TU WIEN Informatics

3D Graph Algorithm for Multilayer Network Analytics

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Daniel Lippeck
Matrikelnummer 11776883

an der Fakultät für Informatik

der Technischen Universität Wien

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

Wien, 7. Februar 2022

Daniel Lippeck	Manuela Waldner

TU WIEN Informatics

3D Graph Algorithm for Multilayer Network Analytics

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Daniel Lippeck
Registration Number 11776883

to the Faculty of Informatics

at the TU Wien

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

Vienna, 7th February, 2022

Daniel Lippeck	Manuela Waldner

Erklärung zur Verfassung der Arbeit

Daniel Lippeck

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, 7. Februar 2022

Daniel Lippeck

Danksagung

An dieser Stelle möchte ich einen besonderen Dank an meine Betreuer Johannes Sorger, sowie Manuela Waldner widmen, welche mir während der Ideensammlung, dem Entwicklungsprozess und dem Schreiben der Arbeit tatkräftige Unterstützungen gaben. Weiters möchte ich mich bei meinem Freund Roman Braunstingl für das erhaltene Feedback zu der implementierten Visualisierung bedanken. Außerdem möchte ich mich bei meiner Freundin Silke Aschauer für die Unterstützung während dem Schreiben der Arbeit und dem emotionalen Support bedanken. Zu guter Letzt möchte ich mich noch bei meinen Eltern für die allgemeine Unterstützung im Studium und über die letzten Jahre bedanken.

Kurzfassung

Einfache Graphen sind in vielen Bereichen der Komplexitätsforschung nicht detailliert genug um komplexe Systeme akkurat abzubilden. Deswegen wurden über die letzten Jahre komplexere Datenstrukturen entwickelt, die zusätzliche Informationen zu einem einfachen Graphen abspeichern.

Eine immer häufigere benötigte Datenstruktur ist das sogenannte "Multilayer Netzwerk", welches in verschiedensten Bereichen verwendet wird: in der Medizin um Korrelationen zwischen Krankheiten zu finden oder in den Sozialwissenschaften um komplexe Beziehungsstrukturen zwischen verschiedenen Akteuren abzubilden.

Durch die Komplexität der Daten ist die Visualisierung besonders schwierig. Zweidimensionale Visualisierungen sind häufig nicht ausreichend, um die große Datenmenge abzubilden, ohne dass "Visual Clutter" die Analyse der Daten stört.
Diese Arbeit stellt daher eine neue Visualisierungstechnik für Multilayer Netzwerke vor. Die Technik orientiert sich an modernsten zweidimensionalen Lösungen und erweitert diese in die dritte Dimension, wodurch die oben beschriebenen Probleme von zweidimensionalen Visualisierungen minimiert werden sollen.

Für unseren Lösungsansatz wurden die verschiedenen Ebenen als Teilgraphen visualisiert, welche anschließend um eine Kugel platziert wurden. Um das Layout innerhalb eines Layers für Multilayer Netzwerke zu optimieren, wurden für die Platzierung der Knoten Verbindungen innerhalb und außerhalb des Layers berücksichtigt.
Um "Visual Clutter" zu minimieren wurde Kantenbündelung eingesetzt, sowie eine eigene Ansicht entwickelt, welche die Visualisierung des Systems auf Knoten und Kanten eines einzelnen Layers, sowie Verbindungen zu diesem Layer beschränkt. Unsere Resultate zeigen, dass unsere Lösung durch den zusätzlichen Visualisierungsraum durch die dritte Dimension, gemeinsam mit einem optimierten Layout, die Visualisierung von größeren Netzwerken und eine bessere Analyse erlauben, im Vergleich zu anderen zweidimensionalen Lösungen.

Abstract

Simple graphs are often not able to accurately represent real world entities. Therefore, more complex data structures have been introduced, one of which is the so-called "multilayer network". Multilayer networks are used in multiple fields, such as medical science, where data can represent the correlation of diseases or social science, where data might represent different actors and their relationship to one another.

Visualizing these complex data types is particularly challenging, as two-dimensional visualizations - the gold standard for graph visualizations - are often not good enough to gather deeper insight into the data, as big datasets quickly fill two-dimensional visualizations with visual clutter. Therefore, this thesis introduces a new visualization technique for multilayer networks by extending existing 2d state-of-the-art methods with a third dimensions.

Our solution visualizes the layers as sub graphs on a two-dimensional plane, which are positioned around a sphere. To optimize the layout within the layer for multilayer networks, our solution calculates the position of individual nodes by considering connections within the same layer, as well as connections to other layers. To minimize visual clutter, edge bundling was implemented, additionally to a view, which restricts the visualization to nodes and edges of a single layer, as well as connections to nodes of other layers. Our results show that our solution with the additional space due to the third dimension, combined with an optimized layout, allows users to visualize larger networks and gather better insight into the data, compared to conventional two-dimensional solutions.

Contents

CHAPTER 1

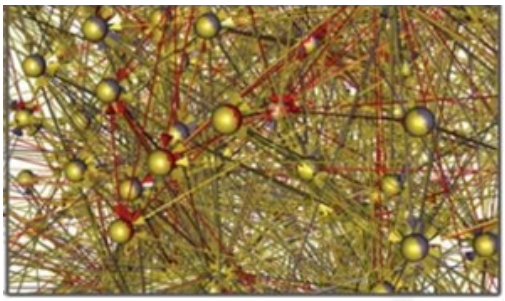
Introduction

1.1 Motivation

The analysis of graphs is an important task in complexity science. However, increasingly large datasets and more complex data structures require better visualizations for efficient analysis. A recurring problem is the analysis of multilayer networks (see Chapter 3.2). The need to visualize a lot of entities and their connections makes it difficult to solely rely on 2D visualization, since two-dimensional solutions typically run into a lot of visual clutter (see Fig. 1.1a), making single entities and connections between them hard to distinguish. Using three dimensions comes with many advantages compared to two dimensions [DRST14]. A new axis allows for more space to distribute data points, which decreases visual clutter. However, three-dimensional visualizations also come with some disadvantages, for example, they inherently struggle with occlusion of data (see Fig. 1.1b), since data is hidden behind other elements depending on the view perspective.

We therefore suggest a compromise solution where nodes inside a layer are placed inside a 2D plane and the layers themselves are positioned in 3d space, in our case around a sphere. We hypothesize that, for some datasets, this method can give the viewer easier insight into a single layer, while simultaneously allowing the viewer to make assumptions about the connections between different layers.

(a) A 2D example visualization, which shows that visualizing highly connected graphs often produces visual clutter.



(b) A 3d example visualization, which shows that depending on the viewing angle certain data points and their connections are hidden by other nodes and edges.

Figure 1.1: Example visualizations, which show visual clutter as a weakness for 2D solutions and occlusion of data as a weakness for 3d visualizations.

1.2 Aim of the Work

The goal of this thesis is to create a new multilayer network visualization technique, which gives experienced users the opportunity to gain insight into multilayer datasets. The user should be able to clearly determine which layers share connections and how strongly they are connected, without losing details like how the nodes are connected within their respective layer.
Visual clutter from unnecessary edge-crossings should be reduced by strategically placing the layers around the sphere depending on their connection with other layers, finding suitable node positions within the layer with a force-directed layout and by bundling edges from inter-connected nodes. Furthermore, the viewer should be able to navigate through the network in an easy and intuitive way. The navigation should allow the user to isolate a single layer, so that connections within a layer can be observed more closely. However, it should still be possible to recognize which nodes share connections to entities of other layers.

1.3 Methodology

To create a new layout for multilayer networks, we first had to analyze already existing solutions, to better understand where current weaknesses lie as well as possible improvements can be made. Section 3.2 shows a summary of our findings. We studied the theory behind graphs (see Section 2.1) as well as multilayer networks (see Section 2.2) to gather better insight into the underlying data. Afterwards, we defined clear requirements for our solution in Section 4.2. The implementation process can be summarized in four steps. Firstly, we implemented the general layout, which positions the layers strategically around a sphere. Secondly, we implemented edge bundling for inter-layer connections, to reduce visual clutter. Thirdly, we implemented the force-based layout to calculate

node positions within their respective layers, and finally, the layer view was implemented, which allows the viewer easier navigation to any layer. A detailed description of the implementation process can be found in Chapter 5. To evaluate our results, we compared our visualization to existing solutions and benchmark-tested our application to see how many elements could be visualized before the frames per second were too low to interact with the application. Additionally, we evaluated how many elements can be displayed before the visualization becomes too convoluted, due to visual clutter. The results can be found in Chapter 6.

CHAPTER 2

Background

2.1 Graph Theory

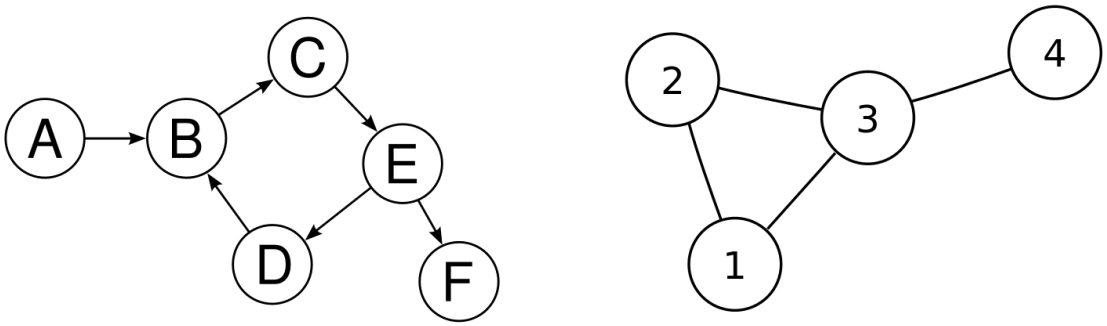
Before visualizing any data, it is important to understand what the given data represent. In our case, the data to be visualized is a graph or a network. In its most basic form, a graph consists of vertices and links [Gou12]. The vertices represent data points and links are the connections/relations between those data points. Therefore, graphs are used to visualize data points and the relationship between them. In most cases, simply showing a connection between different vertices, is not enough to represent the underlying data. Therefore, there are two different types of links:

- Directed links, which differentiate between source and target vertices. This means that the relationship between two entities can be unidirectional (see Figure 2.1a).
- Undirected links, where target and source vertices are interchangeable. Therefore, the connection of these entities is bidirectional (see Figure 2.1b).

Additionally, links can be weighted or unweighted:

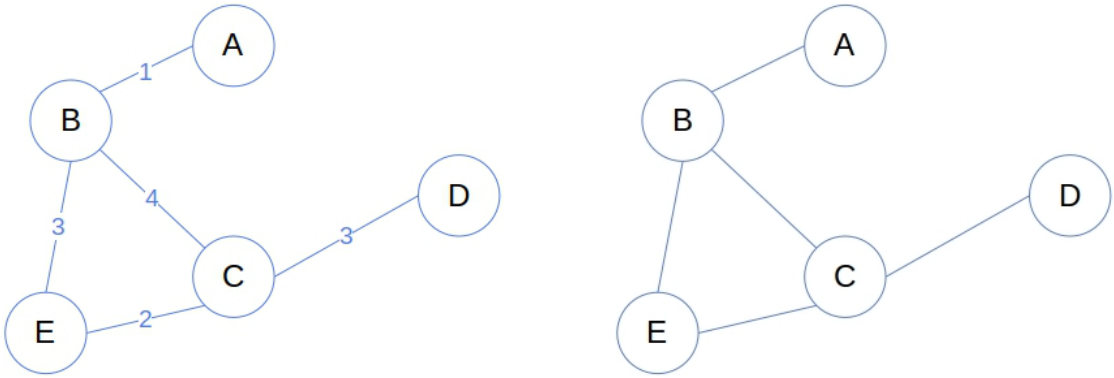
- Weighted links have an additional numerical value associated with them. This value usually represents the strength of the connection between the vertices (see Figure 2.2a).
- Unweighted links do not have any additional value to indicate the strength of the connection (see Figure 2.2b).

For this thesis, the terms “graph” and “network”, “vertices” and “nodes” as well as “links” and “edges” are used interchangeably.



(a) A graph with directed links [Dir]. (b) A graph with undirected links [Dir].

Figure 2.1: A comparison between directed and undirected graphs.



(a) A graph with weighted links [wei]. (b) A graph with unweighted links [wei].

Figure 2.2: A comparison between weighted and unweighted graphs.

2.2 Multilayer Networks

Visualizing more complex data in a simple graph can often lead to an oversimplification, where important details are hidden within the visualization. This has led to an increased effort to visualize more complex networks like multilayer networks in the past few years [KAB+14].

As an illustrative example, consider visualizing illnesses and their relationship to one another. A simple graph could be used to view which disease is the result of another, or how likely a patient is going to contract a certain illness given his or her current infection or condition. However, this simple graph would have problems to accurately represent the real world, given that there are real-world parameters that would be invisible in such an oversimplification. Diseases and their consequences can depend on a multitude of reasons – age of the patient, length of infection, general health of the patient and many more.

Therefore, multilayer networks introduce an additional element in a graph, which we call

the layer (see Figure 2.3). These layers are used as attributes for the vertices, and their purpose can greatly vary between different datasets. However, their basic use is to group different vertices together regardless of their relationship to one another.

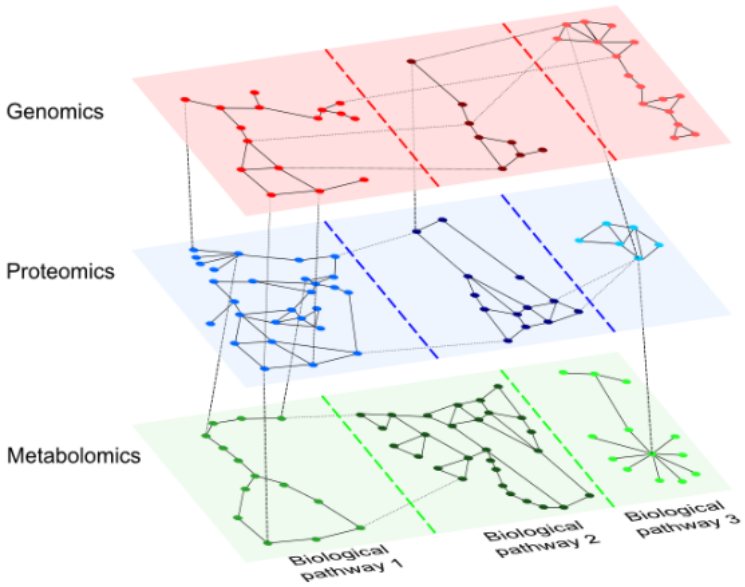


Figure 2.3: An example multilayer network, which shows layers as different colored boxes. Connections within the same color are considered intra-edges, connections from one color to another are considered inter-edges. Additionally, this dataset only has sequential inter-layer connections, as the green and red layer do not directly share connections [MGM+19].

By assigning the nodes to different layers (a node can be part of one or multiple layers), we have to take into account, that a node can have different links depending on the layer it is assigned to. By considering illnesses and their effects on different age groups as layers, we will most likely see that the consequences of any illness on younger people will have staggering differences in their development, compared to seniors with the same illnesses. This would lead to completely different connections of the same nodes in different layers. Additionally, the layers can also be used to visualize completely different networks and their connections inside one visualization. One layer could contain all the illnesses within the dataset, and another layer could contain symptoms a sick patient experienced, which might help health workers to determine the disease of a patient based on the symptoms the patient experiences.

Additionally, edges can be categorized based on the source and target node-layer. On the one hand, if two nodes lie within the same layer and share an edge with each other, we call this type of connection an *intra-connection*. On the other hand, if two nodes have a connection, but they are assigned to different layers, then the connection is called an *inter-connection*. Depending on the characteristics of the dataset, the inter-connections

can either occur sequentially (layers only have connections to the next and previous layer), or the layers can be connected without any limitation to one another.

To sum up, a multilayer network consists of vertices which are assigned to one or more layers. These vertices share connections with one another, which are either called intra-edges if both nodes are assigned to the same layer or inter-edges if they are not.

2.3 Drawing Graphs

Just looking through the data of a graph only provides the reader with a very limited insight into underlying correlations or other information within the data. To gain a deeper understanding, a good visualization of the provided data is necessary.
For any visualization, it is important to keep the absolute basics of information visualization in mind, including the famous saying "Overview first, zoom and filter, then details-on-demand" [Shn03], as well as the "Gestalt Principles" [KMV15]. Specifically, for graph visualization, it is important to keep in mind that the viewer considers objects in proximity and objects with similarities like the same shape or the same color to be in a group or related to one another.
While the color of nodes can easily be changed to signal a connection between different nodes, it is not as trivial to create a layout that uses the proximity of different nodes for the same effect. Therefore, most node-link diagrams in 2D and 3D rely on some form of a force-based layout to calculate the position of every node [VLKS+11]. These force-based layouts use different attracting and repelling forces, which should lead to nodes in proximity if their connection is strong, and nodes farther apart if they have no or a weak connection with each other. The implementation and usage of force-based layouts is further examined in Section 5.4.5.
Drawing multilayer networks comes with additional challenges, since nodes have additional attributes which have to be considered in the visualization. For node-link diagrams, which are the focus of this thesis, additional attributes are commonly encoded in either [NMSL19]:

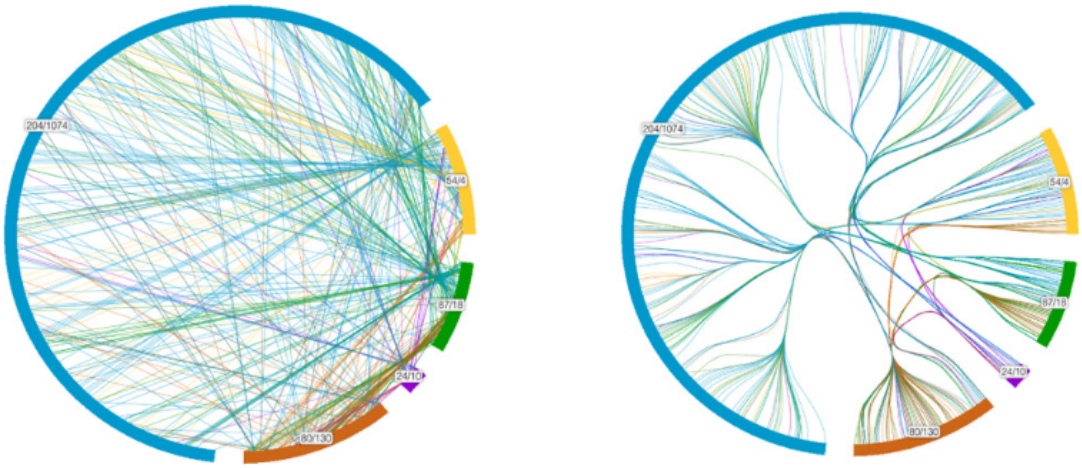
- the node or edge directly (e.g. coloring nodes / edges differently),
- different constraints for nodes (e.g. placing nodes in different regions based on given attributes), or
- into their position directly.

Chapter 4 explains how our visualization makes use of the first two points.

2.4 Edge Bundling

Edge bundling [Hol06] is by now a common technique to reduce visual clutter without completely sacrificing the details of a visualization. As we have already seen in the

previous chapter, links are usually depicted as straight lines between two connected nodes (see Figure 2.1b). However, visual clutter is a common problem for larger node-link diagrams, as too many edges (and nodes) make the graph appear tangled and hard to comprehend.
Therefore, edge bundling represents edges with splines which are rerouted through a control point, instead of drawing the link directly from its source to the target node. By routing for example all connections between the nodes of a layer pair through the same point (depicted in Figure 2.4b), the viewer can more easily determine how many edges the respective layers share and therefore get a better understanding of how strongly different layers are connected.



(a) Visualization without edge-bundling.

(b) Visualization with edge-bundling.

Figure 2.4: A comparison between the layout in Chapter 3.3 with and without bundled edges [CMF+14], clearly showing that bundling edges greatly reduces visual clutter.

2.5 Relevant Tasks

Lastly, considering which tasks a user would want to accomplish with a multilayer visualization, allows us to better analyze existing solutions for their strength and weaknesses. McGee et al. [MGM+19] provide a detailed overview about relevant tasks in a multilayer graph. The Findings show that most tasks in a multilayer network can be grouped into four categories:

C1 “Cross layer entity connectivity” - With these tasks, users want to explore the connectivity of one or multiple nodes across different layers.

C2 “Cross layer entity comparison” - Where the user compares different nodes (typically the connections of the same node) in different layers to one another.

C3 “Layer manipulation, reconfiguration” - These tasks manipulate the layer structure, such as merging or splitting a layer on certain conditions.

C4 “Layer comparison” - Where the user compares different layers based on either numerical attributes, such as node degree distribution or other attributes such as topological or connectivity patterns.

As these are the most common tasks for multilayer networks, a tool for their visualization should give the user the possibility to perform some tasks within the categories mentioned above. As we will see, most-state-of-the art visualizations show great support for tasks in the category C1 and C2, while tasks of the other categories are less commonly supported. However, it is important to keep in mind that not all the tasks a user wants to accomplish necessarily need a multilayer visualization. Identifying correlations within a layer, analyzing a node’s intra-connectivity or even just finding a specific node might be essential to individual users and are oftentimes neglected in multilayer visualizations.

CHAPTER 3

Related Work

3.1 3D Network Visualizations

Because the focus of this thesis is the visualization of multilayer networks and 3D, it is necessary to give a brief overview on general 3D information visualizations. Visualizing information in 3D has been practiced for many decades. Some well known examples include Cone Trees [RMC91], a visualization technique to depict hierarchical data in a three-dimensional tree-like structure. SemNet [FPF98] visualizes knowledge bases as a 3D node-link network by positioning the nodes randomly or based on their connectivity to other elements. Data mountain [RCL+98] utilizes a 3D environment for document management. These visualizations were designed to be observed on a two-dimensional screen, similar to our solution. They use animations through a 3D environment with different depth cues, such as lighting, kinetic depth, linear perspective and motion parallax, to trick the human brain into perceiving depth in the visualization [AHKMF11].

Contrary to two-dimensional approaches, navigating through the visualization is a key aspect of any 3D visualization, as occlusion of data prevents the user from seeing all visualized entities simultaneously. A study found that, due to occlusion of data, users take a longer time to complete navigation tasks in a Cone Tree [RMC91] visualization, compared to a 2D tree visualization [CM00]. Elmqvist and Tsigas [ET08] provided a detailed study about occlusion of data, as well as different techniques to manage occlusion in the visualization.
Applications visualized on a 2D screen, like the examples mentioned above, usually rely on mouse and keyboard actions to perform camera movements, animations and other interactions. However, due to the rise in popularity and the more affordable hardware of VR and AR technologies, there has been an increase in immersive network visualizations. Visualizing networks in immersive environments require new techniques for navigating and presenting the data. An example for presenting immersive networks is the solution by Sorger et al. [SWKA19], which offers the user the option to switch between two

perspectives when analyzing a dataset. An “overview” perspective gives the user the opportunity to examine the entire network. By selecting a node, the user can switch to a “detail” perspective, where the user can explore the local environment.
To offer the user a more intuitive navigation through the immersive network, many solutions rely on handheld controllers instead of mouse and keyboard. A more detailed discussion about common navigation techniques for visualizations in virtual reality can be found in a study by Drogemuller et al. [DCW+20].

3.2 Analysis of Multilayer Visualizations

By considering common tasks (see Section 2.5), we can determine weaknesses of current solutions more easily, and hypothesize that certain tasks could be performed more intuitively in a different or an altered solution. Therefore, we analyze state-of-the-art visualization approaches for their respective strengths and weaknesses, and use the analysis as inspiration for our solution.

One of the first things that needs to be considered when creating a new visualization technique, is how many dimensions can or should the dataset be visualized in. As we have already discussed, we group nodes with layers in a multilayer network and, therefore, also differentiate between two different types of edges: intra- and inter-edges. This is essential, because one of the techniques in visualizing multi-layer networks is to use different dimensions for intra- and inter-connected nodes.

3.2.1 Hive-Plot

A common example implementing this technique, is the "Hive-Plot"(see Fig. 3.1), which shows the intra-connections in one dimension and the inter-connections in two dimensions. The advantage of this technique is that the user intuitively groups the different layers together, because all nodes in a layer share a common characteristic (e.g. all nodes are placed on a line), which therefore makes tasks of the category C1 more intuitive for the user.

By visualizing the data within a layer in one dimension, it becomes harder to convey the connections within a group to the user. The Hive-Plot usually utilizes a node’s proximity to other nodes as an indicator for its connections. However, if the dataset to be visualized becomes larger and more connected, it is logical that a lot of information about intra-connections gets lost. Therefore, some implementations of the Hive-Plot visualize each group twice to allow intra-connection analysis of a network (as depicted in Figure 3.1). One of the advantages of the Hive-Plot is that it is easy to see if a node is heavily connected to another layer, due to the curved and slightly bundled edges. A downside is that it is only possible to visualize sequentially structured multilayer networks, because connecting edges from more than the previous and following layer would result in a lot of visual clutter. To counteract this problem, the Hive-Plot was also

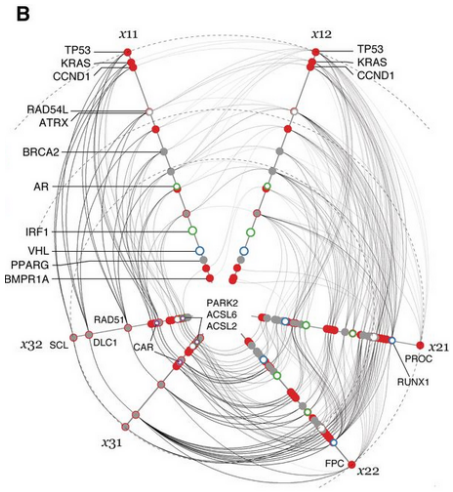


Figure 3.1: Example of a Hive Plot. The visualization shows the different groups (x11,x21,x31) represented by nodes on a line, while the connections between the groups are in 2D space in a radial layout [KBJM12].

implemented with its inter-edges placed in three dimensions [Han17]. As the Figure 3.2 shows, the 3D Hive-Plot allows the visualization of more than just adjacent layers.

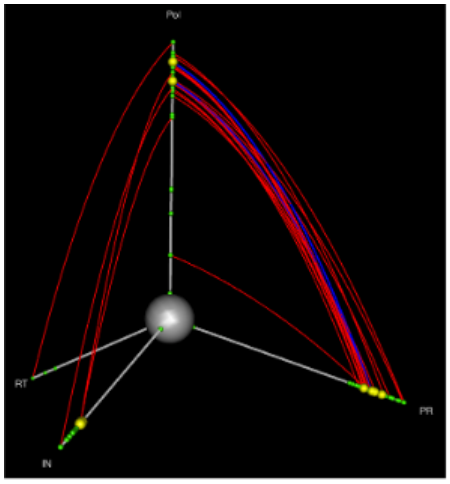


Figure 3.2: Example of a 3D Hive-Plot. The visualization shows the different groups represented by nodes on a line, the connections between the groups are now placed in 3D space [Han17].

3.2.2 Radial design with subgroups

Another example is the visualization technique for social networks by Crnovrsanin et al. [CMF+14]. The layout places the vertices around the perimeter of a circle. Nodes of

the same layer are placed along a circular section and share the same color. Additionally, connections within a layer are drawn outside the circle, while inter-connections are bundled within the circle (see Section 3.3).

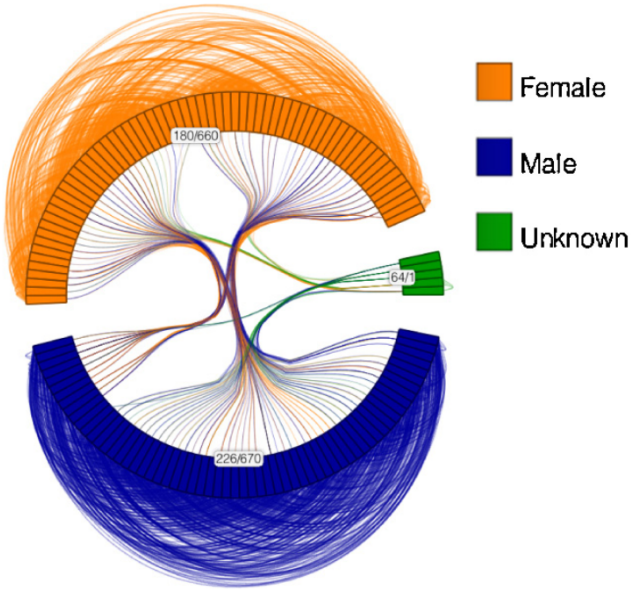


Figure 3.3: Radial layout approach, the nodes of the groups (female, male and unknown) are depicted along the circumference of a circle, the nodes of the individual layers get assigned to a circular arc. Intra-edges are drawn and bundled outside the circle and inter-edges inside the circle.

Similarly to the Hive-Plot, this technique focuses on visualizing the general flow of inter-edges. This allows the user to make claims about the general connectivity of different layers. However, as the example (see Section 3.3) clearly shows, connections within a layer are hardly recognizable due to visual clutter. Contrary to the 2D Hive-Plot, this approach also allows the visualization of connections between more than just adjacent layers, without the need of a third dimension.

3.2.3 Arena-3D

A further example is a framework developed for visualizing mutlilayer networks, which is called "Arena3D" [POS+08]. It was developed for analyzing biological networks. As Figure 3.4 shows their approach was to visualize nodes and their connections within a layer on a two-dimensional plane. The layers are then positioned in 3D space, for example by stacking the layers on top of each other or positioning the layers in a way that resembles a cube.

The nodes inside the layer can either be placed randomly, along a circle or with a force-based layout. However, the force-based layout only considers connections to same layer for a node's position.

Contrary to the other examples, this solution uses a three-dimensional approach, which additionally allows the visualization of connections within a layer, due to the additional design space of the third dimension. However, the random and circle layout inside the layer struggles with visualizing bigger datasets, as in both cases the nodes are essentially randomly placed, which produces additional edge-crossings for intra- and inter-connections. While the force-based layout scales better with bigger datasets it also struggles with many connections between different layers, as a node's position is only affected by other nodes in the same layer. Therefore, many connections between different layers will likely produce lots of visual clutter.

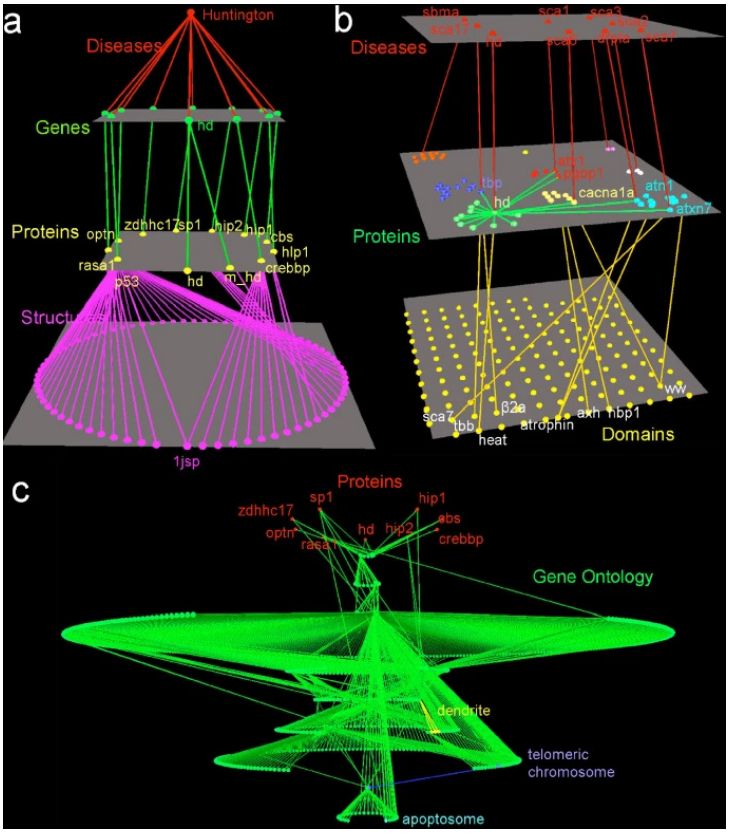


Figure 3.4: Three visualization examples from Arena3D, showing the general layout of the visualization, with nodes inside a layer positioned on a two-dimensional plane. In this case the planes are placed above and below each other.

3.3 Conclusion

By examining the two-dimensional approaches above, we concluded that these techniques are very useful to gain deeper insight into the graph's inter-connections. However, relations within a layer are either lost or hardly recognizable. Additionally, two-dimensional solutions often struggle to visualize non-sequential networks.
Current 2D and 3D approaches have problems with visualizing larger datasets. We already mentioned that this is a common problem with two-dimensional data visualizations, and while Arena3D is likely able to allow the analysis of a larger dataset than its 2D alternatives, it struggles to visualize many connections between different layers, because the positioning of the nodes are only determined by the intra-connections. This means that the edges between two layers will likely have many intersections, which in turn causes a lot of visual clutter.
Therefore, the idea of our solution was to create a mixture of the three solutions mentioned above. We chose to visualize the intra-connections in two dimensions on a plane like in Arena3D, and adapted the radial layout for the layers, which resulted in a spherical layout in three dimensions. Compared to Arena3D, we use the inter-connections for a force-based layout to position the nodes inside their respective layers and our solution is able to visualize non-sequential networks as the positioning of the layers around a sphere allows the visualization of connections between all layers, similar to the radial design. Additionally, we also adapted edge bundling for edges inside the sphere, similar to the radial approach, which substantially decreased visual clutter.
We hypothesize that by visualizing the intra-connections with a 2D node-link diagram, we can analyze connections within the layer and by placing the layers into the three-dimensional equivalent of the radial design, the user should still be able to get a good overview of the layer's inter-connections.

CHAPTER 4

Multilayer Network Visualization

4.1 Motivation

Currently, 2D layouts are still a common way of visualizing multilayer networks. However, as we have already pointed out, two-dimensional approaches quickly run into visual clutter as the size of the visualized dataset increases. Additionally, two-dimensional visualizations are either limited to sequential multilayer networks or sacrifice the readability of other aspects (e.g. the readability of intra-connections) in order to visualize connections between N layers, like some of the previously described two-dimensional solutions showed (see Section 3.2.1). Therefore, the motivation of this thesis was to create a new 3D multilayer layout, by analyzing current 1D and 2D state-of-the-art solutions and implementing a similar but optimized solution in three dimensions.

4.2 Requirements

We define the following requirements to tackle the problems described in prior chapters.

R1 The solution is able to visualize multilayer datasets with N layers.

R2 The solution is able to visualize multilayer networks with NxN connected layers.

R3 The solution should visualize both intra- and inter-edges.

R4 The solution can visualize larger datasets than 2D solutions before visual clutter obstructs a meaningful exploration of the visualized dataset.

R5 The solution should additionally implement ways to reduce edge-crossings and therefore visual clutter.

R6 If a node has connections to nodes from other layers, then ideally a user should be able to tell that these connections exist by the positioning of the node within the layer. The node should be positioned closer to other layers it shares a connection with.

R7 The user should be able to easily view the connections of a single layer.

4.3 Visual Encoding

Our solution uses a typical encoding to visualize graphs in three dimensions. The nodes are represented by spheres, the links are represented by tubes, and the bounding boxes of the layers are represented by rectangles. The layers size is equal the bounding box of the simulation (see Section 4.4.3) and each layer is assigned a unique color. The colors are generated using David Green's "CubeHelix" color scheme which is implemented in the library chroma.js [Ais22]. The size of the nodes is adjustable with a slider and to intuitively group nodes of the same layer a nodes color is set to the color of the layer. The diameter of the tubes is also set to a fixed size, and the color of the tubes depends on the type of the connection. Intra-connections are colored in the same color as the layer, inter-connections are colored in a color mix of the color of the source and target layer.

It is important to note that the layers are depicted as rectangles rather than spherical segments as intra-edges would otherwise be visualized in front of the nodes (when viewed from inside the sphere) rather than between the nodes. This would result in additional visual clutter, as intra-edges would also obstruct the view of the nodes inside the layer. A possible solution would be to curve the edges along the surface of the sphere. However, similarly to edge bundling, visualizing curved edges requires additional geometry and would therefore result in performance losses (see Section 6.1).

4.4 Layout

The first step was to determine the general layout of the visualization. The chosen layout was inspired by the radial layout with subgroups (see Section 3.2.2), which divides the nodes of each layer into different groups and places the groups around a circle. Depending on the size and amount of layers, the circle can either be enlarged or shrunk to fit all the layers around its perimeter. Additionally, this design uses the space inside the circle to visualize inter-connections, which allows the visualization of connections from multiple layers to any given layer. Therefore, this solution is already able to achieve the requirements R1, as well as R2. However, the layout struggles to visualize many intra-connections, because the nodes within a layer are placed along a line, which leaves little space between the nodes to depict all intra-edges and therefore makes it hard for the viewer to analyze connections within a layer.
To counteract this weakness, we visualized the individual layers in two dimensions, essentially drawing every layer as a standard node-link diagram, which additionally allows the visualization of intra-connections. Furthermore, the layers are placed around a sphere

instead of a circle as we have seen in the radial design. By utilizing additional dimensions, the solution is able to visualize more layers in a smaller space and keeps the visualization therefore more compact.

4.4.1 Spherical Layout

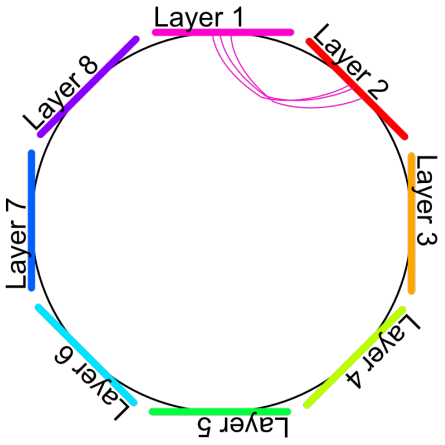


Figure 4.1: 2D sketch of our layout approach. Instead of a circle, the actual visualization places the layers around a sphere. The sketch shows how layers are generally placed around the sphere and how edges from one layer to another should be bundled (depicted between Layer1 and Layer2).

As we can see in the sketch (see Figure 4.1) similarly to the radial design, the core concept of our solution is to create a sphere large enough to fit all layers, so that no two individual layers intersect each other.
We calculate the minimum radius required to visualize all layers on a sphere, by summing up the bounding boxes of all layers and using the resulting area as the minimal area required for the sphere. Therefore, indirectly calculating the minimal sphere radius. However, because the layers have a rectangular shape, the layers will likely not perfectly fit on the sphere, which leaves gaps between some layers and might cause overlaps between other layers. To counteract this problem, we additionally implemented a multiplier to the radius, which is adjustable with a slider in the interface. In its default state, the multiplier is set to 1.5 times the minimal radius calculated above.

Placing the Layers around the Sphere

Afterwards, our solution uses different methods to determine points on the surface of the sphere where the layers should be placed. The chosen method depends on the number of layers:

- If the visualized dataset contains up to 6 layers, we use *fixed points* on the top, bottom, left, right, front and back of the circle for the layers and introduce additional points directly between two or three of the previously mentioned points for up to 14 layers.
- If the input data contains more than 14 layers the positions on the sphere are calculated by using the "golden spiral method" [SK97], which calculates roughly equally distributed points on the surface of a sphere.

We argue that the fixed points on the sphere achieve a more intuitive layout for the viewer compared to the calculated points, as the fixed positions guarantee a more symmetrical layout. However, the algorithmic approach allows the visualization of an unlimited amount of layers and therefore needed to fulfil R1.
It should be noted that both the fixed and calculated points require the layers to be of roughly equal size. If for example, one layer is significantly larger than the others the layers could overlap which has a detrimental impact on the visualization's readability. Therefore, the user interface allows the user to manually apply a multiplier to the radius of the sphere, as mentioned above.

Assigning Layers to the Points

After determining suitable positions for the layers on the sphere, we need to decide which layer should be assigned to which position. If the layers were randomly assigned to the positions on the sphere's surface, heavily connected layers might end up on the opposite side of the sphere, which creates unnecessary visual clutter and makes it harder for the user to comprehend the visualization, as viewers consider objects in proximity related to one another.
Therefore, an algorithm was implemented to place the most connected layer to the closest available position. The Algorithm 4.1 needs two arguments, the amount of layers in the dataset and the available positions on the sphere mentioned above. Simply put, the most inter-connected layer (the layer with the most inter-edges) is assigned the first available position (line 11-14). Afterwards, we place the most connected layer to an already placed layer - in the closest unoccupied point to that most connected placed layer (line 17 - 21). This continues until all layers are "placed". In other words, the algorithm considers all layers which have not been placed yet, chooses the one that has the most connections to one of the placed layers, determines the closest point available to the layer it shares the most connections to and places that layer on the closest available point.

4.4.2 2D Force-Layout

After the positions of the layers are calculated, the positions of the nodes within the layer need to be determined. Therefore, we implemented a 2D force-based layout for the nodes inside the layers, which considers both intra- and inter-connections (for implementation details, see Section 5.4.5). This means that intra-connected nodes tend to be closer to

```
function choosePointsForLayers(amountOfLayers, points) {
    if (amountOfLayers <= 2) {
        return points //Positions do not matter for <= 2 layers
    }
    let unused_points = points;
    let ordered_points = new Array(amountOfLayers);

    let placed_layers = []
    let unplaced_layers = range(1, amountOfLayers)

    let mostConnectedLayer = getMostConnectedLayer(unplaced_layers);
    removeFromAndAddTo(unplaced_layers.indexOf(mostConnectedLayer),
        unplaced_layers, placed_layers);
    //Set Most Connected Layer to first point available
    ordered_points[mostConnectedLayer-1] = unused_points.splice(0,1)[0];

    while(unplaced_layers.length !== 0 ){
        let nextConnection = getMostConnectedLayerPair(placed_layers,
            unplaced_layers)
        let nextPoint = getClosestPoint(ordered_points[nextConnection[0]
            - 1], unused_points)

        ordered_points[nextConnection[1]-1] = unused_points.splice(
            unused_points.indexOf(nextPoint),1)[0];
        removeFromAndAddTo(unplaced_layers.indexOf(nextConnection[1]),
            unplaced_layers, placed_layers);
    }
    return ordered_points;
}
```

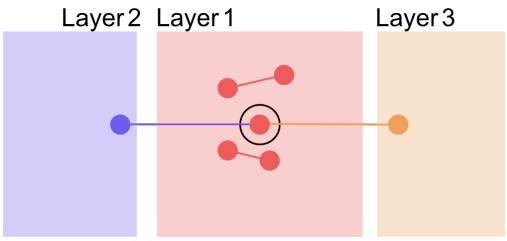
Algorithm 4.1: Algorithm to determine suitable points on the sphere for the available layers.

each other than nodes that do not share a connection. Furthermore, we also consider inter-connections in the force simulation. Therefore, nodes also tend to be positioned closer to the other layers they are connected to. It is important to note, that the word "tend" was carefully chosen, as this force-based layout does not guarantee that all nodes have all their connected vertices in proximity, as there are certain edges-cases (depicted in Figure 4.2), which make it impossible for both of the statements above to always be true:

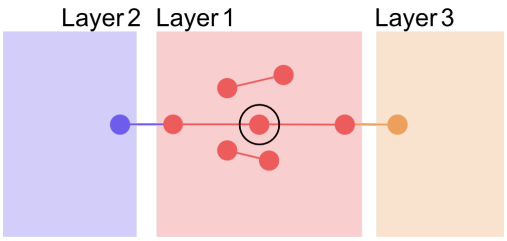
- The first edge-case occurs when a node a is connected to nodes of different layers and the layers are positioned on opposite sides of a's layer (depicted in Figure 4.2a). Therefore, the node a is equally pulled towards both inter-connections and ends up in the middle of the layer.
- A similar effect can occur when a node b is not directly connected to other layers,

but it's intra-connected nodes c and d share connections to different layers (depicted in Figure 4.2b). In this case the intra-connected nodes c and d are pulled towards their respective inter-connections. Similarly to the previous example, the node b is pulled towards both intra-connections and again ends up in the middle of the layer.

It is important to note that if one of the connections is stronger than the other, the force to the stronger connection will be greater and therefore push the node more into the stronger connected direction.



(a) Sketch showing a black-circled node that cannot be placed in proximity to its inter-connections, as the node is connected to two different layers which are on opposite sides.



(b) Sketch showing a black-circled node that cannot be placed in proximity to its intra-connections, due to their respective inter-connections.

Figure 4.2: Two sketches summarizing edge-cases which move connected nodes further apart and therefore making the resulting layout less intuitive and producing more visual clutter. The nodes of other layers are positioned in their respective force simulation.

4.4.3 Forces and Constraints

Our force-based system consists of a force simulation for each individual layer. Every simulation consists of the same forces and a bounding box as a constraint to calculate the positions of all the nodes in the visualization. The used forces and constraints in our force-system are as follows:

- **Many-Body Force:** The many-body force is a repulsive force for each node in the force simulation. The strength of this force decreases as the nodes get further apart, which results in an even distribution of the simulated vertices.
- **Link-Force:** Is an attractive force between a pair of nodes and the counter-part to the many-body force. The strength of the force depends on the strength of the connection between the two nodes. A stronger connection results in a stronger force, which leads to the two nodes being forced closer together.
- **Center-Force:** An attractive force towards the center of the layer. This force is necessary to ensure that nodes on the edge of the layer are placed on their respective position because of connections to other layers and not because the many-body force randomly distributes certain nodes towards said edge of the layer.

- **Box-Constraint:** A bounding box with an individually calculated size for each layer, the size depends on the amount of nodes the layer contains (the more nodes in the layer, the larger the area available to the simulation). If a node were to be pushed outside the bounding box by other forces, its position is instead put at exactly the edge of the box.

It is important to understand, that the forces mentioned above are balanced in such a way that each of these forces achieves its own goal without disturbing the other forces from accomplishing their goals. For example, by increasing the many-body force, the resulting visualization will undoubtedly be more evenly distributed. However, if the many-body force becomes too strong, the counteracting link force might not be strong enough to attract connected nodes, which would result in a randomly distributed network. Ideally, the resulting node positions are evenly distributed inside the 2D plane of each layer, while simultaneously positioning strongly connected nodes in close proximity.

Each simulation consists of a fixed amount of steps (300 steps) and additionally to the previously mentioned forces and constraints, each force simulation uses an *alpha value*, which decreases with every step of the simulation. The alpha value represents the "heat" of the simulation. A higher alpha value means the forces have a stronger impact on the node's position for the next step. Therefore, the simulation starts with stronger forces that pull the nodes roughly towards their ideal position. As the alpha value decreases, the forces become weaker and the nodes are smoothly tugged towards their final position.

For smaller datasets, this approach produced decent results. However, we stumbled upon a problem with bigger datasets. The issue was that by assigning all nodes to the center of the layer at the start of the simulation, the nodes were first catapulted outwards by the many-body force and then had to use the link-force to counteract the many-body force. However, especially with bigger datasets, the link-force was sometimes not strong enough to counteract the repulsion force of all other nodes randomly positioned between the two connected nodes and which caused some nodes to not be positioned in proximity to their connected vertices.

To counteract this issue, we first decreased the many-body force at the start of the simulation and increased the force over time, so that connected nodes are strongly pulled together at first and slowly separated afterwards. Secondly, we also assigned random positions to the nodes at the start of the simulation, instead of placing all the nodes in the center of the layer, which skips the catapulting of the nodes from the many-body force at the beginning of the simulation, and allows the simulation to immediately pull connected nodes together.

4.5 Edge Bundling

After placing the layers around the sphere and calculating the ideal positions for the nodes inside the individual layers, we face a similar issue as the radial design (see Figure 2.4):

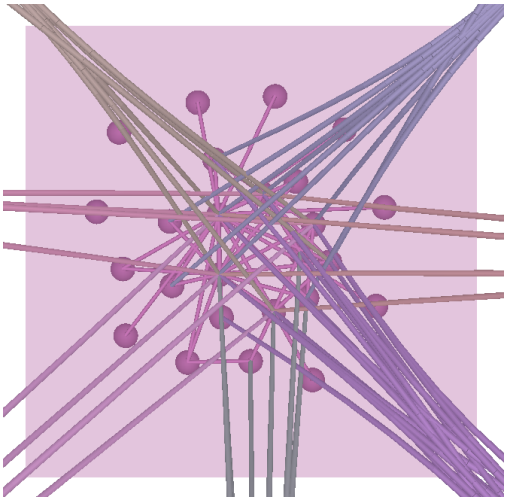
simply connecting inter-edges with straight lines or tubes results in a lot of visual clutter. Therefore, our solution implements edge-bundling (see Section 4.5) for inter-connected edges. In our solution, edge bundling was implemented by assigning a bundling point to each layer pair, where each inter-edge connection is rerouted through. The bundling point p_{bundel} is calculated based on the average position p_i of all nodes with inter-edges between the two layers, this average position p_{avg} is additionally translated either towards or away from the center of the sphere, so that the position is exactly at half the radius r of the sphere (see formula 4.1).

$$\begin{aligned} \vec{p}_{avg} &= \frac{1}{n}\sum_{i=1}^{n}\vec{p}_i \\ \vec{p}_{bundel} &= \hat{p}_{avg}\cdot\frac{r}{2} \end{aligned} \tag{4.1}$$

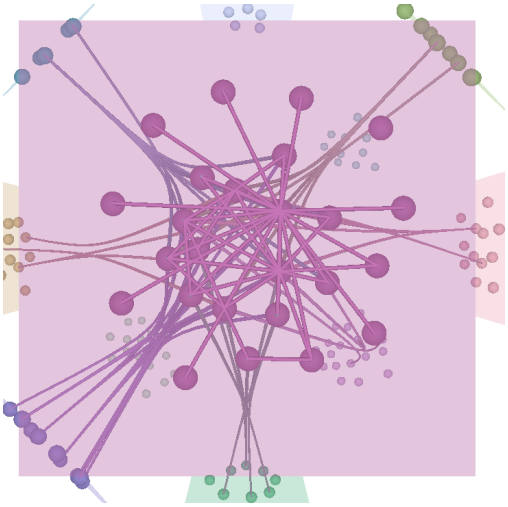
By taking the average position we guarantee that layers close to each other have their bundling point directly between those two layers, however, layers that are directly at the opposite side of the sphere would be bundled through or close to the center of the sphere, which produced a lot of visual clutter when the camera is positioned inside the sphere. Therefore, the average position is additionally translated to be at half the sphere's radius.

4.6 Navigation

To allow the user to analyze specific layers more easily and to achieve R7 a "layer view" was implemented. The view adjusts the camera position and rotation so that only the selected layer and its intra- and inter-connections are visible. All other connections (connections from other layers which are not relevant to the selected layer) are hidden in this view. Additionally, it is possible to view the layer from both the inside (see Figure 4.3a) and the outside of the sphere (see Figure 4.3b), this is useful as viewing the layer from the inside of the sphere highlights inter-connected edges, as these are the edges closest to the camera, while looking at the layer from the outside of the sphere highlights intra-connected edges.



(a) The “layer view” from within the sphere, highlighting inter-connected edges as these are the closest to the camera.



(b) The “layer view” from the outside of the sphere, highlighting intra-connected edges.

Figure 4.3: A comparison between the layer viewed from the outside and the inside of the sphere.

It is important to note, that the viewer is fully able to traverse through the visualization without using the layer-view by using intuitive mouse commands, however, this view is a useful addition if, for example, the current goal of the analysis is limited to nodes within a layer.
The layer view is controlled with the arrows keys (left and right) to jump between the layers, and the up key to toggle between putting the camera on the inside or on the outside of the sphere. To exit the layer-view the user simply has to click somewhere inside the visualization.

CHAPTER 5

Implementation

5.1 Existing Framework

We built our solution on top of an existing framework by Johannes Sorger [Sor]. The framework allowed the visualization of large scale networks on the web. It offered implementations for reading multilayer data, visualizing the data with a precalculated position, showing details on demand, tracking the analyzed link-path as well as camera animations. However, the framework had no functionality for generating a multilayer layout or navigating a multilayer network. A screenshot of the framework can be seen in Figure 5.1.
Therefore, the original framework was used for its basic rendering as well as the interaction capabilities mentioned above, and was extended with the features mentioned in Chapter 4.

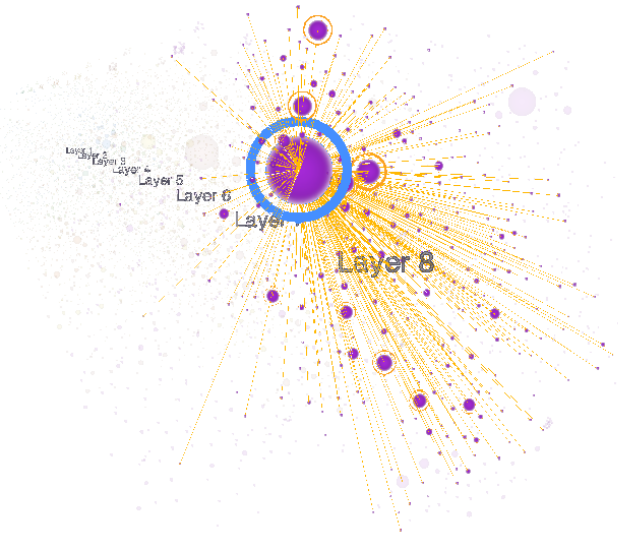


Figure 5.1: Screenshot of the underlying framework. [Sor]

5.2 Technology

The implementation and the existing framework are coded in JavaScript and run in all modern Browsers. The application was thoroughly tested in both Google Chrome (version 99.0.4844.51(64-Bit)) and Firefox (version 97.0.2 (64-Bit)). For simplicity, we continued using three.js [Dan12] as render engine which was already used by the existing framework (version 0.138.3). The final layout used additional libraries for the edge bundling and the force-based layout. For the implementation of edge bundling, our solution uses the library by Séguy [Thi22], which calculates positions on a curve based on weighted control points. For the implementation of the force-based layout we used the library d3-force.js [Bos22], a powerful tool for two-dimensional force simulations. The library has implementations for all forces described in Section 4.4.3 and handles the execution of the simulation. The main focus of our implementation was therefore reduced to balancing the forces, as well as custom behaviour for the inter-connections (see Section 5.4.5).

5.3 Data Structure

Before diving into the implementation of the project, we describe the input data required. The input data consists of two CSV files. One contains all the information of the data points (nodes - see Table 5.1). The rows for the color (r, g, b, a) range from 0 to 255 (expect the alpha value ranges from 0 to 1) and are optional. The program calculates a unique color for each layer and the user can choose to color the nodes like the layer colors instead of providing individual colors for each node. The relative position of the nodes inside the layer (x,y) are also not required. If the data is visualized for the first time, the positions will be calculated by the force-based simulation.

The second file contains all the information about the relations between those data points (links - see Table 5.2). It is important to note, that as for now, the positions of the respective rows are hard-coded. Therefore, other CSV-datasets would need to be brought to the same format or alternatively, the hard-coded positions would need to be adapted.

Label	ID	Description	Layer	r	g	b	a	x	y
Test1	0	Description1	0	155	255	255	1	0	0
Test2	1	Description2	1	100	125	60	1	10	10

Table 5.1: Minimal CSV input data for the nodes.

Source	Target	weight
0	1	0.1

Table 5.2: Minimal CSV input data for the edges. Source and Target refer to the source's node ID and target's node ID. The weight has to be positive and refers to the strength of the connection between the nodes.

5.4 Application Details

5.4.1 Program Structure

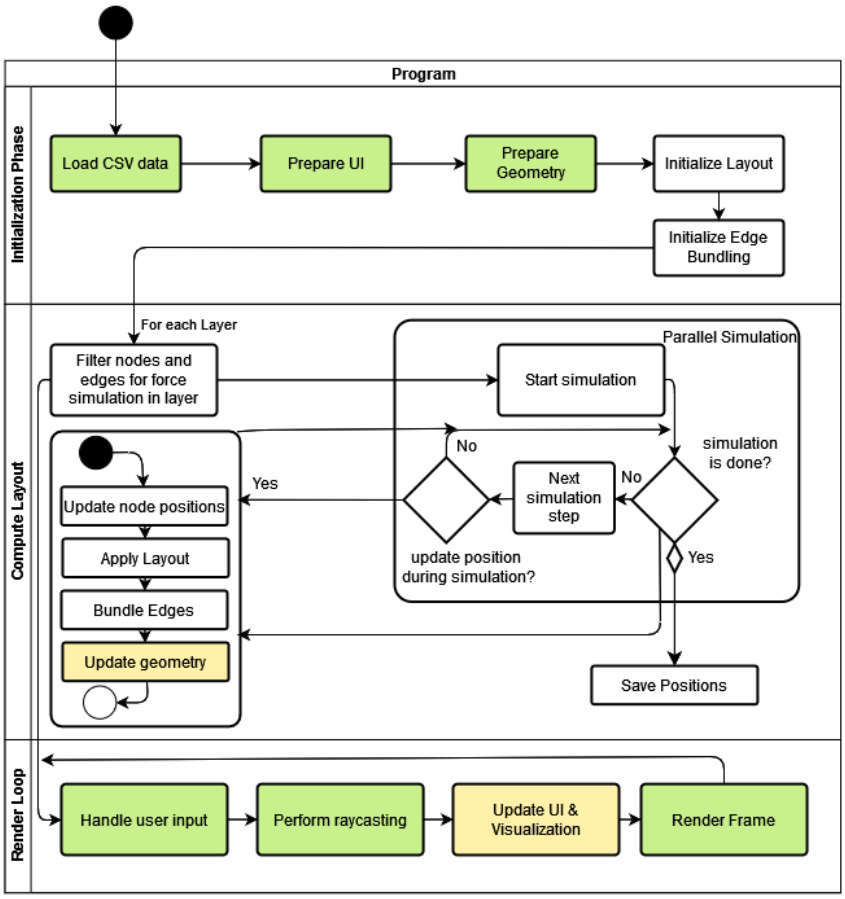


Figure 5.2: Flow chart of our application.

The application is structured into multiple processing steps. Figure 5.2 provides an overview about the steps necessary to visualize the input data. The processing steps colored in green were already largely available in the framework mentioned in Section 5.1, the steps colored in yellow were also already available in the framework but had to be adjusted, other processing steps were newly implemented. The program starts with an initialization phase (see Section 5.4.2). Afterwards, the program starts the force-based simulation to calculate the positions of the nodes (see Section 5.4.5). For performance reasons, the user can decide if the positions of the nodes during the simulation are rendered or if the nodes are stationary during the simulation and only the final position after the simulation is rendered. Whenever the nodes are updated, the layout (see Section 5.4.3) as well as edge bundling (see Section 5.4.4) is applied.

Since the force-based simulations are handled asynchronously, the program starts the render loop shortly after the initialization phase, where the visualization and UI are updated depending on the user input (see Section 5.4.6) and the current state of the force-based simulation.

5.4.2 Initialization Phase

The program begins by first loading in the CSV data for nodes and edges, followed by preparing the user interface and the geometry which represents the nodes, edges and layers. For performance reasons, we use instanced rendering for all elements inside the visualization. It is important to note, that these steps were largely already provided by the initial framework, apart from additionally rendering geometry for different layers as well as some additions to the UI.

5.4.3 Layer Layout

After the preparation described in Section 4.4.1, we can calculate the transformation matrix for each layer. Since the layers start off at the origin of the visualization, all we have to do is translate the layers to their respective point, and apply a "look at" matrix from the assigned point towards the center of the sphere.
Transforming the nodes inside the layer is now remarkably easy, since the node positions are calculated (in the force simulation) at their relative position in the layer, we can simply apply the layer-matrix to all nodes of the layer.

5.4.4 Edge-Bundling

Initialization

Before looking into the details on how edge bundling was implemented, we firstly need to figure out which edges should be bundled together. As we have already mentioned in Section 4.5 we chose to bundle inter-edges with the same source- and target-layer. In other words, all inter-edges which come from the same layer and end in the same layer share the same bundling point. For the calculation of the bundling point, see Formula 4.1.

B-Splines

The implementation of edge bundling was realized with b-splines [DB72]. Simply put, b-splines are used to generate smooth curves from n control points. The control points determine the shape of the curve. Additionally, weights can be assigned to these points which determine how strongly the curve should gravitate towards the control point. Finally, the curve will always start exactly at the first control point and end at the last one.

For our solution, we make use of three control points, the source-node position, the bundling point mentioned above and the target-node position, in that order. Now, all that is left to do, is rendering the generated curve. In our case, we opted to visualize the

curve by sampling the signal to get a series of positions on the curve and joining these positions with edges, instead of directly joining the source- and target-node with an edge. It is important to note, that this method generates additional links that need to be rendered, which means the visualization's performance will be worse if there are many inter-connected edges present. We decided to render nine edges per curve (for every inter-edge, nine edge segments need to be rendered). This is because in our experience, nine segments were enough to create a relatively smooth curve without heavily affecting the performance of larger datasets.

5.4.5 Force Directed Layout

A force-based algorithm was implemented, to optimize the node placement for the visualization. Because the individual layers are in 2D-space, the force-based algorithm was implemented with the JavaScript-library d3-force [Bos22], which allows simulating forces for particles in a two-dimensional environment.
The Algorithm 5.1 consists of a separate simulation for each layer, where only intra- and inter-nodes with a connection to the particular layer (as well as the edges between those nodes) are considered. As shown in Figure 5.3a not considering inter-nodes connected to the layer will result in a layout that can be used to view the individual layers in isolation. However, when all layers are visible connections outside the layer become cluttered. The simplified code version shows the general structure of the Algorithm 5.1, it is important to note, that multiple simulations run simultaneously, and each simulation is responsible for the nodes of one layer, but considers nodes from other layers (*fixed nodes*), which share a link to a node in the current simulation.

(a) Force-based Algorithm only considering intra-connections.

(b) Force-based Algorithm that considers both intra- and inter-connections.

Figure 5.3: Comparison between two simulations, one with and one without inter-connections. By considering inter-connected nodes for the visualization, the links clearly look more organized.

These *fixed nodes* will be considered stationary in the simulation, and get their positions updated outside the simulation. Therefore, we need to access the current position (updated in another simulation) of the individual *fixed node* (see Figure 5.4), and transform it with the transformation matrix for its corresponding layer into 3d space (see Figure 5.5).

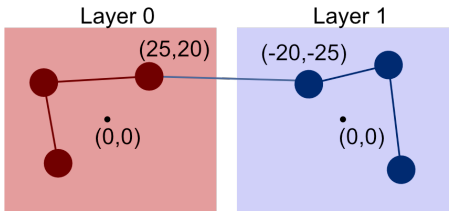


Figure 5.4: Nodes with position in their individual simulation.

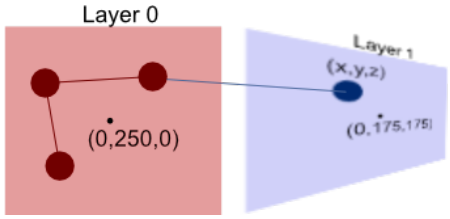


Figure 5.5: Nodes with position after transformation with their individual layer matrix.

This results in the relative position to the transformed simulated layer, but because the simulation is in 2D space at the origin (center at $x = y = 0$) we also need to apply the inverse transformation Matrix of the simulated layer (see Figure 5.6).

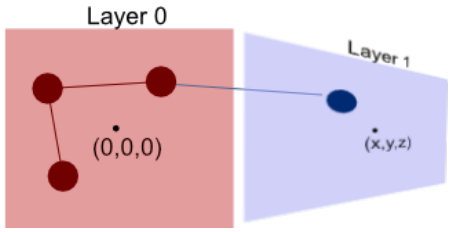


Figure 5.6: Nodes with position after transformation with inverse matrix of simulated layer.

Afterwards, the inter-connected nodes are transformed relative to the untransformed simulated layer. For the simulation, we are only interested in x- and y-values, and we can simply ignore the z-value. Because if only the z-values differ after the transformations, the layers are simply behind each other. The only problem is, that these links are oftentimes very long, since the layers can theoretically be very far apart, which would result in a disproportionately large pull in the simulation. Therefore, the relative position is set just outside the bounding box (see Figure 5.7).

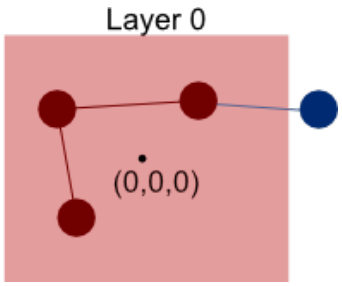


Figure 5.7: Nodes with position after setting the inter-connected node just outside the bounding box.

```
for each(let layer in layers)
    let nodes = getNodesInLayer() //Intra-Nodes
    let fixed_nodes = getNodesConnectedToLayer() //Relevant Inter-Nodes

    //The Bounding Box is calculated depening on the amount of nodes in
        a layer (more nodes means a bigger bounding box)
    let bounding_box = getBoundingBox(nodes.length)

    let all_nodes = [nodes, fixed_nodes] //All relevant nodes in
        simulation
    let edges = getEdgesBetweenNodes(all_nodes)
    let simulation = createSimulation(all_nodes, edges)

    while (!simulation.finished)
         for each(let node in fixed_node):
            //nodes from other layers get assigned a fixed position
                since their position is determined in a different
                simulation
            relativePosition = getNodePositionRelativeToLayer(node,
                layer)

            //Strength of the force depends on the length of the edge so
                 we set the relative Positon to the edge of the bounding
                 box if it is outise of it
            node.fixed_position = setToEdgeIfOutside(relativePosition,
                bounding_box)

        simulation.updatePositions()


function getCurrentNodePositionRelativeToLayer(node, simulated_layer)
    //The current local Position of the node with only x and y values
    pos_node = getPositionOfNode(node)

    //Transform local position to the position on the sphere.
    pos_node = pos_node.applyMatrix(getTransformationMatrix(node.layer))

    //Apply inverse Matrix of the simulated layer, so that we get the
        relative position to the untransformed simulated layer.
    pos_node.applyInverseMatrix(getTransformMatrix(simulated_layer))

    return pos_node
```

Algorithm 5.1: Simplified code of the force-based algorithm, in the real implementation the simulations run simultaneously

5.4.6 Layer-View

After computing the layout, the user can freely explore the visualization by either moving and rotating the camera with the mouse or by using the arrow keys to browse through the different layers with the layer-view. As we have already mentioned in Section 4.6 by using the layer-view only edges connected to the currently selected layer are shown by temporarily changing the opacity of other edges to 0.

The Formula 5.1 shows that the camera is positioned so that the entire layer is in the field of view fov regardless of the rotation of the layer. To calculate the position of the camera we form a triangle with two opposite corners of the layer and the cameras position. The fov of the camera can then be used to calculate the height h of the triangle with simple trigonometry (see Figure 5.8). To get the cameras position $\vec{p}_{camera}$ the position of the layer $\vec{p}_{layer}$ simply has to be multiplied by the height h plus or minus one depending on if the camera should be inside or outside the sphere. After positioning the camera, the camera's rotation is adjusted to look directly at the center of the layer.

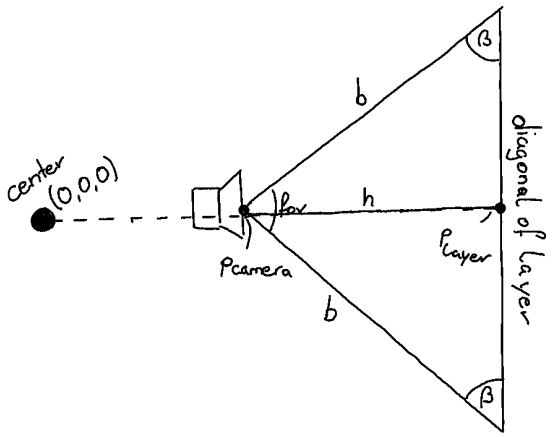


Figure 5.8: Sketch for calculating the optimal camera position to view a single layer. We chose to use the diagonal of the layer to be in the field of view fov of the camera, therefore the entire layer is always visible regardless of how the layer is rotated.

$$\begin{aligned} \beta &= \frac{(180 - fov)}{2} \\ h &= \frac{diag}{sin(\beta)} \\ \vec{p}_{camera} &= \vec{p}_{layer} \cdot (1 \pm h) \end{aligned} \tag{5.1}$$

CHAPTER 6

Results

In this chapter we show the results and findings of our solution. We discuss in detail how many entities can be rendered without dropping the frames per second (FPS) to an unusable level, or causing too much visual clutter to be analyzed in a meaningful way. Additionally, we show how long the application requires to set up the layout for differently sized datasets and discuss the readability of our layout by comparing it to other solutions.

6.1 Performance and Scalability

All performance evaluations have been carried out on a laptop with an Intel(R) Core(TM) i7-10510U CPU and an NVIDIA GeForce GTX 1650 and tested in Google Chrome. Any visualization that allows user interaction should have a constant frame rate, ideally higher than 30 FPS. In our experience, 20-23 FPS were enough to give the user a pleasant experience. However, any lower than 15 FPS and user interactions become increasingly irritating.
We tested our application with a randomly generated dataset, first by increasing the number of nodes, second by increasing the number of edges and lastly by increasing the amount of both nodes and edges simultaneously. The first two evaluations were carried out to see where the performance ceilings lie for links and edges respectively, and by increasing both nodes and edges at the same time, we simulated visualizing a real dataset.

We split up the evaluation into 3 different parts:

- The first step is the initialization, which is basically everything the program has to calculate before it can draw the final visualization, assuming that the positions are

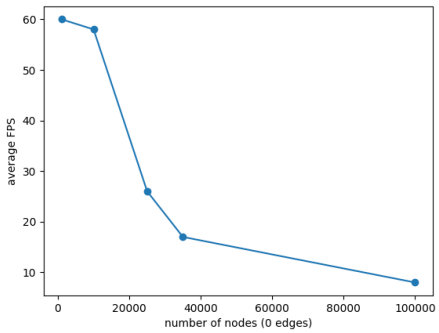
already calculated beforehand and therefore eliminating the need of simulating the positions with the force-based algorithm.

- The next step is the force-based algorithm, since the node and layer positions can be saved this step is only necessary for the first time a graph is visualized, afterwards this step can be skipped.

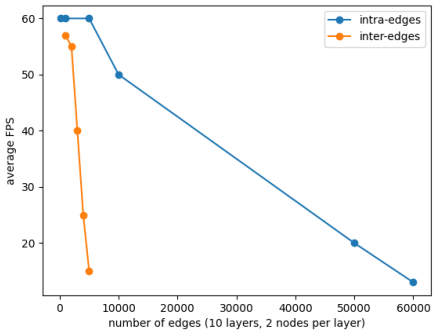
- The general FPS of the application during the render loop while handling user interactions.

As the Figure 6.1a shows, rendering up to 10,000 nodes are handled quite well by the application. When increasing that number even further, the visualization starts to lag when the camera is moved and at around 50,000 nodes, the visualization becomes barley usable for the viewer. The Figure 6.1b shows a very similar trend for intra-edges. Around 10,000 edges can be rendered without noticeably affecting the frame rate. Afterwards the frame rate drops relatively consistently and becomes unusable at around 50,000 intra-edges. However, due to edge-bundling, inter-edges require additional tube segments to be rendered and therefore cause an earlier drop in FPS, and the usability is influenced at around 4,000 inter-edges.

However, it is important to note that even though the overall user experience is not as smooth at 12 - 15 fps, the program still updates fast enough to be usable. As soon as the FPS drop below 10 user interactions become hard to control due to the input lag.



(a) Chart showing the frames per second with an increasing amount of nodes and no edges. The program starts to noticeably lag at around 30,000 rendered nodes and becomes hardly usable at around 50,000 nodes.



(b) Chart showing the frames per second with an increasing number of intra-edges and inter-edges split across ten layers with two nodes in each layer. The visualization starts to noticeably lag at around 50,000 intra-edges and 4,000 inter-edges.

Figure 6.1: Two charts showing the frames per second of our application against either the amount of nodes or the amount inter- / intra-edges present in the visualization.

By generating random datasets which contain an equal number of nodes and edges, we simulate data which resemble more realistic datasets. As the Figure 6.3a shows, the program can handle a dataset with a size of roughly 2,500 nodes and edges in the worst case scenario (a dataset with only inter-connected edges). In the best case scenario the program can visualize up to about 17,500 nodes and intra-connected edges, without dropping into critically low FPS.

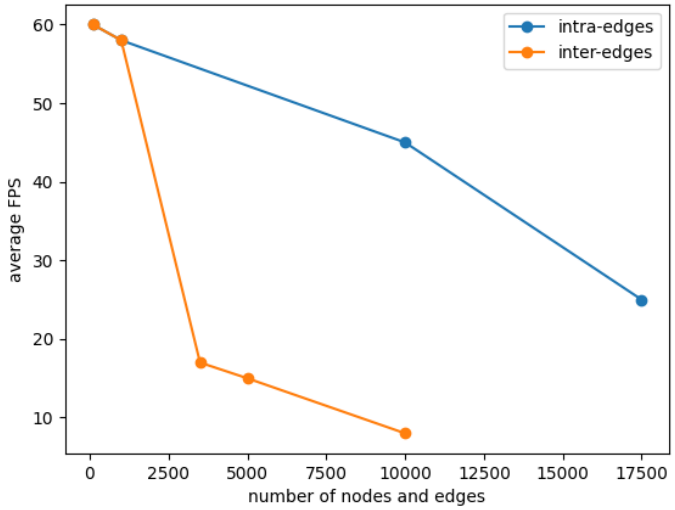
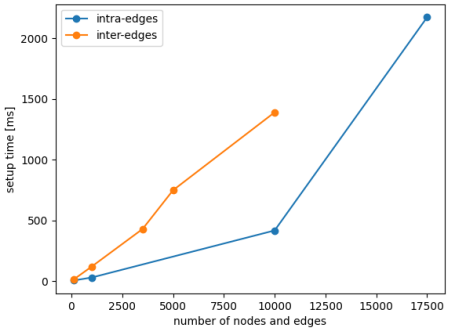


Figure 6.2: Chart showing the FPS with an increasing amount of edges and nodes. The orange line shows a network consisting of only inter-edges, while the blue line consists of only intra-edges. As expected, due to edge-bundling more edge segments have to be rendered per inter-edge, causing an earlier performance drop.

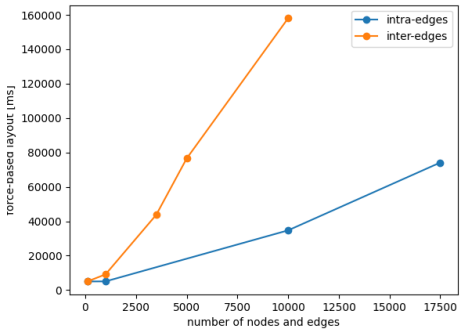
Furthermore, we analyzed the time our application requires to set up the visualization. As mentioned above, we split the setup time into two categories: the setup time without the simulation and the simulation alone. Because the simulation for the positions of the nodes only has to be carried out the first time a dataset is analyzed and can be saved and reloaded later.

On the one hand, Figure 6.3a shows that even though inter-connected edges need additional setup time due to edge bundling, the time required is below two seconds even for very large datasets with 10,000 nodes and edges. While testing the application, we found that the waiting time was, in general, barley noticeable. The force-based simulation on the other hand takes quite a bit longer compared to the rest of the layout (see Figure 6.3b). The results show that the inter-connections are again the bottleneck. This was expected as inter-edges add additional nodes to the individual simulation (as discussed in Section 5.4.5). While the simulation takes roughly five times longer for a dataset containing only inter-connected edges compared to only intra-connections, the

total time required in the worst case scenario for a dataset of 10,000 nodes and edges was only about two and a half minutes.



(a) Chart showing the total setup time the application needs before the visualization is displayed with an increasing number of nodes and edges (without the force-directed layout.)



(b) Chart showing the total time required for the force-directed layout with an increasing amount of nodes and edges.

Figure 6.3: Two charts showing the setup time of our application.

6.2 Readability

The focus of this thesis was to create a novel layout optimized for multi layer networks together with an application that allows users to visualize dataset. To accurately evaluate the visualization's readability, a user study would need to be conducted, where users would have to evaluate multiple datasets, complete tasks with the application and give formal and informal feedback. However, such a study would go beyond the scope of this thesis.

Instead, we will compare our visualization with other visualization tools and show where our visualization technique can be more helpful than currently existing solutions. We will use the same dataset to compare multiple multilayer network visualizers. The dataset is a multilayer network with diagnoses related to diabetes.

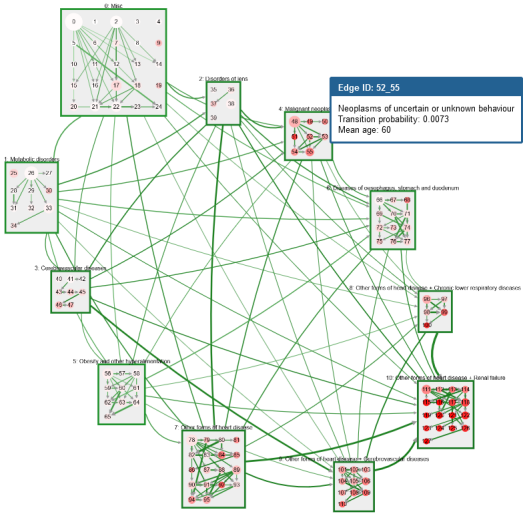


Figure 6.4: A 2D visualization that orders different layers within a circle. The color of the nodes shows the mortality of the diseases, the color of the edges shows the mean age of the patient.

The application depicted in Figure 6.4 was originally designed for this dataset. It is important to note that inter-edges have been reduced to one edge between layers in order to reduce clutter. The nodes inside a layer follow a more tabular design. While aggregating inter-edges results in less clutter, a lot of information about the direct connections between the individual nodes of a layers it also lost.

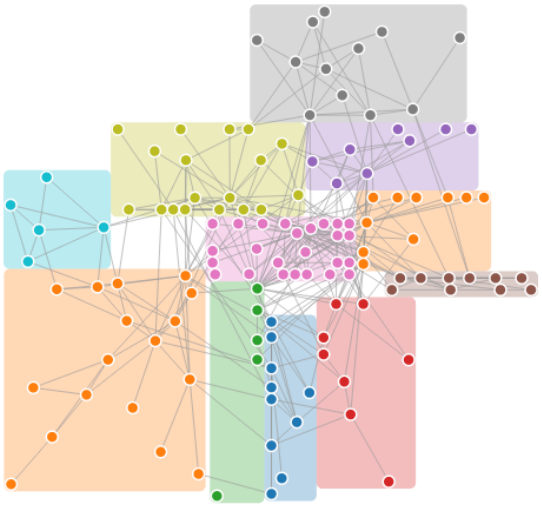
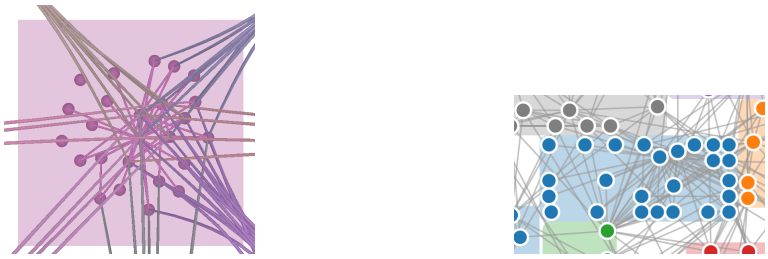


Figure 6.5: A 2D visualization that determines the layer position with a force-based layout. A different color is chosen per layer, the edge width shows the strength of the connection.

Another solution is an example from cola.js [Fra]. The solution depicted in Figure 6.5 visualizes the entire network on a 2D plane. Similarly to our design, layers are represented as different colored rectangles, which encompass all nodes assigned to that layer. The positioning of the nodes and layers are determined by the default force-based simulation implemented in the cola.js library. The visualization is challenged by heavily inter-connected layers. We can see that a lot of nodes are simply dragged towards the edge of the layer, making it hard to see the connections between them. Additionally, we can see why the previous visualization 6.4 aggregated the inter-edges, because the connections in the middle layer already produce a lot of visual clutter.



(a) A screenshot of the layer-view from our visualization.

(b) A screen shot of the layer visualized with cola.js.

Figure 6.6: A comparison between the visualization of our solution and cola.js. We argue that our visualization in (a) clearly gives the viewer a better understanding about which nodes have connections to other layers and to what layer they are mostly connected to.

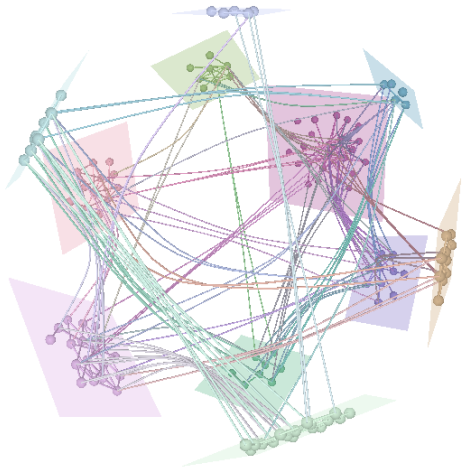


Figure 6.7: The same data set visualized with our application. The nodes and edges are given a custom color per layer. The layout within a layer is determined by a force-based simulation.

By observing the visualized dataset in our application from further away (see Figure 6.7),

we found out that it is harder to get a general overview about the whole visualization compared to the other mentioned solutions. Because our application works in 3D, edge crossings are dependent on the camera's position and by viewing the visualization from outside the sphere it becomes harder to determine connections between layers and nodes.

However, by taking a closer look with the layer-view (see Figure 6.6a), we can more easily understand individual connections towards intra- and inter-nodes. While the screenshot might look a bit cluttered at first, the position of the nodes already give a good idea with which layers the nodes are connected.

As we can see in Figure 6.6a, nodes with a strong connection to another node in a different layer are always positioned close to that layer (within the bounds of their own layer). Additionally, by hovering over a node only its connections are visible which gives the viewer the option to observe individual nodes and their connections more easily (as depicted in Figure 6.8).

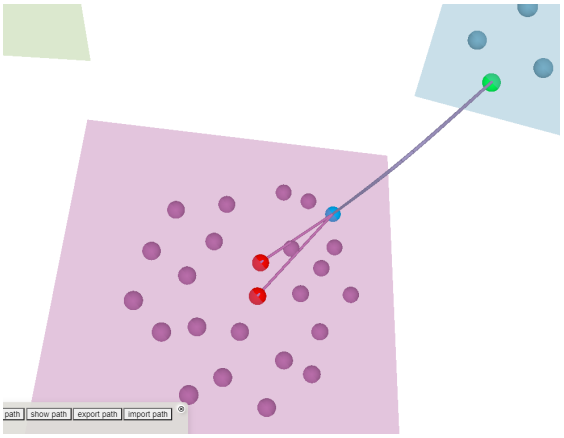


Figure 6.8: By hovering over a node only its intra- and inter-connections are visible.

6.2.1 Scalability

Lastly, we discuss the scalability in terms of readability of our solution. Therefore, we analyze how many nodes and edges can be rendered before visual clutter makes it difficult to derive meaningful information from the visualization. Therefore, we generated increasingly large datasets, because there were not enough real datasets available to us. We analyzed how visual clutter accumulates with increasing elements in the visualization. It is important to note, that randomly generated datasets might help to find general problems which cause visual clutter, but they are not necessarily representative of what real data looks like in our solution, as the connections in real datasets usually show patterns and correlations and are not randomly distributed. Therefore, we additionally analyzed one larger real dataset to see if our previously made assumptions also occurred when testing real data.

The first observation is that, due to edge bundling inter-connections do not clutter the visualization as much as the connections within the layers. This is because intra-connections are limited to two dimensions and therefore face similar problems as the two-dimensional alternatives mentioned in Chapter 2. This means that a sensible visualization in our application is limited to the nodes inside the layer rather than the number of layers present in the dataset.

By closely observing the layout inside a layer, we observe three factors which amplified the visual clutter present in the visualization:

- **Amount of Intra-Connections:** As mentioned above, due to the visualization in a two-dimensional space, the amount of connections inside a layer are the biggest source of visual clutter in our visualization.
- **Amount of Nodes:** The amount of nodes and edges generally correlate and therefore both entities are responsible for the visual clutter caused. We found that the visualization appears more cluttered if the average intra-connected node-degree in the visualized dataset is higher, unless the dataset contains only a small amount of nodes (< 30) inside each layer.
- **Amount of Inter-Connections:** While the inter-edges themselves cause only a small amount of the visual clutter present in the visualization, because inter-connections alter the positioning of nodes connected to other layers (as described in Section 4.4.2). Visual clutter accumulates inside the layer with increasing inter-connections, especially if many nodes inside a given layer are inter-connected to the same layer. This happens because all the nodes connected to other layers are dragged towards that layer inside the layer's plane. Therefore, many nodes are dragged towards the border of the layer, which leaves little space to visualize intra-connections (as depicted in Figure 6.9).

Due to the last effect, our visualization works better for non-sequential multilayer networks, as connections to multiple layers spread the nodes inside the layer to different edges, which reduces the effect mentioned above.

The visualization of a real dataset (with 8,649 nodes over 8 layers and 75,253 edges), which is depicted in Figure 6.10 shows that a combination of all the effects mentioned above are producing visual clutter. The dataset depicts different illnesses as nodes and age ranges as layers. Most of the visual clutter in the visualization (see Figure 6.10) is produced because the layer is only connected to two layers and shares significantly stronger connections to the layer positioned on its top right, therefore most nodes are pulled towards that direction. This effect immediately signals the user which layers are strongly connected. However, it also worsens the layout inside the layer, as there is less space to visualize intra-connections. This in turn also forces intra-connected nodes which do not share connections to the other layers towards the same side of the visualization.

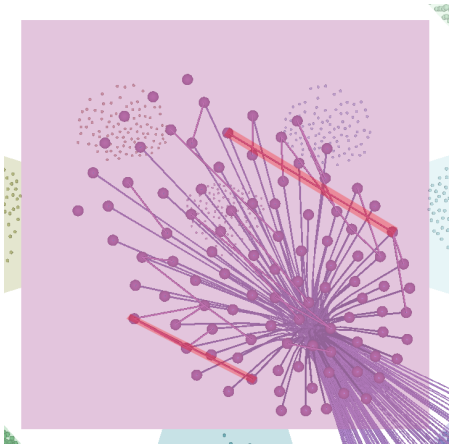


Figure 6.9: Layer-view of a randomly generated dataset, with a high inter-connected node degree. This shows that many inter-edges cause some intra-connections (marked in red) to be farther apart, because the inter-connected nodes are dragged towards the edge of the layer and the many-body force pushes all intra-connected nodes further away.

This then leaves less space to visualize all intra-connections, so that the connections are densely packed and appear cluttered.

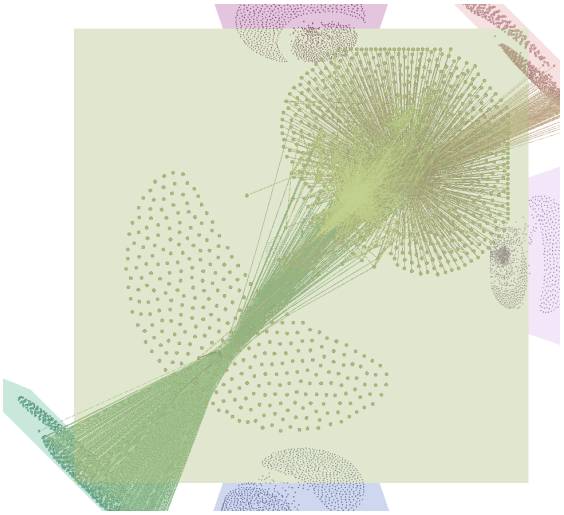


Figure 6.10: Layer of a dataset containing illnesses as nodes and different age ranges as layers. This shows a lot of visual clutter is produced by the intra- and inter-connections. The layout is dragged towards the top-right, due to the inter-connections which in turn drag intra-connected nodes further to this side, leaving little space to visualize all the connections inside the layout.

Conclusion

Even though the dataset with roughly 8,500 nodes spread across 9 layers appears generally cluttered. The possibility to filter edges by hovering a single node, in addition to the layer-view, allows the user to still remove some clutter present in the visualization.

In general, we predict that our solution struggles to visualize only sequentially connected multilayer networks, because inter-connections shift most of the nodes to one or two edges, leaving little room for the connections inside the layer. We argue that a non-sequential dataset (see Figure 6.11) with roughly the same size (around 1,000 nodes per layer), produces a vastly better layout, as the nodes are pushed towards multiple edges and are distributed more evenly. However, the amount of inter-connections in the background already makes it hard to decipher some of the nodes and their connections. Therefore, a network of this size (1,000 nodes per layer) is likely close to the maximum amount of entities visualizable without producing too much visual clutter to allow any analysis.

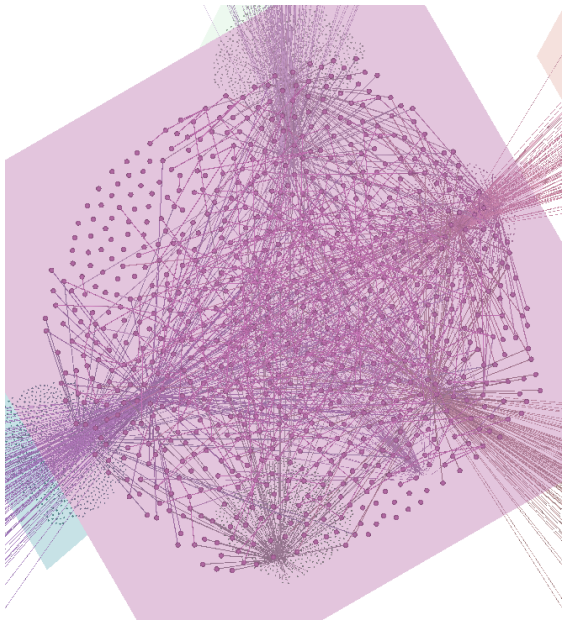


Figure 6.11: Randomly generated dataset with 10,000 nodes (1,000 nodes per layer) and a non-sequential multilayer data structure. Considering a lot of entities are visualized, the visual clutter is relatively low.

Additional visual clutter produced by more layers present in the dataset will hardly affect the visualization, as densely connected layers will always be positioned relatively close to one another and the layer-view can be used to additionally remove irrelevant nodes and edges for a given layer. However, as we already mentioned, multiple effects cause visual clutter in our visualization. Therefore, the number of nodes visualizable largely depends on the characteristics of the visualized data (i.e. average node-degree, amount

of inter-edges compared to intra-edges, how many layers a layer is connected to) .

CHAPTER 7

Conclusion and Future Work

The aim of this thesis was to propose a layout optimized for multilayer analysis, which utilizes the advantages of other two-dimensional state-of-the-art methods as well as the advantages of visualizing in three dimensions. Additionally, the solution provides an intuitive view to analyze intra-connections as well as inter-connections within a single layer and an optimized layout to analyze connections between multiple layers. Data scientists can use this layout to get better insight into multilayer dataset than with conventional two-dimensional alternatives. To create our solution, we extended an existing framework [Sor]. Our solution included creating an entirely new layout, edge bundling to reduce visual clutter, an optimized navigation method to analyze connections of a single layer, and a force-based simulation to determine suitable node positions within our layout, by considering both their intra- and inter-connections.

To our knowledge, this thesis provides a first approach on using a force-based layout that considers intra- and inter-connected edges to determine the node positions within a two-dimensional plane. Which allows the user to gather information about a node's inter-connection by analyzing the node's position within the layer.

We mentioned in Chapter 4 that our approach has some known weakness. One of the weaknesses, is the positioning of the nodes with connected layers on opposite sides (described in Section 4.4.3). A possible solution might be to position the layers on the sphere by considering more than just the most connected layer to make sure that two layers which share connections with a third layer are not placed on opposite sites of that third layer. Additionally, as we have described in Chapter 4 the current layout works best if the layers are of roughly equal size. Future work could improve this solution by additionally considering the bounding box size of different layers when generating suitable positions on the surface of the sphere.

As a final note, during the implementation we made sure that our code could be easily extended in the future, in particular for the positions of the layers, as well as the bundling

of the edges. Therefore, arranging the layers in a different layout (i.e. positioning the layers in a half sphere, or cube) could be implemented and the other functionalities (edge-bundling and force-based layout) should work without any adjustments. Another possible extension would be to implement a different strategy for calculating the bundling points (see Section 4.5), for example to further separate the inter-connections, and thereby reducing visual clutter produced by these connections, some edges could be bundled through a point positioned on the outside of the sphere, if the layers are positioned in proximity.

Bibliography

[AHKMF11] Basak Alper, Tobias Hollerer, JoAnn Kuchera-Morin, and Angus Forbes. Stereoscopic highlighting: 2d graph visualization on stereo displays. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2325–2333, 2011.

[Ais22] Gregor Aisch. chroma.js. `https://gka.github.io/chroma.js/`, 2022.

[Bos22] Mike Bostock. d3-force. `https://github.com/d3/d3-force`, 2022.

[CM00] Andy Cockburn and Bruce McKenzie. An evaluation of cone trees. In *People and Computers XIV—Usability or Else!*, pages 425–436. Springer, 2000.

[CMF+14] Tarik Crnovrsanin, Chris W Muelder, Robert Faris, Diane Felmlee, and Kwan-Liu Ma. Visualization techniques for categorical analysis of social networks with multiple edge sets. *Social Networks*, 37:56–64, 2014.

[Dan12] Brian Danchilla. Three. js framework. In *Beginning WebGL for HTML5*, pages 173–203. Springer, 2012.

[DB72] Carl De Boor. On calculating with b-splines. *Journal of Approximation theory*, 6(1):50–62, 1972.

[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, 2020.

[Dir] Images for directed vs. undirected graphs. `https://pediaa.com/what-is-the-difference-between-directed-and-undirected-graph/`. Accessed: 22-02-21.

[DRST14] Steve Dübel, Martin Röhlig, Heidrun Schumann, and Matthias Trapp. 2d and 3d presentation of spatial data: A systematic review. In *2014 IEEE VIS International Workshop on 3DVis (3DVis)*, pages 11–18, 2014.

[ET08] Niklas Elmqvist and Philippas Tsigas. A taxonomy of 3d occlusion management for visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(5):1095–1109, 2008.

[FPF98] KM Fairchild, SE Poltrock, and GW Furnas. Three-dimensional graphic representations of large knowledge bases. *Cognitive Science and its Applications for Human-Computer Interaction*, pages 201–234, 1998.

[Fra] Max Franz. Simple groups cola.js. `https://ialab.it.monash.edu/webcola/examples/smallworldwithgroups.html`. Accessed: 2022-04-11.

[Gou12] Ronald Gould. *Graph theory*. Courier Corporation, 2012.

[Han17] Bryan A Hanson. The hiver package. 2017.

[Hol06] Danny Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on visualization and computer graphics*, 12(5):741–748, 2006.

[KAB+14] Mikko Kivelä, Alex Arenas, Marc Barthelemy, James P Gleeson, Yamir Moreno, and Mason A Porter. Multilayer networks. *Journal of complex networks*, 2(3):203–271, 2014.

[KBJM12] Martin Krzywinski, Inanc Birol, Steven JM Jones, and Marco A Marra. Hive plots—rational approach to visualizing networks. *Briefings in bioinformatics*, 13(5):627–644, 2012.

[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*, pages 558–560, Cham, 2015. Springer International Publishing.

[MGM+19] Fintan Mcgee, Mohammad Ghoniem, Guy Melançon, Benoit Otjacques, and Bruno Pinaud. The state of the art in multilayer network visualization. In *Computer Graphics Forum*, volume 38, pages 125–149. Wiley Online Library, 2019.

[NMSL19] Carolina Nobre, Miriah Meyer, Marc Streit, and Alexander Lex. The state of the art in visualizing multivariate networks. In *Computer Graphics Forum*, volume 38, pages 807–832. Wiley Online Library, 2019.

[POS+08] Georgios A Pavlopoulos, Seán I O'Donoghue, Venkata P Satagopam, Theodoros G Soldatos, Evangelos Pafilis, and Reinhard Schneider. Arena3d: visualization of biological networks in 3d. *BMC systems biology*, 2(1):1–7, 2008.

[RCL+98] George Robertson, Mary Czerwinski, Kevin Larson, Daniel C Robbins, David Thiel, and Maarten Van Dantzich. Data mountain: using spatial memory for document management. In *Proceedings of the 11th annual ACM symposium on User interface software and technology*, pages 153–162, 1998.

[RMC91] George G Robertson, Jock D Mackinlay, and Stuart K Card. Cone trees: animated 3d visualizations of hierarchical information. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 189–194, 1991.

[Shn03] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *The craft of information visualization*, pages 364–371. Elsevier, 2003.

[SK97] Edward B Saff and Amo BJ Kuijlaars. Distributing many points on a sphere. *The mathematical intelligencer*, 19(1):5–11, 1997.

[Sor] Johannes Sorger. Network viewer. `https://vis.csh.ac.at/netviewer/`. Accessed: 2022-03-10.

[SWKA19] Johannes Sorger, Manuela Waldner, Wolfgang Knecht, and Alessio Arleo. Immersive analytics of large dynamic networks via overview and detail navigation. In *2019 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*, pages 144–1447. IEEE, 2019.

[Thi22] Séguy Thibaut. Library for b-splines. `https://github.com/thibauts/b-spline`, 2022.

[VLKS+11] Tatiana Von Landesberger, Arjan Kuijper, Tobias Schreck, Jörn Kohlhammer, Jarke J van Wijk, J-D Fekete, and Dieter W Fellner. Visual analysis of large graphs: state-of-the-art and future research challenges. In *Computer graphics forum*, volume 30, pages 1719–1749. Wiley Online Library, 2011.

[wei] Images for weighted vs. unweighted graphs. `https://www.baeldung.com/cs/weighted-vs-unweighted-graphs`. Accessed: 22-02-21.