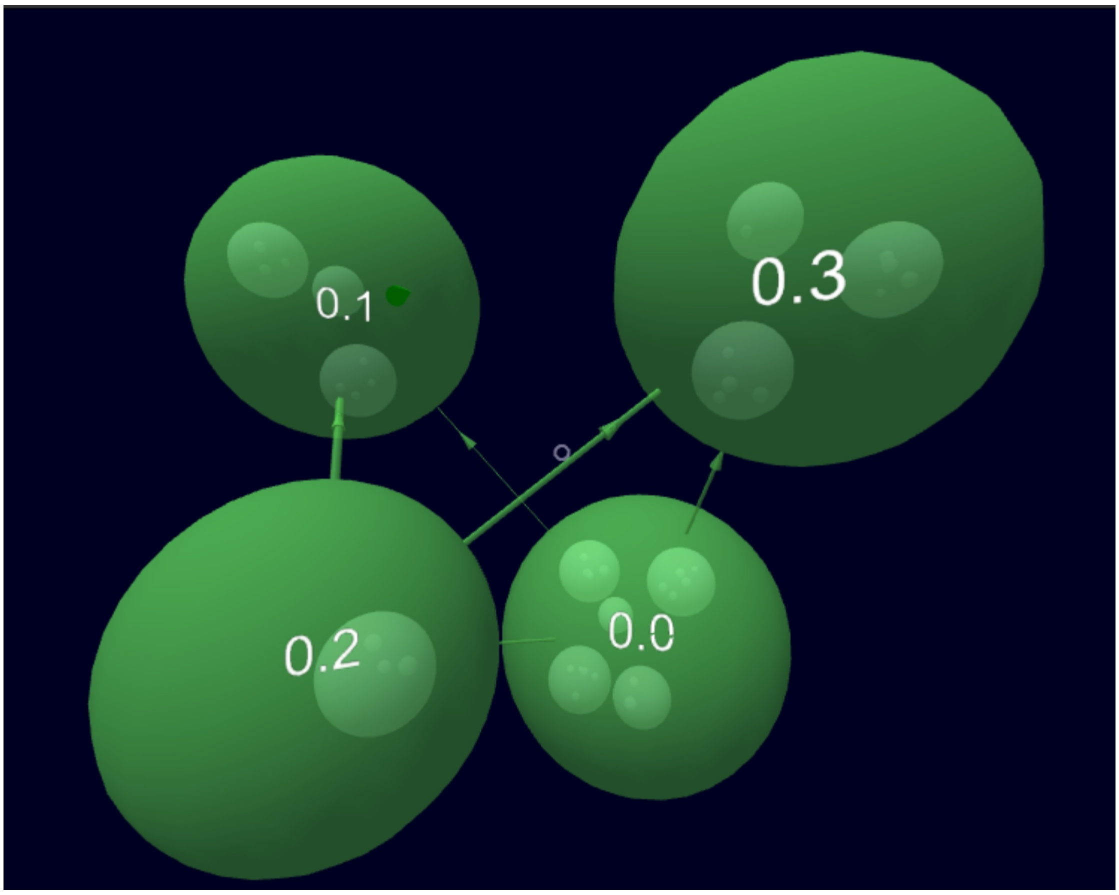


Figure 4.3: Overview perspective of the visualization: Each node is rendered semi-transparently. This allows the user to detect the overall topology of the data.

filter conditions, which are described in Section 4.4.1. When the user is inside a node, the transparent sphere of the node is not rendered anymore. Instead of a solid transparent sphere, we render a wireframe (see Figure 4.4). The improved spatial impression given by the virtual reality experience allows the user to better see the boundary of the wireframe than visible in the Figure. To further improve the assignability of nodes to their layers, each node and link of the same layer shares a predefined color. For each node instance, a billboard text label is placed on the border of the node and always points towards the camera. For our artificially dataset the ID of the node is displayed, for a real dataset it would display the appropriated node and cluster names.

Our adapted sphere packing layout algorithm requires us to render nodes and links smaller for each nested sphere. To achieve that, we added a dynamically scaling factor that is applied for each rendered entity (node, link, text-label). In addition

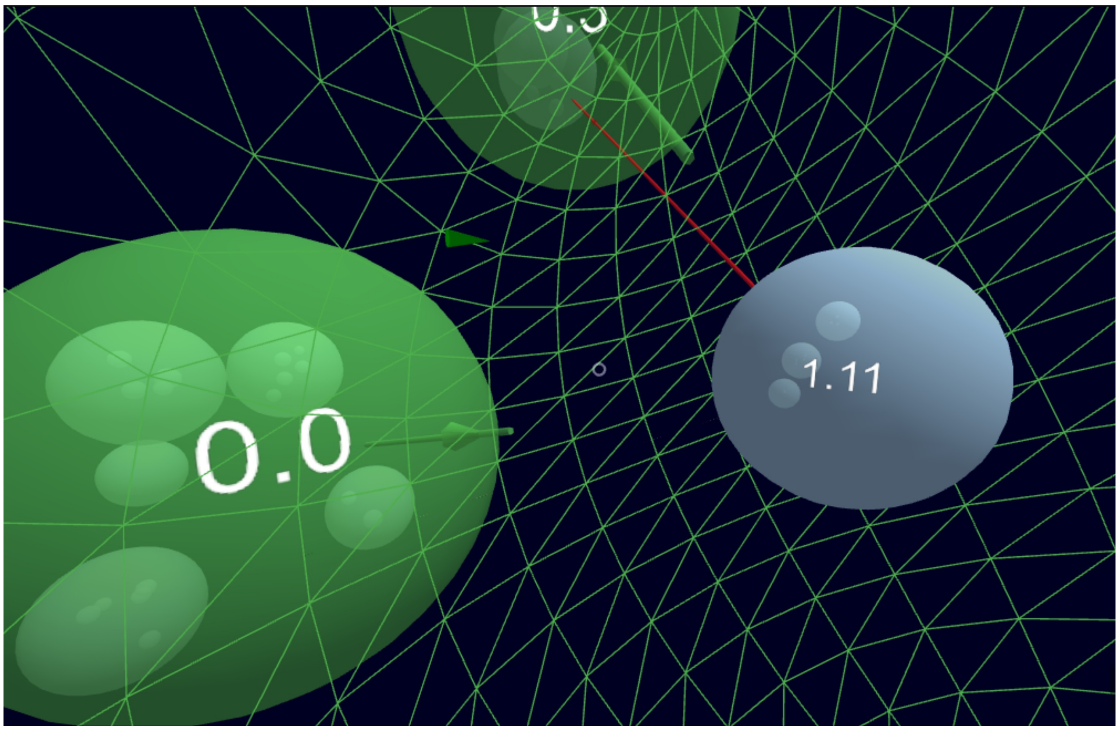


Figure 4.4: Detail perspective inside of node 0.2: To be able to tell the border of the nodes, the node is rendered as a wireframe when looking from the inside out.

to the rendering algorithm, the force algorithms also uses that scaling factor to ensure a correct collision detection and sphere packing. This globally scaling factor is described by the following formula:

$$s_d = \frac{1}{(d+1)^{(d+1)} \cdot s} \tag{4.2}$$

Where s_d is the resulting scaling factor, d is the depth of the current layer and s is a global constant that allows us to balance the force algorithm according to the rendering objects. The resulting scaling factor is used, in combination with the number of child nodes, to calculate the size of the node:

$$r_{d,i} = (r \cdot s_d) + (|N_{d,i}| \cdot rs) \tag{4.3}$$

Where d is the depth, i the node index at depth d, r is a base radius, s_d is the previous scaling factor, $|N_{d,i}|$ is the set of child nodes of the node i at depth d, and rs is a global constant to balance the size of the node radius to the according force radius.

4.4 Interaction

For achieving R4, we use the 6 DOF tracking of the headset and controllers from the HTC Vive. Therefore, we aimed to implement intuitive interaction techniques to allow teleportation navigation throughout the virtual scene (see Section 4.5). In addition, the user should be able to switch the visibility filter between multiple groups of links (see Section 4.4.1). What both interactions have in common is that the user has to select a specific node before these actions can be triggered. This concludes that our visualization needs a general way to support the selection of nodes which then can be used by various other methods we apply during the exploration process. In addition, we also stated in our requirements that we strive to use the tracking capabilities of VR to create intuitive interaction methods. Therefore, all user interactions can only be provided by the native VR supported hardware, in particular from 6-DOF tracked controllers. To this end, we use the common concept of ray cast selection for selecting nodes in the virtual scene. Along the length of the right controller, a ray is cast through the scene. To better visualize the effect, we render a straight red line to imitate a virtual laser pointer (see Figure 4.6). The first intersected node is then used as the selected node for other methods like filtering or teleportation. The node label of the selected node is displayed in a Head-up-Display element centered in the lower part of the screen inside the headset (see Figure 4.6). In addition, the selected node and directly connected nodes are visually highlighted (see Figure 4.6). This allows the users to quickly see, which node they currently have selected and provides the context of the connected nodes.

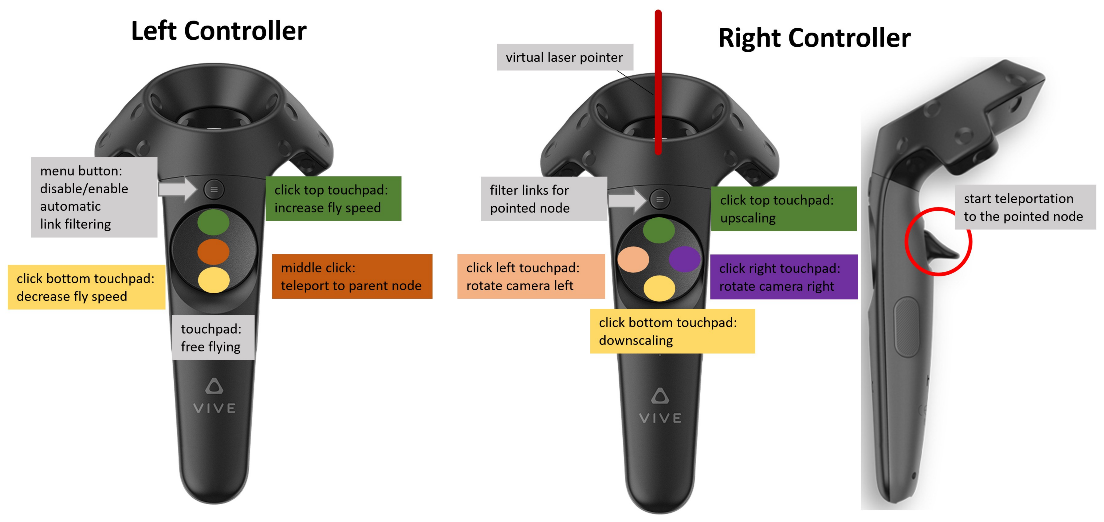

Left Controller
menu button: disable/enable automatic link filtering
click top touchpad: increase fly speed
middle click: teleport to parent node
click bottom touchpad: decrease fly speed
touchpad: free flying
VIVE
Right Controller
virtual laser pointer
filter links for pointed node
click top touchpad: upscaling
click left touchpad: rotate camera left
click right touchpad: rotate camera right
click bottom touchpad: downscaling
VIVE
start teleportation to the pointed node

Figure 4.5: Our controller mapping for the interactive VR experience. Depending on the clicking position on the touchpad, different actions can be called. The different clicking positions are marked in the image with different colors.

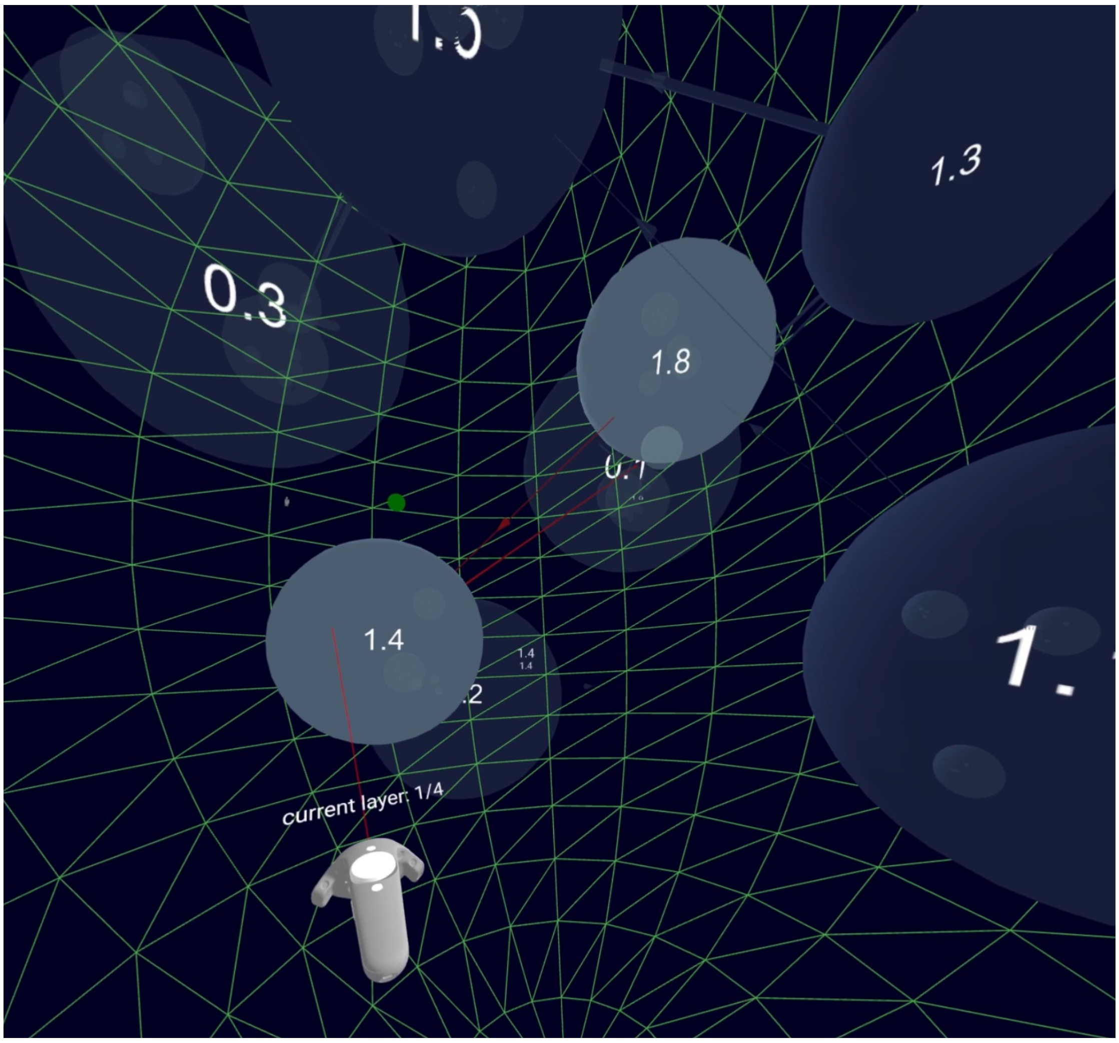


Figure 4.6: Screenshot during the selection of a node: The user can see the virtual laser pointer as a red line. In the lower part of the screen, the node label (1.4) of the pointed node is written (in this screenshot it is centered in the middle of the screenshot, because the field of view in the headset is different as in the screenshot). In addition, the selected node, as well as directly connected nodes, are highlighted in a brighter shading. For improved navigation, the layer where the user currently is located in is displayed as a text above the controller.

Now that the users can select specific nodes in the scene, we also have to give them the opportunity to trigger certain actions. These actions include the teleportation navigation (see Section 4.5) and the link visibility filter (see Section 4.4.1). Other actions do not require the selection of a node, i.e., manual rotation, free flying with an adjustable flying speed, and teleportation to an upper layer (see Section 4.5), as

well as manual scaling (see Section 4.5.1).
However, we have to assign each of these actions to a button on the controller (see Figure 4.5). In order to provide a smooth workflow, we extend the normal trackpad click by the position of clicking. This allows us to have 5 virtual buttons on the trackpad: top, bottom, left, right and center. The advantage of this method is that these virtual buttons can be used fluently while also interacting with the touch sensitive trackpad at the same time by one finger. Therefore, the users can control the free flying direction and speed with their left thumb, as well as triggering the teleportation to the upper hierarchical layer. The right controller acts as a virtual laser pointer. The two actions, which need a node selection beforehand, can be started by the menu button and trigger. On the right trackpad, the scaling and rotation can be controlled.

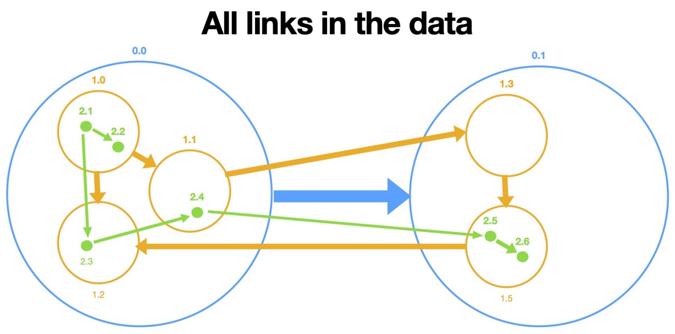
We optimized our application for the HTC Vive. Therefore, all described button mappings refer to the HTC Vive controllers. However, we only use simple buttons, trackpads and industry standard tracking methods. As other 6-DOF headsets usually provide the same or similar interaction possibilities, the interaction concept of our visualization can be applied to other headsets as well.

4.4.1 Filtering Link Visibility

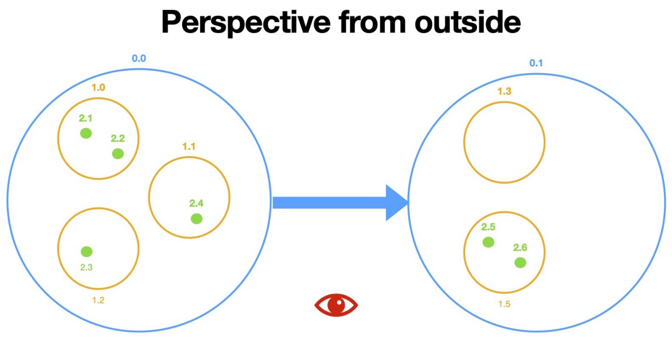
To ensure the clarity of the visualization (see R6) and display larger networks than 2D approaches (see R2), we apply a filtering approach for links according to the requirements. The goal is to reduce visual clutter while providing as much information as possible. As a quick recap: Links in our visualization can exist between nodes within the same hierarchical parent node but also to nodes with a different parent node as long as the hierarchical depth is the same. Figure 4.7a shows a small example of all possible link combinations for three hierarchical layers. Displaying all links at the same time would quickly lead to visual clutter where a meaningful exploration is not possible anymore. Therefore, we came up with our own filtering logic that decides which links are displayed and which are not. The filter uses a specific node. This can either be a directly selected node from the user (by the laser pointer interaction) or an indirectly selected node through the position of the user. A directly selected node always overrules the indirectly selected one. Our filter logic uses the following rules:

1. If no node is selected and the user is not inside any node only the links from the top layer 0 are displayed (see Figure 4.7b).

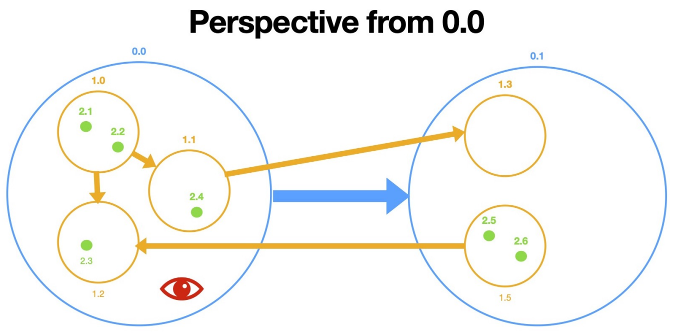
2. All directly connected links to the selected node are displayed (currently our visualization only supports links between nodes with the same depth d).



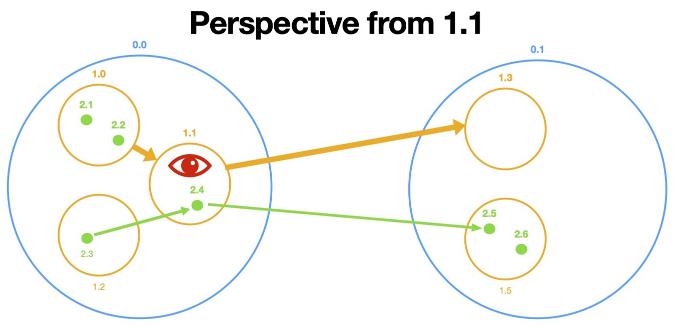
(a) All links represented in the data.



(b) Visible links when the user is outside every node.



(c) Visible links when the user is inside node 0.0 or when node 0.0 is selected by the laser pointer interaction.



(d) Visible links when the user is inside node 1.1 or when node 1.1 is selected it by the laser pointer interaction.

Figure 4.7: Our filtering technique visualized on a small example graph.

3. All links connected to direct child nodes from the selected node are displayed. Direct child means the depth of the child node is $d + 1$.

4. All other links are not visible.

As an example, Figure 4.7c shows a situation where node 0.0 is selected. Therefore, all links connecting direct neighbor-nodes of 0.0 and links to the child-nodes of 0.0 itself are displayed. However, links of deeper nested nodes, like 2.1, are not displayed. A different situation can be seen in Figure 4.7d: here only links from node 1.1 and 2.4 are visible but the links from node 1.2 and 0.0 are not visible anymore. In an earlier version, we not only displayed the links of direct child nodes but instead links of all recursive child nodes. For example, in Figure 4.7c, the green links would also be visible. However, this quickly leads to visual clutter. Therefore, we kept the direct child approach.

In order to support an intuitive and flexible exploration workflow, the user also has the ability to lock a currently applied filter. With the left menu button (see Figure 4.5), the state of the lock can be toggled. If the lock is active, the indirect selection of nodes by the position of the user is disabled. The lock is also automatically set active if a node is manually selected by the laser pointer. This allows the user to freely navigate around the visualization without constantly having to deal with changing link visibility. On freeing the lock, the indirect selection of nodes automatically kicks in again and selects the current node the user is positioned in.

4.5 Navigation

In order to fulfill R3, we support a walkable room scale VR experience setup to navigate the virtual scene. As we stated in Section 3.2.2, walking probably provides the best immersive and intuitive navigation method. However, the graph will usually be larger than the user's available space. To this end, we also include other navigation methods. To allow maximum of flexibility during the navigation, as required for R5, we provide two navigation methods: animated teleporting for covering larger distances and free flying to perform small and precise position adjustments.

The user should be able to move to other distant parts of the graph without loosing orientation or being interrupted in the exploration flow. A common example is to jump to a connected node of another parent node. Therefore, we implemented an animated teleportation method. The user can select a node with the laser pointer and then press the right trigger to initiate the teleportation. The target position is the closest edge barely inside the node. This allows the user to get an overview of all nodes and links in that selected node. As with a simple teleportation over long distances, the user might lose the orientation. Therefore, we perform a short animation to the target position. The speed of the animation is adapted with an ease-in and ease-out transition. Remember that, as soon as the user enters another node, the visibility of links changes, provided there is no active lock. Instead of teleporting deeper into the hierarchical network, the user can also teleport to the parent hierarchical layer; in particular to the area barely outside the node the user is currently located in. This teleportation can be triggered by pressing the middle of the left trackpad. All teleportation methods can be freely combined even during the animation, therefore allowing the user to fluently move around the graph. To further improve the overview, we display the current hierarchical layer as a text element floating over the right controller (see Figure 4.6).

To perform small adjustments in position, especially when the user's free space is limited, the user can freely fly in the virtual scene. To control the direction, the left touch sensitive trackpad can be used. The forward direction is linked to the user's gaze direction, so the final fly direction is the combination of gaze direction and trackpad touch position. In addition, the fly speed is adjustable with a click on top or bottom of the trackpad. Similar to other VR applications, we also implemented rotation of the entire room scale space in the virtual scene. This allows the users to better position themselves in the real world. This is particularly useful with cable bound headsets because multiple rotations in the real world require the user to step over the cable. In other VR applications, movement is usually carried out with the left controller and rotation with the right, analogous to Gamepad controls in most games. Therefore, we mapped the rotation to the right trackpad by clicking either on the left or right corner.

4.5.1 Challenge of Spatial Reference in VR

In R7, we stated that the visualization should reduce the problem of a multi scale scene. The challenge is that nodes and links are getting exponentially smaller with each hierarchy step. This introduces a new problem as after a few iterations, the nodes and links in the virtual scene are too small to recognize. 2D visualizations usually deal with that by adjusting the viewport of the visualization, e.g., zooming in and out. However, in the context of a 3D VR visualization, a simple 2D zooming approach is not possible due to the additional dimension. In a 3D non-VR visualization that problem can be mitigated by adjusting the movement speed. Therefore, distances seem similar in size for the inner nodes and navigation is also possible without overshooting the target position. In VR the situation is trickier due to the spatial impression through stereoscopic rendering. On a 2D projected image, like on a normal monoscopic computer screen, distances in the scene can only be estimated through movement. From a static image, the human eye is not able to recognize the distance or size of an object without any additional size reference. In the real world however, we constantly see objects with two eyes in a stereoscopic view, this enables the human mind to estimate distances without the need of any movement. VR works the same way by rendering the scene from two different perspectives each for one eye. This means that a simple trick of adjusting the movement speed is not working for our VR visualization. That problem makes itself noticeable, for example, as layer 4 nodes in VR are the size of a needle where layer 0 nodes are the size of an entire house. In addition to the perspective problem, changing the movement speed also does not work because movement in a 6-DOF VR headset is also always carried out by movement in the real world and obviously the physical movement speed can not be adjusted.

One solution to that problem is scaling the entire scene. Therefore, we implemented two techniques in our visualization:

- Dynamically adjusting the fly speed while navigating in the graph. On entering a node, the speed slows down. When leaving, it speeds up again.
- Dynamically scaling the entire scene. When teleporting to a node in a deeper hierarchy layer, an upscaling transition is started. On teleporting to the parent layer, a downscaling transition is started.

The challenge while scaling is that the user's relative position in comparison the to virtual scene must not change. Otherwise, the user will get distracted while exploring. When applying a scaling matrix, the center of the scene (0,0,0) stays in the same position. All other points are moved during the scaling process. Therefore, we translate the target position of the animated teleport into the center of the scene, then scale up/down and afterwards translate the scene back. In order to achieve a smooth user-friendly scaling, instead of scaling to the desired size once, we animate the scaling process. The duration of the animation has to be the exact time as the animated teleportation. Otherwise, the teleportation path will get distorted and result in a curve instead of a straight line. The automated scaling process is described by the formula:

$$S_p = T_p \cdot S_i \cdot T_p^{-1} \tag{4.4}$$

Where S_p is the scaling matrix for the target flying teleport position p, T_p is the translation matrix from the position to the center, T_p^{-1} the reverse translation matrix and S a usual 3D scaling matrix for a given scaling size i.

In addition, the scale can also be manually changed by the user at any time by pressing the upper or lower corner of the right trackpad. In R3, we stated to support small and large room sizes. However, the scale can not be optimized beforehand for a walking experience. The manual scaling gives the users the ability to adjust the scale to their available space and personal preference. In the manual scaling process we have to distinguish between the rig and camera position. The reason for this is how camera movement in VR frameworks are handled (see Section 5.5.4). The manual scaling process is described by the formula:

$$S_p = T_{p_{rig}} \cdot T_{p_{camera}} \cdot S \cdot T^{-1}_{p_{camera}} \cdot T^{-1}_{p_{rig}} \tag{4.5}$$

Where S_p is the scaling matrix for the current user's position p, $T_{p_{rig}}$ is the translation matrix from the relative rig position to the center, $T_{p_{camera}}$ is the translation matrix from the relative camera position to the center, T^{-1}_{rig} and T^{-1}_{camera} the reverse translation matrices and S a usual 3D scaling matrix for a given scaling size i.

4.6 Exploration Flow

To provide an optimal exploration experience of the hierarchical network, we use the overview and detail concept of the visual information seeking mantra (see Section 2.1.1) as stated in R6. At the beginning of an exploration session, the user is in the overview perspective. The user is placed on a further position away from the center where a good overview of the entire graph is possible. In addition, rotation to the graph can be applied. When the user decides to dive deeper into the network, a node can be selected via the ray cast controller interaction. This triggers the transition into the detail perspective. Here all navigation and interaction methods described in Sections 4.4 and 4.5 are available. A core concept for the exploration in the detail perspective is the use of the VR room-scale navigation experience. As the user is able to tweak the scale of the scene to their liking, walking around the graph is possible for different room sizes. The goal is to improve the spatial impression of the graph, which results in a better clarity and understanding of the data.

CHAPTER 5

Implementation

5.1 Programming APIs

To implement VR applications, we can work directly with the SDK provided by each headset manufacturer. The advantage is that we can interact at a low level with the devices. Another way is to use an SDK of a well-defined standard for example OpenVR from Valve [Cor21] or OpenXR from the Khronos Group [Gro21b]. This allows us to write applications for a target group of devices like 6-DOF supporting headsets without having to deal with different manufacturer SDKs (see Figure 5.1). On a side note: the X in OpenXR means that the specification is not only used for virtual reality (VR) applications but also for augmented reality (AR) and other technologies (XR) possibly proposed in the future. Lastly, we can work with different 3D graphic engines which provide various additional features. While classical desktop engines rely on downloading a complete bundled software package for distributing the application, browser based engines like A-Frame [Inc21] that use WebXR [Gro21a] and WebVR [Gro21c] allow us to distribute and run our application via the browser, similarly to WebGL.

At the time of writing, the OpenXR and WebXR specification is still relatively new being in its first revision. Therefore, it is not implemented by each headset manufacturer, browser and engine yet. However, these standards are meant to replace the old OpenVR and WebVR in the future.

5.2 Prior Publication

For our implementation, we extend the code of a prior publication from Sorger et al. [SWKA19]. They implemented an immersive visualization for node-link networks that

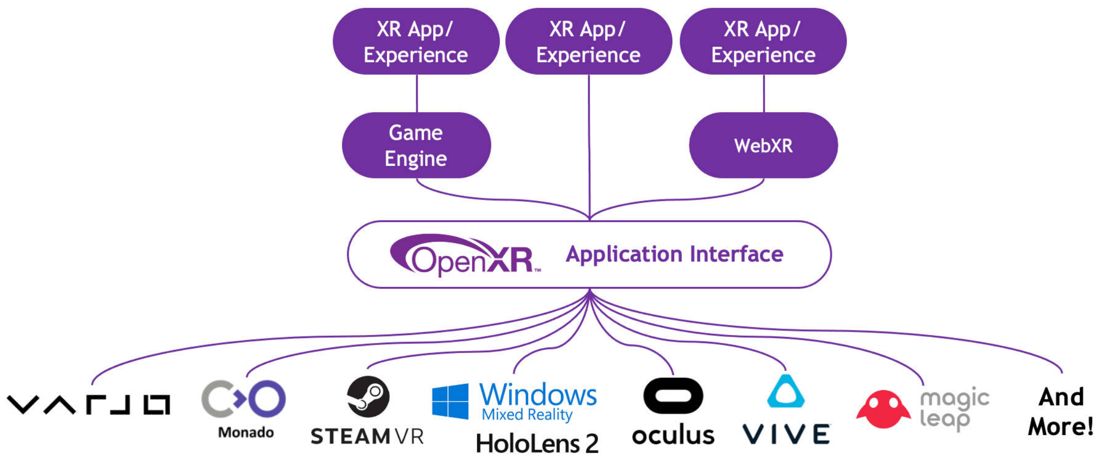


Figure 5.1: Overview of the OpenXR API Stack [Gro21b].

does not explicitly support hierarchical network data. Their implementation already contains a common force-based layout, the laser-pointer ray casting technique to select objects, free flying as well as animated teleport navigation and lastly a code structure with an included render engine. A screenshot of their work is shown in Figure 5.2.
We extended their implementation by adding the ability to visualize and interact with hierarchical networks as described in Section 4. To achieve that, we had to adapt the layout algorithm, add transparent rendering, change the target position for the animated teleport, change the controller button mappings, implement the link filtering technique, add automatic/manual scaling of the virtual scene and automatic/manual adaption of the free flying speed. This process is described in detail in Section 5.4 and 5.5.

5.3 Technology

The application is implemented in plain JavaScript and runs in browsers that support the WebVR standard. At the time of writing, this only applies to Firefox for Windows (our tested version is 85.0.2 (64-Bit)). In order to reduce the complexity for rendering and implementing the WebVR standard, the application uses the framework A-Frame [Inc21]. The code of the publication we extended uses A-Frame in version 0.9.2 from September 2019, which internally uses three.js [tosc21] in version 0.108.0 as a render engine. This old version of A-Frame brings the limitation of only supporting WebVR because WebXR was not yet ready back in 2019. Unfortunately, WebVR is already deprecated and being replaced with the new WebXR standard. While we know that it is unlikely that other browser vendors will support WebVR in the future,

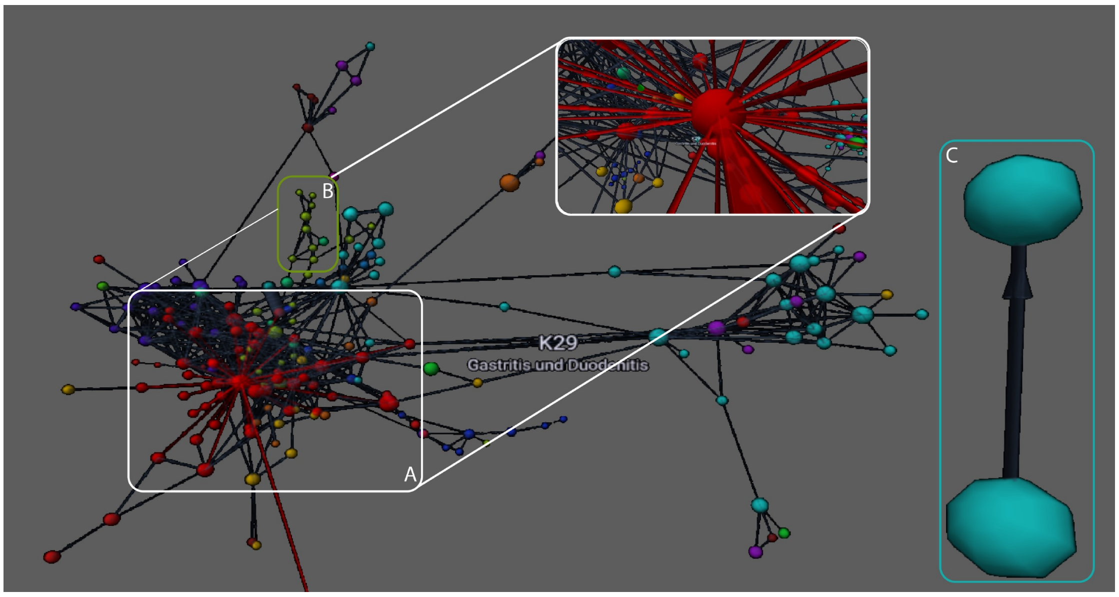


Figure 5.2: Screenshot of the visualization from Sorger et al. [SWKA19] we extend our implementation on.

we decided to not update the A-Frame version. Updating to a current version would break many features already implemented and a complete migration would not be possible within reasonable time. Therefore, we decided to stick to the old version and only support WebVR browsers and headsets. The primary VR headset we are targeting with our application is the original HTC Vive.

5.4 Application Overview

5.4.1 Data Structure

To get a better understanding of the implementation, we begin with describing the data structure for storing the graph. Listing 5.1 shows a minimalistic example of the data structure that we use an input data source. The data is formatted as a JSON object with a flat list of nodes and links. The hierarchical information is encoded within the attributes childNodesIDs and parentNodeID as references. In addition to the necessary attributes, each node and link object can have additional attributes depending on the specific dataset.

Listing 5.1: minimal JSON input data structure. The id attribute has to follow the pattern ”<layerNr>.<uniqueNodeIDByLayer>”. The desc, color and weight attribute can be set individually. Besides these required properties there can be any number of additional properties.

```
{
    "nodes": [
        {
            "id": "0.0",
            "weight": 4.5,
            "color": "rgb(77, 175, 74)",
            "layer": "0",
            "desc": "0.0",
            "childNodeIDs": [
                "1.1",
                "1.3",
                ...
            ]
        },
        ...
        {
            "id": "1.1",
            "weight": 1.7,
            "color": "rgb(250, 250, 110)",
            "layer": "1",
            "desc": "1.1",
            "parentNodeID": "0.0",
            "childNodeIDs": [
                "2.1",
                ...
            ]
        },
        ...
    ],
    "links": [
        {
            "color": "rgb(77, 175, 74)",
            "layer": "0",
            "source": "0.0",
            "target": "0.1",
            "label": "0.0 - 0.1",
            "linkwidth": 0.35
        },
        ...
    ]
}
```

5.4.2 Program Flow

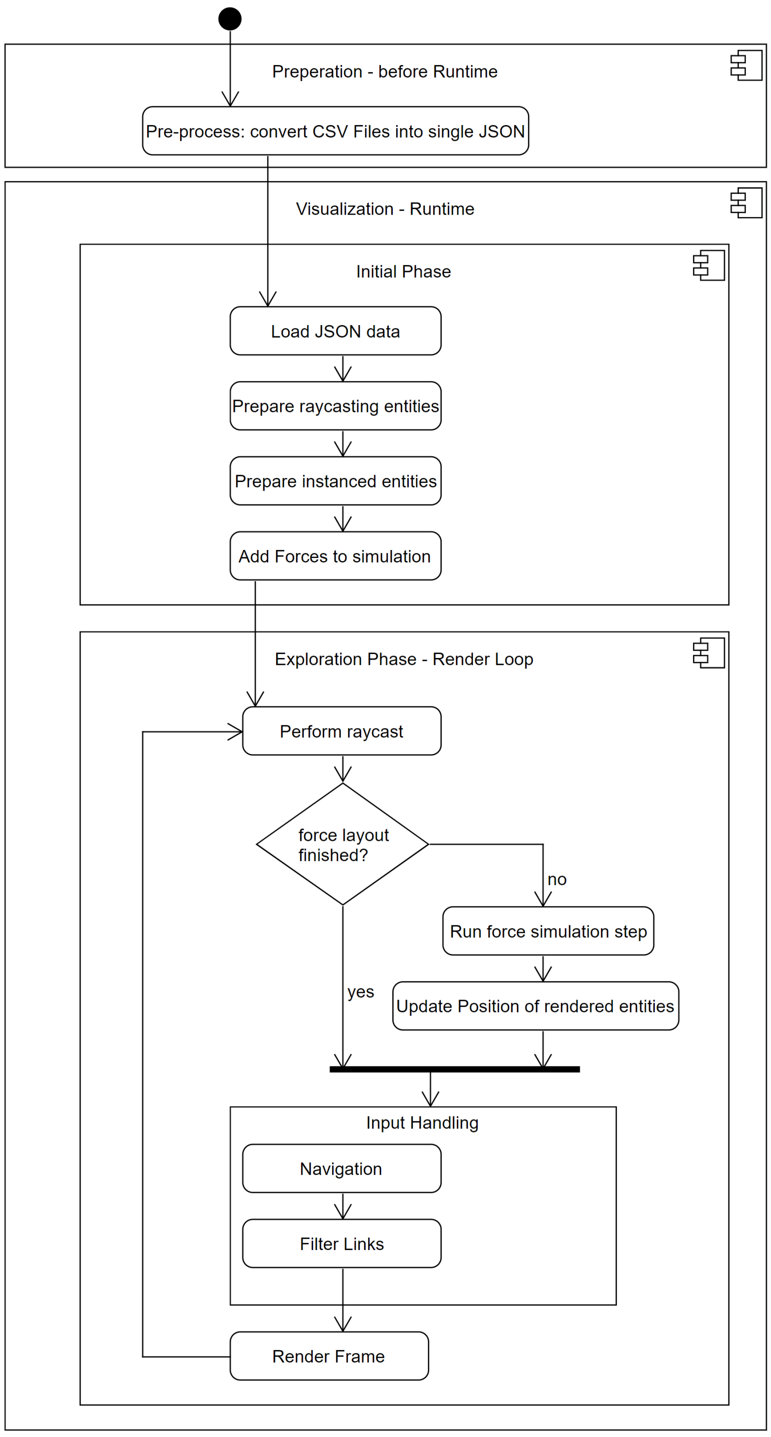

Preperation - before Runtime
Pre-process: convert CSV Files into single JSON
Visualization - Runtime
Initial Phase
Load JSON data
Prepare raycasting entities
Prepare instanced entities
Add Forces to simulation
Exploration Phase - Render Loop
Perform raycast
force layout finished?
no
Run force simulation step
yes
Update Position of rendered entities
Input Handling
Navigation
Filter Links
Render Frame

Figure 5.3: Sequence diagram of the program flow for our application.

We separated the application into multiple processing steps. Figure 5.3 describes the program flow and all abstract steps that are necessary for the visualization. It consists of an offline preparation phase where the graph data is converted once to our internal JSON data structure (see Section 5.5.1) and a runtime phase, which is executed every time the browser loads the webpage. The runtime phase starts with loading the data, afterwards preparations for the instanced rendering (see Section 5.5.3) and the force based layout (see Section 5.5.2) are carried out. Then, the main render loop is executed, which produces a rendered frame for each interaction. In addition, the layout calculation (see Section 5.5.2), navigation methods (see Section 5.5.4), scaling (see Section 4.5.1), filtering visible links (see Section 5.5.5) and performing the ray casting for the virtual laser pointer is handled by the main render loop. The objects that are intersected by the virtual laser pointer are used in the navigation and interaction techniques. Details on how intersected objects are determined by ray casting is not further described as this method is already provided from the original implementation.

5.4.3 Virtual Scene Graph

A-Frame applications are built by creating a virtual scene graph. The framework uses an entity-component-system architecture, which follows the “composition over inheritance and hierarchy” principle. This means that every object in A-Frame is an entity that can be extended and customized. There are various components that can be reused and extended. The base component is represented by the <a-entity> element.

Listing 5.2 shows a simplified version of the virtual scene graph the application uses. It consists of entities for both Vive controllers, a passive and active camera, the rig setup and the graph object itself. Most data and logic is encapsulated in the graphData object. We maintain two types of data lists: a list of nodes/links for the ray casting entities and a list of nodes/links for the actual rendered entities. The reason for this duplicated data is that we use instanced objects for rendering (see Section 5.5.3) and these can not be used for ray casting directly.

Listing 5.2: Simplified virtual A-Frame scene graph used by the application.

```
<body>
    <a-scene ... >
        <a-entity vive-controls="hand: left"> </a-entity>
        <a-entity vive-controls="hand: right"> </a-entity>

        <a-entity id="graphData" json-url="data/inputData.json" ... >
            <a-entity id="passive_rig"  ... >
                <a-entity id="passive_cam" ... > </a-entity>
            </a-entity>
            ...
        </a-entity>

        <a-entity id="active_rig" ... >
            <a-text id="controllerLabel" ... > </a-text>
            <a-entity id="active_cam" ... > </a-entity>
        </a-entity>
        ...
    </a-scene>
</body>
```

5.5 Application Details

5.5.1 Preprocessing Scripts

The visualization expects a combined JSON file with all nodes and links and their assigned layer as seen in Section 5.4.1. A common data export format for networks are multiple CSV files: one file for the nodes and another for their edges. In a hierarchical or clustered network, there can be an additional file with the hierarchical mapping. Therefore, we needed a script to convert these CSV files to our JSON data format. For convenience, most parameters can be configured by an additional config file without changing any program code. While transforming the data structure, all node IDs are extended with their hierarchical layer information to ensure unique IDs for the entire dataset. The node with an ID 1 on hierarchical layer 0 is assigned the new ID 0.1. In addition, we also normalize the node and link width and add color attributes for a simplified rendering process. To test our visualization with various networks of different sizes, there is also a script for generating test data.

5.5.2 Automated Force Layout

The node layout is automatically calculated during runtime by a force system powered by D3-Force [Bos21]. At the beginning, all D3-Force specific parameters including alphaDecay, alphaTarget, alphaMin, velocityDecay and cooldownTime are set. These parameters adjust the flexibility of the simulation by changing the strength behavior for all forces, how much the velocity is decreased between every simulation step and how many simulation steps are applied in total. In a second step, we add all forces to the simulation. This process is shown in Listing 5.3. The center, many-body, link and collision forces use a slightly adapted version of the original implementation in D3 force. The spherical constraint is our own implementation which can be seen in Listing 5.4. The main goal here is to add a velocity for each child node depending on the current node position. This velocity pulls the nodes to the center of their parent node which allows us to achieve the proclaimed hierarchical nested layout. Lastly, a simulation step is called every frame, which handles the subroutine calls for the different forces. The simulation is then stopped when a fixed number of steps is executed. The amount of simulation steps is determined by the alpha parameters set in the first step.

Listing 5.3: Simplified algorithm that shows which forces are added to the simulation.

```
//add forces for hierarchical layer 0
d3ForceLayout.force("layer0-center",centerForce(...));
d3ForceLayout.force("layer0-link",linkForce(...))
d3ForceLayout.force("layer0-manyBody",manyBodyForce(...));
d3ForceLayout.force("layer0-collide",collideForce(...)));

//add forces for each parent node (layer 1-n)
parentNodeIDs.forEach((parentNodeID, index) => {
    const filteredNodesForLayer = filterNodes(nodes, function (d) {
        return d[parentNodeIDKey] === parentNodeID && parseInt(d.layer)
            === childLayer;
    })

    d3ForceLayout.force(parentNodeID + "-sphericalConstraint",
        sphericalConstraint(filteredNodesForLayer,...));
    d3ForceLayout.force(parentNodeID + "-forceCollision",collisionForce(
        filteredNodesForLayer,...));
    d3ForceLayout.force(parentNodeID+ "-manyBody",manyBodyForce(
        filteredNodesForLayer,...));
    d3ForceLayout.force(parentNodeID+ "-link",linkForce(
        filteredNodesForLayer,...));
});
```

Listing 5.4: Simplified algorithm for the spherical constraint. We apply adapted velocities whenever the child node is inside the parent. We further determine if the childnode is already closer to the center of the node or not.

```
nodes.forEach(node => {
    let layerInt = parseInt(node.layer);

    let diffX = node.x - parentNode.x;//distance to the center
    let velocityDiffX;
    if(Math.abs(diffX) < radius){//child is inside parent
        if (Math.abs(diffX) < innerRadius) {//node is already close to
           the center
            velocityDiffX = diffX * jiggle(layerInt * const);
            //jiggle provides a randomized value
        }else{
            velocityDiffX = diffX * strength / const;
        }
    }else{//child is outside parent
        velocityDiffX = diffX * strength + jiggle(layerInt);
    }
    //Add velocity towards the center of the parent node
    node.vx -= (velocityDiffX/constant);
    //Add alpha value to allow the simulation to 'cool down'
    node.vx *= (1-alpha);
    ... (same for y,z)
}
```

5.5.3 Rendering

There are multiple goals for the rendering part, one of them is to increase the performance for large graphs. To achieve such FPS improvement, the application uses instanced rendering for nodes and links. This means that there is not one 3D object for every node/link in the virtual scene. Instead, we group multiple items and render them as a group. The final rendered position is then determined in the vertex shader, which reduces the overhead of draw calls from WebGL. The implementation of the prior paper [SWKA19] used two instanced buffers: one for the nodes and one for the links. Another goal for the rendering was to use transparent nodes. This requires to create multiple instanced buffers for the nodes, one for each hierarchical layer. Therefore, enabling us to set the render order of the instanced buffers in ascending order according to the hierarchical layers. A correct render order ensures a valid transparency so that child nodes can be seen from outside.

5.5.4 VR Navigation

As described in Section 4.5 for navigation we provide two methods: free flying and an animated teleport. Before we go into the details of the techniques, it is important to understand the core concept of navigation for 6-DOF VR headsets. The camera is placed in a virtual camera rig, which represents the user's available movement space. Programming guidelines from VR APIs prohibit programmatically changing the position of the headset or controllers in the rig as this would create confusion for the user. The only way the headset and controllers are moved inside the rig is by physical movement in the real world, thus enabling enable free walking inside the virtual rig. Instead of the camera directly, we can programmatically change the position of the rig inside the virtual scene. We apply this technique for our free flying and teleport technique. The free flying technique is already provided by A-Frame and the implementation from the prior paper. We just added additional controller button callbacks for adapting the fly speed and rotate of the rig in the virtual scene on demand. For the animated teleport technique, we no longer fly to the center of the selected node, as in the original implementation. Instead, the target position is the node border barely on the inside of the node. Therefore, we changed the calculation for the target position, which can be seen in Listing 5.5. The process is also visualized in Figure 5.4.

Listing 5.5: Matrix calculations for determining the target position of the animated teleport.

```
//1. get all needed information
const nodePos = node.pos;
const nodeRadius = multilayerNodeDiameter(node.__data);
const sceneScale = detailLayout.getCurrentSceneScale() ;
let cameraPos = getCurrentCameraRigWorldPos().add(
    getRelativeCameraToRigPos());
//2. calculate the node border position between the camera and nodeCenter
const vecNodeToCamera = cameraPos.add(nodePos.clone().negate());
const vecCenterToNodeBorder = vecNodeToCamera.normalize().multiplyScalar(
    nodeRadius*0.95*sceneScale);//*0.95 as we want to be slightly inside
    the selected node
//3. convert position so that the headset instead of the rig is centered
const positionNodeBorder = nodePos.clone().add(vecCenterToNodeBorder);
const positionNodeBorderCorrectedWithCurrentCameraPos =
    positionNodeBorder.clone().add(getRelativeCameraToRigPos().negate());
//4. initiate animated teleport
flyToPosition(positionNodeBorderCorrectedWithCurrentCameraPos,
    flyingElement);
```

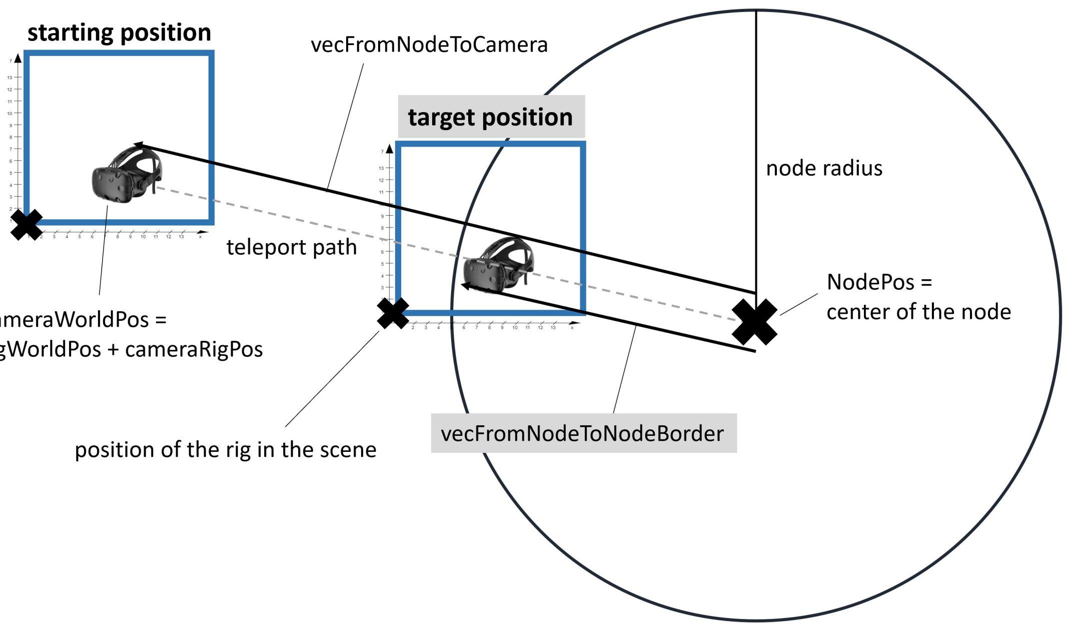


Figure 5.4: 2D representation of the calculation for the target position for the animated teleport. A virtual teleport path is formed between the center of the node and the position of the headset. This virtual path is extended until it reaches the border of the node.

In a first step, we get all information needed including the camera position in world space, the position of the node center, the node radius and the current adaptable scale of the scene. Then we calculate the position of the crossing between the node border and the direct path from the camera to the node center. We do this by calculating a vector from the node center to the camera position. After normalizing the length and applying this vector with the correct length, determined by the node radius and scene scale, to the center of the node, we get the position where the headset should end up when teleporting to the selected node. We can only manipulate the headset position indirectly through changing the rig position. Therefore, we have to calculate the relative rig position in the third step. The resulting position contains the final coordinates for centering the rig so that the VR headset ends up barely inside the node. For calculating the target position when flying to the parent node, we apply a similar technique shown in Figure 5.5.

5.5.5 Filtering Visible Links

In our visualization, links can be filtered by two ways: either by directly selecting a node, which is accomplishment with the virtual laser pointer and ray casting, or by an automated process, which selects the inner-most node related to the user’s position if no node is directly selected. In order to determine the inner-most node, we first

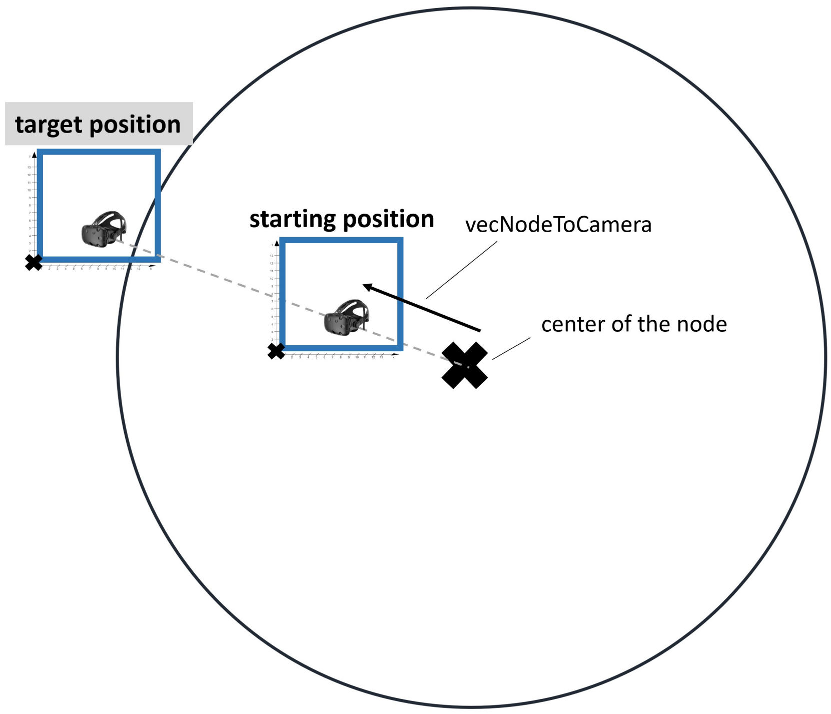


Figure 5.5: 2D representation for calculating the teleporting positing when teleporting to the parent node.

have to check for every node if the user is inside it or not (see Listing 5.6). We store all entered nodes in a temporary node cache in order to calculate the inner-most node (see Listing 5.7). Then we use that determined node to update the visibility of all links with the rules described in Section 4.4.1.

Listing 5.6: Simplified algorithm to determine whenether the user is inside a node or not.

```
//called for every frame
for (var i = 0; i < graphData.nodes.length; i++) {
    ...
    var dist = nodePos.distanceTo(getCurrentCameraRigWorldPos().add(
        getRelativeCameraToRigPos()));
    var threshold = multilayerNodeDiameter(node) * sceneScale;
    if (dist <= threshold) {
        detailLayout.onEnterNode(node);
    }else{
        detailLayout.onLeaveNode(node);
    }
}
```

Listing 5.7: Simplified algorithm to determine the inner most node.

```
getCurrentInnestNode(){
    let innestNode = {id:"-",layer:-1};
    for (let enteredNodesKey in this.enteredNodes) {
        if(innestNode.layer < parseInt(this.enteredNodes[enteredNodesKey
            ])){
            innestNode.id = enteredNodesKey;
        }
    }
    return this.nodeCache[innestNode.id];
}
```

CHAPTER 6

Results

Our proclaimed goal for the visualization was to enable large hierarchical networks in VR. Therefore, we decided to test our application in a performance evaluation in Section 6.1 and gather user feedback in the form of an informal feedback in Section 6.2.

6.1 Performance

Real time 3D applications need to have a constant frame rate in order to achieve a smooth experience. For VR-based applications this is even more important as a low frame rate could lead to cyber sickness. In our performance test, we distinguish between the duration of the initial phase, which lasts from the application start until the end of the force based layout algorithm, and the render performance during the exploration phase.

In the performance evaluation, we want to investigate the scalability of our application. For an optimal VR experience, the frame rate should be constant and equal to the maximum supported refresh rate of the VR headset's displays. In the case of the HTC Vive, this requires us to have a constant frame rate of 90 FPS. Our own experiments showed that 20-30 FPS is the minimum for a smooth experience while exploring the virtual scene.

We measured the FPS directly from the visualization by adding an FPS counter on the virtual controller model during the exploration. Our setup was a desktop PC with a Ryzen 7 3700X CPU and a Radeon RX 590 GPU. We tested multiple datasets with different sizes. The results are shown in Table 6.1. In order to detect possible performance problems, we tested the scaling of the nodes and links individually. We found that the number of nodes scale worse than the number of links (see Figure

6.1 and 6.2). For an equally distributed graph, the number of hierarchical layers only indirectly influence the frame rate, as with an increased number of layers the number of nodes grows exponentially. We always apply a constant number of simulation steps. Therefore, at about 6 hierarchical layers, the layout algorithm becomes unstable and can not guarantee a correct hierarchical nesting anymore. While the average frame rate is not drastically reduced by the amount of links, the frame rate seems to get more unstable with an increasing amount of links. This was especially noticeable during the teleportation animation were the frame rate dropped on a large dataset down to 20 FPS. We assume the automated scaling reduced the frame rate quite significantly. During the animation, the scale of nearly all objects in the scene is updated every frame. Our data only shows a maximum of 90 FPS due to the headset's maximum refresh rate of 90 Hz. In conclusion, with the current state of the implementation, the visualization can support datasets up to 1500 nodes and 3000 links depending on the hardware available. In addition, we noticed that during the exploration, CPU and GPU utilization was not at maximum. An explanation for this could be that the application is mostly CPU bound and further limited by a single threaded JavaScript implementation.

nodes	**links**	**layers**	**layout duration (s)**	**min FPS**	**avg FPS**
310	0	3	5	90	90
730	0	3	7,5	70	75
1100	0	3	9,5	55	60
1560	0	4	10,5	40	55
2343	0	5	15	30	35
774	387	3	10	60	70
774	1548	3	15	55	70
774	3870	3	27	40	60
1560	3120	4	30	25	45
1020	2040	4	20	40	60

Table 6.1: Results from the performance evaluation, separated into duration of the layout phase in seconds and render performance in FPS during the exploration phase. Test setup: Ryzen 7 3700X + Radeon RX 590.

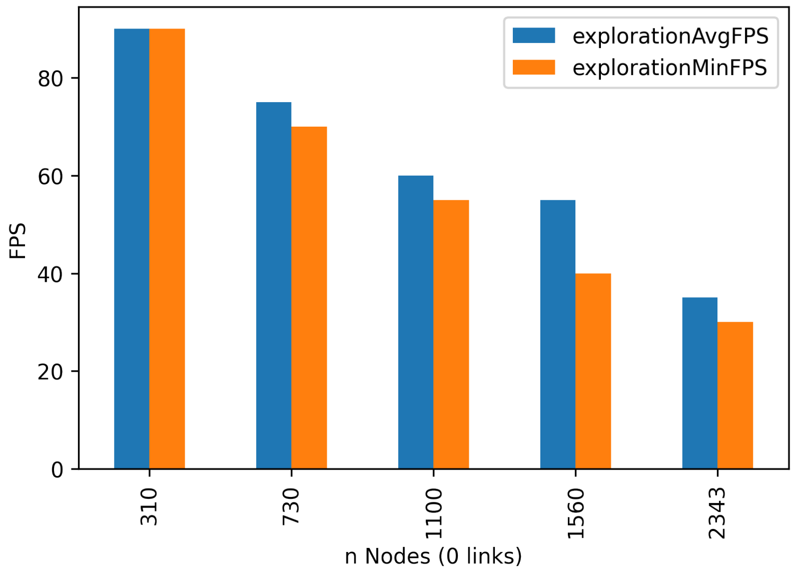


Figure 6.1: Performance chart of the scalability for the number of nodes. To better compare the results, only datasets with 0 links are shown in this graph. Increasing the number of nodes quickly leads to a performance issue.

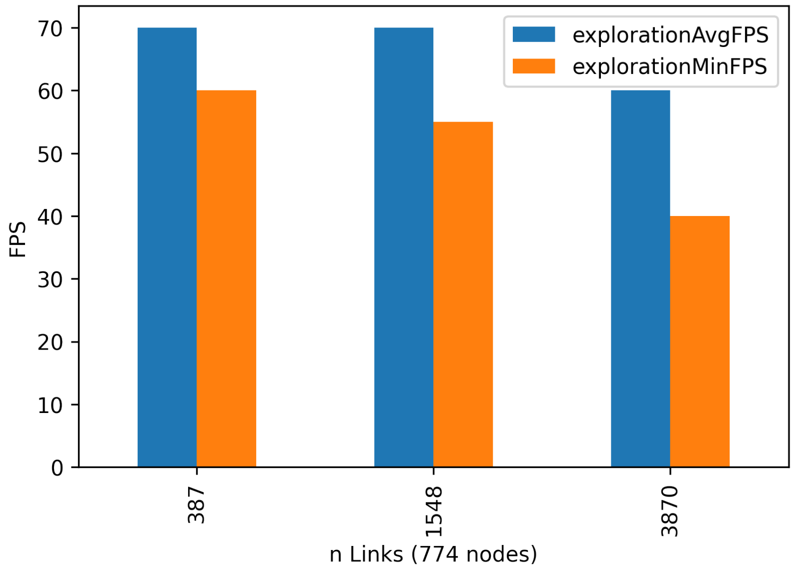


Figure 6.2: Performance chart of the scalability for the number of links. To better compare the results, only datasets with the same amount of 774 nodes are shown in this graph. In comparison to the nodes, links can be easier scaled up without impacting the performance too much.

6.1.1 Possible Optimization

To increase the overall performance, the scalability for the number of nodes needs to be improved. We can separate the performance problems between GPU and CPU bottlenecks. To increase the GPU performance, we already limit the draw calls by rendering the nodes with instanced buffers. However, the number of rendered triangles is still very high. One improvement could be to use simpler sphere models with a lower polygon count. Another to use impostor rendering. Implementing a technique that prevents the rendering of the small nodes which can not be seen anyway could further increase GPU performance. To increase CPU performance, we have to reduce the number of tasks during each iteration of the main render loop (see Figure 5.3). A big performance issues might be the calculation of the intersected objects from ray casting by the virtual laser pointer. This could be improved by taking advantage of the hierarchical layout. Selection of child nodes from different parents is not possible. Therefore, these nodes could be excluded from the ray intersection checks. Another improvement could be achieved by limiting the active time of the ray casting by binding the virtual laser pointer to a button that the user actively needs to hold while selecting objects. That would enable an improved frame rate while the user is only looking around without using the virtual laser pointer. Besides performance improvements to the techniques themselves, improving the internal data structure would also increase the performance. Based on the prior implementation, we use a flat list similar to the JSON seen in Section 5.1. With a tree data structure that directly represents the hierarchical relationships, our internal algorithms can be implemented more efficiently when accessing the node details, therefore reducing the workload on the main render loop and increasing the FPS.

6.2 Informal Feedback

The focus of this thesis was to develop a new visualization technique. This included a hierarchical layout and VR optimized navigation and interaction techniques. The application should deliver a smooth and user-friendly visualization. As the application runs in the browser, we set up a website [Eiw21] with an introduction to the topic, instructions on how to start and interact with the visualization, and the option to try out the application with multiple datasets of various sizes. Afterwards, we invited users to participate in the evaluation and fill out an online form that we prepared beforehand. One person took part in the online evaluation. Besides the online evaluation, we also had another participant who tried the visualization on site. During and after the exploration, we conducted an interview. The results of the interview as well as the online forms are presented in the following subsections.

6.2.1 Evaluation Results

Both participants described their first impression as slightly overwhelming. Especially in the beginning it was hard for them to get a good overview of the graph. The overview perspective was only used for a short duration. However, after a few minutes of exploring the visualization in the detail perspective, the structure of the graph became more clear. The hierarchical nesting helped them to understand the clustering of the nodes. At the initial scale of the scene, the room scale experience did not help much as the nodes where too large to wander around the graph. After manually scaling down the scene, one participant described that the overview got a lot better as he now was able to navigate in the graph by moving around in the real world. The improved spatial impression made it easier to detect the hierarchical nesting of the nodes. Both participants described the visualization as aesthetically pleasant and were able to detect the nodes and links. As the links get smaller with increasing hierarchical layer depth, participants had problems to detect and hover over them with the laser pointer. In addition, sometimes there was a noticeable flickering, due to z-fighting, of nodes and node labels.

For navigation, we had mixed results. One participant mostly used the free flying technique, the other preferred real world movement with the combined animated teleport technique. The participant of the interview commented that the free flying speed was too slow. In addition, he missed explicit buttons to move up and down without having to change his gaze direction. The animated teleport was intuitive and useful for both participants, while one noticed a bit of cyber sickness during the transition and suggested speeding up the animation. Another suggestion was to use the left trigger for the animated teleport to the parent node. In general, there were no major problems for navigating around the graph.

Lastly, filtering of the visible nodes was rather confusing for the participants. It was not clear when and how the automated filtering works. On entering another node, there was a slight delay before the visibility of the nodes updated. One participant suggested adding a visual indicator for the current state of the automatic filtering. The manual filtering was easier to understand. Selecting a node for filtering with the laser pointer was considered intuitive and there was also an immediate update visible.

6.2.2 Usability Conclusion

With the feedback we received from participants of the usability evaluation, we analyze the current shortcomings of our approach. As already mentioned before, on large graphs with more than six layers, the visualization can not ensure a correct hierarchical nesting, which leads to resulting problems for the navigation during the exploration. By adapting the simulation parameters according to the number

of hierarchical layers or including a check before stopping the layout calculation, this could be improved and enable the visualization to represent more hierarchical layers. During the evaluation, we found that the usage of the application has a learning curve. While some techniques were confusing in the beginning, after about 15 minutes exploring and testing the experience, the perceived intuitiveness of navigation, filtering and overview got a lot better. This could be improved by optimizing multiple parameters, like the free flying speed, node and link size and automatic scaling size. In addition, a user guide would enable an easier start for exploring large hierarchical networks using our visualization. In a future version of the visualization, a customizable button mapping would allow the users to configure the application to their preferences. An additional visual indicator if the automated filtering is active and for which node, would give the user more control over the filtering process.
The combination of the instanced and transparent node rendering creates a new challenge, as the correct render order can not be determined by the default implementation from A-Frame. This leads to a sometimes wrong representation of parent node affiliation. To improve the clarity and prevent a flickering effect, the position and scale of the node labels should be optimized.

CHAPTER 7

Conclusion and Future Work

The aim of this thesis was to enable the visualization of hierarchical networks in VR and fully utilize the benefits of 3D information visualization and VR. In addition, the improved spatial impression, intuitive interaction and navigation methods VR applications can provide are adapted for our use case. This visualization should enable data scientists in the future to explore even larger hierarchical datasets and provide more insights than currently possible with conventional 2D representations. To accomplish this, we adapted the original VR node-link visualization by Sorger et al. [SWKA19]. Our customized implementation includes an optimized force-based layout with spherical constraints that enables a hierarchical nested node layout of n hierarchical layers. The visualization introduces transparent node rendering and optimized navigation methods, as well as a technique to filter visible links. To solve the multi-scale scene problem, we automatically adjust the movement speed as well as the scene scale. In addition, to provide a fully flexible user experience, the movement speed and scale can be adjusted by the user at any time and all navigation methods can be combined flexibly. To evaluate our results, we conducted a performance test and gathered user feedback using an on site interview and an online form.

To our knowledge, this thesis provides a first approach on visualizing hierarchical networks in VR. Therefore, future work could extend our concepts and improve the stability, performance and feature richness. With an added support for links between different hierarchical layers, applications could enable the visualization of multilayer datasets without a hierarchical relationship. Edge bundling could further improve the clarity of the visualization and allow displaying larger amounts of links at once. Different navigation and interaction techniques, like the ones we have seen in Section 3.2.2 and 3.2.3, could be implemented to improve the user experience while exploring the graph. As for the future of our application in particular, upgrading to the newest

A-Frame version would enable the use of the WebXR standard and therefore support a variety of different VR headsets and browsers. By introducing a dependency management software, like npm the application's expendability, would greatly be improved, therefore easily allow the implementation of new features in the future. In the first feedback we found out, that after about 15 minutes exploring the data in VR, users start to get a clear overview of the data. Therefore, we believe that our proposed visualization provides a promising approach for data science applications. In addition, we believe that, in the future, the visualization community will see many other publications related to visualization in VR and that our visualization can provide important knowledge for other researchers.

List of Figures

List of Tables

Bibliography

[AMA08] Daniel Archambault, Tamara Munzner, and David Auber. Grouse-Flocks: Steerable Exploration of Graph Hierarchy Space. *IEEE Transactions on Visualization and Computer Graphics*, 14:900–13, July 2008.

[AOCVMQ17] Armando Arce-Orozco, Luis Camacho-Valerio, and Steven Madrigal-Quesada. Radial Tree in Bunches: Optimizing the use of space in the visualization of radial trees. In Sergio Lujan Mora, editor, *INCISCOS'17: Proceedings of the International Conference on Information Systems and Computer Science*, pages 369–374. IEEE Computer Society, 2017.

[BD04] M. Balzer and O. Deussen. Hierarchy Based 3D Visualization of Large Software Structures. In *IEEE Visualization 2004*, pages 4p–4p, October 2004.

[BD07] M. Balzer and O. Deussen. Level-of-detail visualization of clustered graph layouts. In *2007 6th International Asia-Pacific Symposium on Visualization*, pages 133–140, February 2007.

[BM99] François Bertault and Mirka Miller. An Algorithm for Drawing Compound Graphs. In Jan Kratochvíyl, editor, *Graph Drawing*, Lecture Notes in Computer Science, pages 197–204, Berlin, Heidelberg, 1999. Springer.

[BM07] D. A. Bowman and R. P. McMahan. Virtual Reality: How Much Immersion Is Enough? *Computer*, 40(7):36–43, July 2007.

[Bos21] Mike Bostock. d3-force - https://github.com/d3/d3-force, January 2021.

[Bra14] Richard Brath. 3D InfoVis is here to stay: Deal with it. In *2014 IEEE VIS International Workshop on 3DVis (3DVis)*, pages 25–31, November 2014.

[CNSD$^+$92] Carolina Cruz-Neira, Daniel J. Sandin, Thomas A. DeFanti, Robert V. Kenyon, and John C. Hart. The CAVE: audio visual experience automatic virtual environment. *Communications of the ACM*, 35(6):64–72, June 1992.

[Cor21] Valve Corporation. Openvr source code github repo. https://github.com/ValveSoftware/openvr, February 2021.

[DCW$^+$17] Adam Drogemuller, Andrew Cunningham, James A. Walsh, William Ross, and Bruce H. Thomas. VRige: Exploring Social Network Interactions in Immersive Virtual Environments. 2017.

[DCW$^+$20] Adam Drogemuller, Andrew Cunningham, James Walsh, Bruce H. Thomas, Maxime Cordeil, and William Ross. Examining virtual reality navigation techniques for 3D network visualisations. *Journal of Computer Languages*, 56:100937, February 2020.

[DDPA15] Manlio De Domenico, Mason A. Porter, and Alex Arenas. MuxViz: a tool for multilayer analysis and visualization of networks. *Journal of Complex Networks*, 3(2):159–176, June 2015.

[Die17] Reinhard Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer-Verlag, Berlin Heidelberg, 5 edition, 2017.

[Duc17] César Ducruet. Multilayer dynamics of complex spatial networks: The case of global maritime flows (1977–2008). *Journal of Transport Geography*, 60:47–58, April 2017.

[EF97] Peter Eades and Qing-Wen Feng. Multilevel visualization of clustered graphs. In Stephen North, editor, *Graph Drawing*, Lecture Notes in Computer Science, pages 101–112, Berlin, Heidelberg, 1997. Springer.

[Eiw21] Manuel Eiweck. 3dmultilayer.emanum.dev, March 2021.

[FR91] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, 1991.

[GMM$^+$19] Mohammad Ghoniem, Fintan Mcgee, Guy Melançon, Benoit Otjacques, and Bruno Pinaud. The State of the Art in Multilayer Network Visualization. *arXiv:1902.06815 [cs]*, February 2019. arXiv: 1902.06815.

[GPK12] Nicolas Greffard, Fabien Picarougne, and Pascale Kuntz. Visual Community Detection: An Evaluation of 2D, 3D Perspective and 3D Stereoscopic Displays. In Marc van Kreveld and Bettina Speckmann, editors, *Graph Drawing*, Lecture Notes in Computer Science, pages 215–225, Berlin, Heidelberg, 2012. Springer.

[Gro21a] Immersive Web Working Group. Webxr specification. https://immersiveweb.dev/, February 2021.

[Gro21b] Khronos Group. Openxr specification - official website. https://www.khronos.org/openxr/, February 2021.

[Gro21c] WebVR Working Group. Webvr specification. https://webvr.info/, February 2021.

[GSWD18] J. Görtler, C. Schulz, D. Weiskopf, and O. Deussen. Bubble Treemaps for Uncertainty Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):719–728, January 2018.

[HZBK08] Harry Halpin, David J. Zielinski, Rachael Brady, and Glenda Kelly. Exploring Semantic Social Networks Using Virtual Reality. In Amit Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy Finin, and Krishnaprasad Thirunarayan, editors, *The Semantic Web - ISWC 2008*, pages 599–614, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[Inc21] Supermedium (Super XYZ Inc.). A-frame - https://aframe.io/, February 2021.

[IYIK04] T. Itoh, Y. Yamaguchi, Y. Ikehata, and Y. Kajinaga. Hierarchical data visualization using a fast rectangle-packing algorithm. *IEEE Transactions on Visualization and Computer Graphics*, 10(3):302–313, May 2004.

[JLG$^+$17] David Jonker, Scott Langevin, David Giesbrecht, Michael Crouch, and Nathan Kronenfeld. Graph mapping: Multi-scale community visualization of massive graph data. *Information Visualization*, 16(3):190–204, July 2017. Publisher: SAGE Publications.

[KKM$^+$20] J. Kotlarek, O. Kwon, K. Ma, P. Eades, A. Kerren, K. Klein, and F. Schreiber. A Study of Mental Maps in Immersive Network Visualization. In *2020 IEEE Pacific Visualization Symposium (PacificVis)*, pages 1–10, June 2020.

[KMLM16] Oh-Hyun Kwon, Chris Muelder, Kyungwon Lee, and Kwan-Liu Ma. A Study of Layout, Rendering, and Interaction Methods for Immersive Graph Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 22(7):1802–1815, July 2016.

[KMV15] Stephen G. Kobourov, Tamara Mchedlidze, and Laura Vonessen. Gestalt Principles in Graph Drawing. In Emilio Di Giacomo and Anna Lubiw, editors, *Graph Drawing and Network Visualization*, Lecture Notes in Computer Science, pages 558–560, Cham, 2015. Springer International Publishing.

[Kob12] Stephen G. Kobourov. Spring Embedders and Force Directed Graph Drawing Algorithms. *arXiv:1201.3011 [cs]*, January 2012.

[KPW14] Andreas Kerren, Helen C. Purchase, and Matthew O. Ward. Introduction to Multivariate Network Visualization. In Andreas Kerren, Helen C. Purchase, and Matthew O. Ward, editors, *Multivariate Network Visualization: Dagstuhl Seminar #13201, Dagstuhl Castle, Germany, May 12-17, 2013, Revised Discussions*, Lecture Notes in Computer Science, pages 1–9. Springer International Publishing, Cham, 2014.

[KWO$^+$20] M. Kraus, N. Weiler, D. Oelke, J. Kehrer, D. A. Keim, and J. Fuchs. The Impact of Immersion on Cluster Identification Tasks. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):525–535, January 2020.

[LAK$^+$20] Jong-In Lee, Paul Asente, Byungmoon Kim, Yeojin Kim, and Wolfgang Stuerzlinger. Evaluating Automatic Parameter Control Methods for Locomotion in Multiscale Virtual Environments. In *26th ACM Symposium on Virtual Reality Software and Technology*, VRST '20, pages 1–10, New York, NY, USA, November 2020. Association for Computing Machinery.

[MCH$^+$18] Kim Marriott, Jian Chen, Marcel Hlawatsch, Takayuki Itoh, Miguel A. Nacenta, Guido Reina, and Wolfgang Stuerzlinger. Immersive Analytics: Time to Reconsider the Value of 3D for Information Visualisation. In Kim Marriott, Falk Schreiber, Tim Dwyer, Karsten Klein, Nathalie Henry Riche, Takayuki Itoh, Wolfgang Stuerzlinger, and Bruce H. Thomas, editors, *Immersive Analytics*, Lecture Notes in Computer Science, pages 25–55. Springer International Publishing, Cham, 2018.

[MH08] Laurens van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[Mun97] T. Munzner. H3: laying out large directed graphs in 3D hyperbolic space. In *Proceedings of VIZ '97: Visualization Conference, Information Visualization Symposium and Parallel Rendering Symposium*, pages 2–10, October 1997.

[NVRS18] Thinh Nguyen-Vo, Bernhard E. Riecke, and Wolfgang Stuerzlinger. Simulated Reference Frame: A Cost-Effective Solution to Improve Spatial Orientation in VR. In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 415–422, Reutlingen, March 2018. IEEE.

[Rib20] Severino Ribecca. Circle Packing - The Data Visualisation Catalogue, December 2020.

[RMC91] George G. Robertson, Jock D. Mackinlay, and Stuart K. Card. Cone Trees: Animated 3D visualizations of hierarchical information. In Scott P. Robertson, Gary M. Olson, and Judith S. Olson, editors, *CHI'91: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 189–194. ACM Press, 1991.

[Sch11] H. Schulz. Treevis.net: A Tree Visualization Reference. *IEEE Computer Graphics and Applications*, 31(6):11–15, November 2011.

[Shn92] Ben Shneiderman. Tree visualization with tree-maps: 2-d space-filling approach. *ACM Transactions on Graphics*, 11(1):92–99, January 1992.

[Shn96] B. Shneiderman. The eyes have it: a task by data type taxonomy for information visualizations. In *Proceedings 1996 IEEE Symposium on Visual Languages*, pages 336–343, September 1996. ISSN: 1049-2615.

[SQX$^+$06] Hongzhi Song, Yu Qi, Lei Xiao, Tonglin Zhu, and Edwin P. Curran. LensTree: Browsing and Navigating Large Hierarchical Information Structures. In Qingzhang Chen, Ronghua Liang, and Zhigeng Pan, editors, *ICAT'06: Proceedings of the International Conference on Artificial Reality and Telexistence*, pages 682–687. IEEE Computer Society, 2006.

[Sut65] Ivan E. Sutherland. The ultimate display. In *Proceedings of the IFIP Congress*, pages 506–508, 1965.

[SWKA19] Johannes Sorger, Manuela Waldner, Wolfgang Knecht, and Alessio Arleo. Immersive analytics of large dynamic networks via overview and detail navigation. pages 144–151, December 2019.

[tosc21] three.js open source contributors. three.js - https://threejs.org/, February 2021.

[UAW$^+$99] Martin Usoh, Kevin Arthur, Mary C. Whitton, Rui Bastos, Anthony Steed, Mel Slater, and Frederick P. Brooks. Walking > walking-in-place > flying, in virtual environments. In *Proceedings of the 26th* *annual conference on Computer graphics and interactive techniques - SIGGRAPH '99*, pages 359–364. ACM Press, 1999.

[Veh15] Corinna Vehlow. The State of the Art in Visualizing Group Structures in Graphs. January 2015.

[vLKS$^+$11] T. von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J.J. van Wijk, J.-D. Fekete, and D.W. Fellner. Visual Analysis of Large Graphs: State-of-the-Art and Future Research Challenges. *Computer Graphics Forum*, 30(6):1719–1749, September 2011.

[WF96] Colin Ware and Glenn Franck. Evaluating stereo and motion cues for visualizing information nets in three dimensions. *ACM Transactions on Graphics*, 15(2):121–140, April 1996.

[WWDW06] 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, pages 517–520, New York, NY, USA, April 2006. Association for Computing Machinery.

[YCB$^+$20] Yalong Yang, Maxime Cordeil, Johanna Beyer, Tim Dwyer, Kim Marriott, and Hanspeter Pfister. *Embodied Navigation in Immersive Abstract Data Visualization: Is Overview+Detail or Zooming Better for 3D Scatterplots?* August 2020.

[YFY$^+$17] Yi-Jheng Huang, T. Fujiwara, Yun-Xuan Lin, Wen-Chieh Lin, and Kwan-Liu Ma. A gesture system for graph visualization in virtual reality environments. In *2017 IEEE Pacific Visualization Symposium (PacificVis)*, pages 41–45, April 2017. ISSN: 2165-8773.

[ZMBS14] XueZhong Zhou, Jörg Menche, Albert-László Barabási, and Amitabh Sharma. Human symptoms–disease network. *Nature Communications*, 5(1):4212, June 2014. Publisher: Nature Publishing Group.

[ZWB$^+$17] D. Zielasko, B. Weyers, M. Bellgardt, S. Pick, A. Meibner, T. Vierjahn, and T. W. Kuhlen. Remain seated: towards fully-immersive desktop VR. In *2017 IEEE 3rd Workshop on Everyday Virtual Reality (WEVR)*, pages 1–6, March 2017.