ChatGPT

Graph Network Visualization in an Obsidian Plugin: Frameworks, Components & Theory

UI Frameworks and Charting Options for an Obsidian Plugin

Web-Based vs. Native Approaches: Since Obsidian is an Electron-based app (essentially a desktop web app), its plugins run in a web environment. This means we'll primarily leverage web UI frameworks and libraries, though it's useful to note native frameworks for completeness. On the web side, developers commonly use libraries like **D3.js** for custom interactive visuals or higher-level charting libraries (e.g. Chart.js, ECharts) for standard charts [1]. However, standard chart libraries are less relevant here because our focus is on custom network graphs rather than bar/pie charts. In a native desktop context (outside of Obsidian), one might consider UI toolkits like Qt (C++), JavaFX/Swing, or .NET WPF for rendering graphics, or even game engines (Unity, Unreal) for immersive 3D visualization. Those offer powerful native 3D/2D drawing capabilities, but they are not directly applicable within Obsidian's web-based plugin environment. Given that **we are building an Obsidian plugin**, the practical route is to use **web technologies** – effectively treating the Obsidian canvas as a browser. This allows using HTML/CSS for UI elements and either SVG, Canvas, or WebGL for custom drawings.

Obsidian's Native UI and API: Obsidian's plugin API provides some basic UI facilities (for example, modals, setting tabs, and the ability to manipulate the DOM) but it doesn't enforce a specific framework. Many plugin developers use plain TypeScript/JavaScript and DOM manipulation to inject panels or views. Others integrate frameworks like React or Svelte to manage complex UIs within a plugin. (There's even a community project, *Ophidian*, which acts as a component framework to help structure large plugin codebases [2] [3].) For simple needs – e.g. adding buttons, dialogs, forms – one can use Obsidian's provided classes or minimal CSS to match the app's look. But for a rich, custom visualization like a graph view, we will likely create a dedicated canvas or WebGL context inside a plugin pane. In summary, **the UI will be web-based**, using Obsidian as a host. We can incorporate both high-level libraries and low-level drawing as needed: for instance, employing a proven **visualization library** for networks or 3D, and layering Obsidian's UI elements (controls, dialogs) around it.

Relevant Visualization Libraries: For our **network graph rendering**, specialized JS libraries are extremely helpful. Popular choices in 2025 for graph/network visualization include:

- **Cytoscape.js** – a robust library for graph theory and network visualization [4], known for handling large graphs with many layout algorithms and analysis features.
- **Sigma.js** – a lightweight, WebGL-based renderer optimized for large-scale graphs [5]. Sigma provides smooth interaction and can handle thousands of nodes efficiently.
- **vis.js** – an older but versatile library that supports network diagrams with physics (and even has a 3D graph option) [6] [7]. It's easy to set up and highly customizable.
- **Graphology** – a JavaScript graph analysis library often used with Sigma for data handling [8]. It's not a renderer itself but can complement visualization by computing metrics (centrality, etc.).

- **Three.js** – the go-to WebGL library for 3D scenes [9]. While not specific to graphs, it provides the foundation to draw 3D objects and could be used to plot nodes as spheres and edges as lines in space, with full control over cameras, lighting, and effects.
- **Babylon.js or A-Frame** – other 3D frameworks (Babylon is a game engine, A-Frame for VR) [9]. These might be overkill for our needs, but they indicate the range of WebGL tools available if we wanted a more immersive or VR-capable graph view.

Because *we plan to create our own charts (graphs)* and not rely on Obsidian's default, we have freedom to choose or mix these libraries. A likely approach is to use a high-level network lib (for physics layout, node-edge data handling) in combination with a 3D engine for rendering. For example, one could use **Three.js** for rendering the 3D "outer space" scene of nodes, and incorporate a force-directed layout algorithm from something like d3-force or Sigma's engine to position nodes. In fact, the existing Obsidian community plugin "Juggl" used a force-directed graph in a 2D canvas. It improved on the default graph but still "lack[s] the structural insight" that more advanced analysis could provide [10]. Since we want a *refined, polished result*, we'll likely go beyond basic force layouts – integrating smooth physics (so nodes drift nicely into place), graceful interaction controls, and possibly analytical enhancements (more on that in later sections).

To summarize this section: **we will leverage web visualization frameworks** for the heavy lifting of drawing and interaction. This might involve a combination of libraries (e.g. a network visualization lib for graph data + a 3D rendering lib for visuals). We also keep in mind Obsidian's environment – ensuring the solution plays nicely as a plugin (for instance, using Obsidian's theming for UI chrome like buttons or using its API to respond to note selection). With the **"do both" approach confirmed**, we can mention both web and native frameworks, but given our context, the focus remains on web-based solutions that run inside Obsidian.

3D Graph Network Visualization Components and Design

Focus on a 3D "Outer Space" Graph: The core of our plugin will be a **3D graph network visualization** – an interactive node-link diagram rendered in three dimensions. The user described it vividly: "it's 3D, you can navigate like you're in outer space… The nodes look like stars." This suggests a design where nodes are points of light (perhaps glowing spheres or dots) scattered in a dark space, and connections are lines linking them, somewhat like constellations. Achieving this requires a few key components:

- **3D Rendering Engine:** As mentioned, a library like Three.js is ideal to create a 3D scene with camera controls. We'll set up an *interactive canvas* where the user can zoom, pan, and rotate the view (likely using an orbit-controls utility so the user can orbit around the center, zoom in/out, etc.). This gives the feel of floating through a starfield of notes. Using WebGL via Three.js ensures we get smooth graphics acceleration for potentially hundreds of nodes. It also allows fancy visual polish – e.g. depth-of-field, particle effects, or shader effects – if we want to really make it feel "sideral" (outer space-like).

- **Nodes and Edges Representation:** Each **node** (representing a note or concept) can be drawn as a small 3D object. A simple approach is a colored sphere or point. We might make important nodes larger or brighter. In fact, as we incorporate network theory, we could size nodes by their significance (e.g. higher centrality = bigger node) [11]. Edges (connections) would be thin lines or tubes connecting nodes. We'll likely use a **force-directed layout** to position nodes: this algorithm treats edges like springs pulling related nodes together and non-linked nodes pushing apart,

naturally clustering the network. Libraries like D3's force simulation or the physics engine in vis.js/Sigma can handle this. For a 3D force-directed layout, we will compute positions in three dimensions instead of two – some existing projects or plugins use 3D force layouts (the InfraNodus plugin uses "an advanced force-layout algorithm [that] creates a 3D graph layout" [12]). The goal is to have clusters of interrelated notes forming distinct groups in space, with less-connected clusters floating further apart – much like galaxies or constellations separated by voids.

- **Interactivity & UI Controls:** The graph isn't just a static image; the user should be able to **navigate and interact**. Basic interactions include: *zooming and panning/rotating the view*, *clicking or tapping a node to highlight it* (and perhaps see the note's title, maybe a preview or open it in Obsidian), *hovering to show tooltips* (like the note name or some stats), and *dragging nodes*. Dragging a node could allow manual repositioning or just provide a satisfying way to tug the network – many graph tools let you pull one node and see the network adjust like a spring network. We also might include UI widgets like filters or search. For example, a **search box** to find a particular node and focus/zoom it. Or filters to show subsets of the graph (perhaps by tag, folder, or other metadata) – Obsidian's default graph view allows filtering by tags or path, and the research literature notes that filtering is essential once graphs get complex [13]. We could incorporate an on-screen slider or checkboxes to toggle certain nodes/groups. These peripheral UI elements can be built with standard HTML/CSS (or a small React component, etc.) within the plugin pane, overlaying the 3D canvas.

- **Performance Considerations:** A polished implementation must handle the potentially **large scale** of a user's vault graph. Dozens of notes are easy, but some users have thousands of notes with tens of thousands of links. Rendering that many nodes in 3D is challenging. We will lean on **WebGL** for its performance – thousands of objects can be handled if we use techniques like buffering (e.g. draw all nodes as a single geometry if possible) and level-of-detail reductions (maybe very distant nodes could be merged or not drawn). The physics layout for so many nodes can also be heavy; a solution is to do initial layout computations offline or gradually, and perhaps use clustering to reduce complexity. Academic studies on network visualization **scalability** show that beyond a certain graph size, a raw node-link diagram becomes overwhelming – both computationally and cognitively. One study found that people had "significant difficulty" understanding paths in node-link diagrams once graphs exceeded about **50 nodes** if heavily connected, or **100 nodes** even if relatively sparse [14]. This suggests that for very large graphs, we need features like clustering, filtering, or summarization to keep the visualization useful [13]. Our design can account for this by, for example, collapsing less-important nodes or only rendering a focused subset at a time (with an option to expand a region on demand).

- **Native UI Components (Tables, Modals, etc.):** The user specifically asked if we should consider general UI components like tables, modals, forms – likely for the plugin's non-graph interface. While the *primary* UI is the graph view, we shouldn't ignore standard elements entirely. We may have a **settings modal** (for options like "toggle 3D/2D mode, adjust physics strength, enable community detection coloring", etc.). This would use Obsidian's native dialog or a custom modal. If we display analytical data (say, a list of top influential nodes or clusters), that could be shown in a side panel or table. For instance, after computing metrics, we might list the highest centrality nodes in a small ranked list UI, or pop up a modal comparing clusters. These conventional UI pieces can be built using Obsidian's styles (to match the app's look) – possibly with the help of a UI library or just vanilla HTML. In summary, **graph-specific components come first** (3D canvas, network interactions), but we'll integrate them with supporting UI (dialogs, sidebars, forms) to provide controls and detailed

information. By focusing on the graph visualization as the centerpiece, we ensure our efforts go into what makes this plugin unique, while still using familiar UI elements for settings and supplementary info.

Comparison with Industry-Leading Graph Visualization Tools

To design a top-notch solution, it's wise to learn from existing **industry-leading applications** in the realm of networked thought and data visualization. We'll consider both tools in the *personal knowledge management (PKM)* space (like Obsidian itself and its competitors) and more general network visualization software.

Obsidian's Built-in Graph View: Obsidian's default graph is a 2D force-directed network of all notes, which many users love for its eye-candy appeal. However, its utility is limited. As Nodus Labs points out, the native graph *"visualizes connections"* but **provides no structural insight** – it doesn't convey which notes are most important, what clusters of ideas exist, or where the gaps are [15] [16]. It treats all notes equally and links only represent direct note links. You can see a big hairball of notes (which is fun to watch as your vault grows), and identify maybe the most linked notes (they appear central), but that's about as far as the insight goes [17]. The user can manually color notes by tag or folder, but this is tedious and still doesn't reveal deeper relationships [18]. Essentially, Obsidian's graph is more decorative than analytical. Our plugin aims to fix exactly these shortcomings, much like the InfraNodus project did. (We'll discuss InfraNodus in a moment, as it's a prime example of an advanced Obsidian graph plugin.)

Roam Research and Others: Roam Research is another PKM tool known for its *networked note-taking*. Roam also offers a **graph overview** of all your pages. Implementation-wise, Roam's graph is a force-directed web canvas (users have observed it's likely done with D3 or similar on an HTML canvas) – it allows zooming, dragging, clicking nodes, very similar to Obsidian's default graph. However, Roam's graph also tends to become unwieldy with many nodes, and it doesn't add special analytics; it's mainly a visual map of links. Other note-taking apps like **Logseq** and **Craft** have introduced graph views too, but generally these follow the same pattern: a basic network of notes, sometimes limited to showing local relations. A noteworthy mention is **TheBrain**, which is an industry veteran in visual knowledge bases. TheBrain uses a center-and-periphery visualization: when you focus on a note (a "thought"), it brings that to the center and shows directly linked notes around it, allowing you to navigate by moving the center. This is a bit different from a full graph overview; it's more of a contextual neighborhood view that helps traversal. TheBrain's interface is highly optimized for quick browsing of connections, and it's very polished in terms of visuals (smooth animations, etc.). However, it's not 3D – it's a 2D network that re-centers. For inspiration, TheBrain shows how focusing on one node and exploring outward can be useful, as opposed to always showing the entire vault at once. We might incorporate a similar concept (e.g. the ability to double-click a node to "focus" or isolate its neighborhood in the 3D view).

Gephi and Network Analysis Tools: Outside of PKM, tools like **Gephi** (an open-source network analysis software) and **Neo4j Bloom** (graph database visualization) show what's possible with heavy-duty network visualization. Gephi is not interactive in the same real-time way (it's more for static layouts, and it's 2D), but it offers a plethora of layout algorithms and metrics. For instance, Gephi can calculate centralities, find communities, and then you can visually encode those (size, color of nodes). **Neo4j Bloom** and similar graph database front-ends allow users to query and explore data in a graph, often with a slick UI to expand neighbors and apply layouts. The common thread is that **industry tools combine visualization with analytics** – they don't just throw nodes randomly; they use algorithms to help users make sense of the

network. This is a lesson we will apply: incorporate network science metrics and perhaps AI to highlight what's interesting in the graph, not just display every node equally.

InfraNodus 3D Graph Plugin (Obsidian): Among community solutions, *InfraNodus* by Nodus Labs stands out as a direct analogue to what we're attempting. InfraNodus is essentially a text network analysis tool that they adapted into an Obsidian plugin. It provides a **3D graph view with network science insights**, addressing the very limitations we noted in Obsidian's default. The plugin ranks notes/pages by their **relative influence** using the betweenness centrality measure (so important nodes appear larger or highlighted) [12] . It also runs a **modularity algorithm** to detect clusters (communities of notes) and uses a force-directed 3D layout that pushes those communities apart, making clusters visually distinct [19] . This results in a spatial grouping where you can clearly see clusters of ideas that belong together, and you can even quantify distances between clusters to identify *structural gaps* in your knowledge graph [19] . Those gaps are essentially areas where there are few connections – which could hint at topics you haven't linked (possible new insights if you connect them). InfraNodus even integrates GPT-4 to suggest ideas bridging those gaps [20] . Another key feature: it shows **connections between any notes that appear in the same context**, not just direct links [21] . For example, if note A links to B and A links to C in the same paragraph, the default graph would show A–B and A–C links, but *not* B–C (since B and C aren't directly linked). InfraNodus does show a B–C link because B and C appeared together contextually [21] . This gives a richer graph where secondary relationships are visible, avoiding the "star-shaped" networks that Obsidian's default view often produces [22] . All these enhancements (centrality, cluster separation, contextual links, AI integration) make InfraNodus arguably the *state-of-the-art graph view* for Obsidian right now. For our project, we will definitely compare our approach with InfraNodus. We want to be inspired by it but also think beyond – not necessarily restricting to their exact feature set, as the user said *"we are not restricting to do as they do."* That means while we will **learn from these tools, we'll consider novel or theory-driven features** that set our plugin apart.

In conclusion, the landscape of existing tools teaches us that a plain graph visualization can be cool but not very informative. The **industry leaders enhance visualization with analysis** (be it network metrics, filtering, or novel UI paradigms). To be polished and refined, our plugin should do the same: not just show a pretty 3D web of notes, but also guide the user to insights (important nodes, clusters, unseen connections) in an intuitive way. We will incorporate these lessons and also ground our design in proven theory, as discussed next.

Theoretical Foundations and Best Practices for Network Visualization

Instead of just mimicking features blindly, we aim to ground our design in **academic theory and principles** from fields like information visualization, cognitive science, and network theory. This "meta theory" will ensure our plugin isn't just flashy, but truly helps users think better with their data. Here are the key theoretical insights guiding us:

- **Cognitive Alignment of Graphs:** Research and cognitive theory suggest that representing knowledge as a network aligns well with how the human brain organizes information. Rather than linear lists or hierarchies, our minds form associative networks of concepts. As one author put it, *"Nodes in graphs represent thoughts more naturally by mirroring how our brains process and store information."* Each node stands for an idea and edges show associations, which *"aligns with the*

human cognitive mapping process, where we naturally link related thoughts to form a coherent understanding." [23] [24] . In practical terms, this means a graph view can aid **recall and creativity** because it externalizes the web of connections our brain might otherwise form internally. Traditional note-taking (pages of text or isolated files) doesn't visualize these links, potentially missing out on this cognitive boost. Embracing this theory, our graph visualization should strive to make relationships explicit and easy to explore, helping users "think in networks" which can enhance learning and idea generation.

- **Knowledge Graphs vs. Linear Notes:** Building on the above, there's a known critique of purely linear note-taking – it fails to capture interconnections. When notes are linked in a network (and shown visually), it can improve retention and understanding by enabling non-linear traversal of ideas [25] [26] . Modern PKM tools like *Heptabase* and *Scrintal* (which the user might be aware of) leverage a spatial graph-like approach: they give you an infinite canvas to lay out ideas and draw connections, very much like a mind-map or graph. The benefit of such **visual, node-based note-taking** is a more **organic flow of thoughts**. As one commentary notes, these tools *"provide an infinite canvas where ideas can grow organically, unlike the restrictive nature of document-based methods."* They cater to "modern thinkers" who need flexible, non-linear ways to organize thoughts [27] . This concept validates our pursuit of a graph-based UI in Obsidian – it's not just a gimmick, but has theoretical support as a better medium for certain kinds of thinking. Our plugin, by adding a 3D canvas of notes, taps into this advantage.

- **Information Visualization Best Practices:** The field of InfoVis offers some guiding principles for any visual design. One classic is Shneiderman's mantra: *"Overview first, zoom and filter, then details-on-demand."* In our context, this suggests the graph should provide an **overview of the whole knowledge base** (the big picture of all notes and their connections), while also allowing the user to zoom in on areas of interest, filter out clutter, and get details (like note content) when needed. We've already considered features like filtering (e.g. hide less relevant nodes, or filter by tag) which align with this mantra. Academic studies highlight that without such filtering or aggregation, large graphs turn into incomprehensible *"hairballs"*, where everything overlaps [13] . By supporting multiple levels of detail (whole vault vs focused cluster vs single note details), we adhere to proven design wisdom that makes complex visuals usable. We also plan to use interaction techniques like highlighting neighbors of a selected node, or isolating a subgraph, which are forms of *focus+context* interaction – showing a part of the graph in detail while still keeping a sense of the larger context.

- **Graph Layout Algorithms and Clarity:** From graph drawing research, we know that the choice of layout and how information is encoded visually has huge impact on readability. A **force-directed layout** is popular for its intuitive clustering, but it can suffer from randomness or local minima (different runs give different arrangements, and sometimes overlapping happens). We might explore improved layouts or constraints – for example, ensuring clusters (communities) are separated (InfraNodus did this by repelling communities from each other in 3D space [19]). Also, 3D vs 2D: There's debate in visualization literature about whether 3D helps or hinders understanding of graphs. 3D adds a dimension for separation (potentially reducing overlaps if done well) and can be more engaging (the "wow" factor of flying through data). However, 3D can also introduce occlusion (nodes hiding behind others) and depth perception issues. One exploratory study on 2D vs 3D network views in VR found that for small networks, 3D or "2.5D" views were liked, but for larger networks, users sometimes preferred 2D due to clarity [28] . We will keep this in mind: our 3D view should be implemented with cues to depth (maybe using node size or fog to convey distance) and

possibly an option to flatten to 2D if a user gets lost in space. The ability to rotate the scene should mitigate occlusion (you can turn the graph to see behind clusters). We will need to test the usability, but theoretically, if done carefully, 3D can separate clusters along a third axis and reduce visual clutter, making complex graphs easier to parse. It certainly adds to the "polish" and delight factor, which is a goal as well.

- **Network Analytics – Centrality and Communities:** A key theoretical aspect is the application of **network science metrics** to enhance understanding. Academic network analysis introduces measures like **centrality** (e.g. betweenness centrality to find bridging nodes, or degree centrality for popularity) and algorithms like **community detection** (modularity clustering, etc.). By integrating these, we turn a raw graph into a richer informational graphic. As mentioned, InfraNodus used betweenness centrality to rank ideas by "relative influence" [12] – this is based on theory that nodes connecting different parts of the network are critical (in knowledge terms, such a node might represent an idea that serves as a bridge between topics). Highlighting or enlarging high-centrality notes helps the user spot key concepts in their vault. Community detection is rooted in graph theory; it finds groups of nodes that are densely connected internally. In a knowledge graph, these could correspond to themes or clusters of related notes. By coloring nodes by their community (as InfraNodus does [11]), we give a visual cue of which notes belong together. The theory of "structural holes" (from sociology/network theory) says that gaps between communities are potential areas of innovation – in other words, if you can connect two clusters, you might create a novel idea. This directly informed InfraNodus's feature of identifying gaps between clusters [20], and it's something we can leverage as well. We might, for example, implement a feature to suggest "You have Topic A cluster and Topic B cluster that are far apart – is there a concept linking them?" Possibly integrating with AI or at least prompting the user to consider linking those. All of this is meta-theory guiding features that **empower the user to not only visualize but analyze their knowledge network**.

- **User Cognitive Load and Graph Size:** Another theoretical angle is cognitive load management. We touched on this with the 50/100 node rule of thumb from the study [14]. The takeaway is that **human working memory can easily get overwhelmed by too many nodes/links at once**. Thus, a polished tool should include ways to reduce cognitive load – e.g., by limiting how many nodes are shown (maybe default to showing the neighborhood of a currently selected note rather than everything), or by using visual abstraction (combine multiple less-important nodes into a summarized node). There's active research on making network visualization scalable by clustering or by switching to different representations (like matrices) for dense portions [29]. For our scope, we likely stay with node-link diagrams, but we will employ interaction to mitigate overload (filtering, as discussed, and on-demand detail). It's also worth considering performance tuning not just for FPS but for user mental comfort: perhaps introduce a dynamic detail level – e.g., when zoomed out, we hide individual node labels or very minor nodes so the screen isn't full of tiny dots, and as you zoom in they appear. This way the *overview* is clean (just major structures) and *details* emerge when focusing, which is again a practical application of Shneiderman's mantra guided by cognitive limits.

In summary, our research-driven approach means we're designing the plugin with a blend of **cognitive science (how users think and remember)** and **network visualization principles (how to effectively display complex relationships)**. We are taking inspiration from academic findings to ensure the plugin not only looks impressive but genuinely improves the user's ability to understand their knowledge network. By aligning with how the brain naturally maps ideas and by utilizing proven visualization techniques (overview+detail, highlighting important nodes, clustering), we aim to create a tool that *"infuses your second*

brain with astounding new insights," to borrow a phrase from TheBrain's marketing [30]. In practice, this means a user of our plugin should be able to see the **constellation of their thoughts** and glean patterns or ideas that were non-obvious from the raw notes alone.

Conclusion

Bringing it all together, developing this Obsidian graph visualization plugin involves **multiple layers of deep research**: understanding the technical frameworks available, designing the 3D network UI components, learning from existing tools, and grounding decisions in theory. We've concluded that a web-based approach (leveraging libraries like Three.js, Sigma.js, etc.) is appropriate, given Obsidian's environment, and we can incorporate both custom rendering and use of native UI elements for support. Our focus remains on the **3D graph network visualization**, treating notes as stars in a navigable space – with attention to interactive polish and performance. By examining current solutions (Obsidian's default, Roam's graph, TheBrain, and the advanced InfraNodus plugin), we have a clear picture of what's lacking and what's state-of-the-art. This plugin will differentiate itself by going beyond eye-candy: it will employ **network analysis and academic best practices** to make the visualization insightful. Theoretical insights into cognition and visualization guide us to create a tool that not only displays data but amplifies the user's thinking – helping to discover clusters of knowledge, key connector ideas, and gaps ripe for exploration. In essence, the project is not just about replicating a graph view, but about *reimagining* it with a solid foundation in both the art and science of data visualization. We aim to deliver a polished, refined 3D graph plugin for Obsidian that feels as immersive as "outer space" yet as informative as a research report – a true fusion of innovative UI and deep intellectual utility.

Sources:

- Muhammad Usman, "20 Must-Know JavaScript Libraries for Data Visualization," *Dev.to*, Feb 19, 2025 [4] [9].
- Nodus Labs, *"Obsidian 3D Graph View Plugin with Network Science Insights,"* August 22, 2024 [12] [11].
- *Yoghourdjian et al.*, "Scalability of Network Visualization from a Cognitive Load Perspective," *IEEE TVCG*, 2018 (study manuscript) [14] [13].
- Theo James, "Nodes in Graphs Represent Thoughts More Naturally," *Medium*, Sep 2, 2024 [23] [24].
- Nodus Labs, *InfraNodus Obsidian Graph Plugin description*, 2024 [31] [21].

1 4 5 7 8 9 20 Must-Know JavaScript Libraries for Data Visualization - DEV Community
https://dev.to/web_dev-usman/20-must-know-javascript-libraries-for-data-visualization-508d

2 3 GitHub - ophidian-lib/core: A Component Framework for Obsidian Plugins
https://github.com/ophidian-lib/core

6 Graph visualization library in JavaScript [closed] - Stack Overflow
https://stackoverflow.com/questions/7034/graph-visualization-library-in-javascript

10 11 12 15 16 17 18 19 20 21 22 31 Obsidian 3D Graph View Plugin with Network Science Insights – Nodus Labs: Ecological Thinking through Network Analysis
https://noduslabs.com/featured/obsidian-3d-graph-view-plugin-with-network-science-insights/

13 14 29 Scalability of Network Visualisation from a Cognitive Load Perspective
https://ialab.it.monash.edu/~dwyer/papers/scalablilityofnetworkvis.pdf

23 24 25 26 27 Nodes In Graphs Represent Thoughts More Naturally | by Theo James | Medium
https://medium.com/@theo-james/nodes-in-graphs-represent-thoughts-more-naturally-d044343d3849

28 2D, 2.5D, or 3D? An Exploratory Study on Multilayer Network Visualisations in Virtual Reality
https://research-portal.uu.nl/ws/files/245925661/2D_2.5D_or_3D_An_Exploratory_Study_on_Multilayer_Network_Visualisations_in_Virtual_Reality.pdf

30 TheBrain: The Ultimate Digital Memory
https://www.thebrain.com/