3-Dimensional Graph Visualization of Mathematical Knowledge

Richard Marcus[0000−0002−6601−6457], Michael Kohlhase[0000−0002−9859−6337], and Florian Rabe[0000−0003−3040−3655]

Computer Science, FAU Erlangen-Nürnberg

Abstract. We describe TGView3D, an interactive graph viewer optimized for exploring mathematical knowledge as 3D graphs. To exploit all three spatial dimensions, it extends the commonly-used force-directed layout algorithms with hierarchical components that are more suitable for the typical structure of mathematical knowledge. We use different interaction methods to cope with the flood of information and allow users to alternate between different visualization settings. Our system is accessible through web browsers and in the form of a desktop application with optional virtual reality support. TGView3D can communicate with OMDoc-based knowledge management tools in order to offer semantic, highly mathematics-specific interaction with the graphs.To demonstrate the flexibility of TGView3D, we present and discuss multiple case studies based on different kinds of graphs including large proof assistant libraries and computer science lecture notes.

1 Introduction

Digital libraries of both informal and formal mathematics have reached enormous sizes. For instance, at least half-a-dozen theorem prover libraries exceed 10^5 statements. Thus, it is getting more and more difficult to organize this knowledge in a way that humans can understand and therefore access it. While library sources, generated presentations, and IDEs – e.g. those of PIDE [Wen19] – give good access to local knowledge structures, global properties of the induced knowledge spaces are very difficult to assess.

This can lead to the duplication of effort because users are unaware of them or cannot find previous work. Often systems employ modularization features to introduce a high-level structure that can, in principle, help to navigate through the libraries. Many of these features can be captured in the language of theories and theory morphisms as used in, e.g., the OMDoc/MMT format [Koh06; RK13], which has been used as a uniform representation standard for libraries related to theorem proving, computation, and mathematical documents. To help users understand this high-level structure better, we have explored visualizing theory graphs in the interactive 2D graph viewer TGView [RKM17].

But the sheer size of theory graphs — they can easily contain thousands of nodes and an order of magnitude more edges — not only complicates the process of computing structured graph layouts, but also human comprehension.

Beyond, these graphs often have **multidimensional** structures, i.e., they are multi-graphs with complex edge relations, possibly including different types of edges, that are difficult to map to two dimensions, even in smaller graphs. For example, we have never been satisfied with the visualization and user interaction features that state-of-the-art tools could provide for our own graph of logic formalizations [Cod+11] with (only) a few hundred nodes because its many edges representing logic translations are a challenge to standard layouting algorithms. By now, we have identified 3D visualization as a possible solution in order to use the third spatial dimension to better organize the graph [Bra14]. Extending this idea, we have experimented with virtual reality (VR) devices to allow a wide field of view and intuitive ways of interacting with the 3D world.

Related Work With the correct techniques, 3D graphs proved to be a viable alternative to 2D graphs [WM08], especially in in combination with virtual reality [Kwo+16] have been explored for use in theorem provers [Liu+17]. Generally, force-directed graph drawing [FR91] is the most popular type of layout algorithm for organizing large networks. It introduces forces so that nodes repel each other but at the same time attract connected ones, generating a layout that forms groups and reduces the edge lengths. However, these do not offer special treatment for directed edges, which are key features of many mathematically relevant graphs. In the theory graphs, in particular, the inheritance hierarchy is a directed acyclic graph (DAG) that is a central, cognitive aspect.

To better visualize graphs like this, layered graph drawing [STT81] usually is the method of choice. This is a 2D algorithm that first places the nodes on a minimal number of layers so that the edge direction is consistent and then tries to reduce edge crossings by reordering the nodes within their layers. But that makes it difficult to incorporate a second type of edges that are undirected or lead to cycles. The latter in particular is a central feature of using theory morphisms which routinely lead to cycles, e.g., when showing two theories to be isomorphic, or use multiple different theory morphisms between the same nodes. There are some approaches that incorporate hierarchies into force-based layouts such as [DKM06], albeit only in 2D settings.

To visually separate parts within a graph, it is possible to compute graph clusters [Sch07] based on different criterions. In contrast to that, TGView3D uses existing information about node affiliations; for example, we use the folder and document structure of source files to cluster into related areas that users are already familiar with.

There are dedicated tools for interactive 3D graph visualization such as **Gephi** [BHJ09] or web applications (e.g., based on WebGL) like [3DGV]. Alternatively (and this is our choice), we can use general purpose systems, in particular game engines like Unity. These have the advantage of allowing more flexible programming environments, cross-platform support and rapid prototyping. Notably, Unity allows both a WebGL version that we can embed into browser-based interfaces for casual users like our MathHub portal as well as executables for power users, which offer better performance and VR integration.

Contribution We contribute an adaption of traditional force-directed layout algorithms so that they can represent hierarchies of mathematical knowledge in 3D. To complement this, we propose interaction and clustering concepts to further support efficient exploration of the respective libraries. Finally, we have developed a 3D graph viewer, TGView3D, based on the **Unity** game engine [UGE], that implements both of these, offers interoperation with MathHub and supports VR interactions.

The system is licensed under GPLv3 and is available at `https://github.com/UniFormal/TGView3D`. A web application can be found at `https://tgview3d.mathhub.info` and a virtual reality demo video can be found at `https://youtube.com/watch?v=Mx7HSWD5dwg`. A preliminary demonstration was given at CICM 2018 as a demo-only presentation, i.e., without any accompanying write-up.

Overview Section 2 recaps the underlying knowledge representation framework including theory graphs. Section 3 describes our layout approach and Section 4 the user interaction capabilities of the system. Section 5 presents case studies and applications, and Section 6 concludes the paper.

Acknowledgments The authors gratefully acknowledge financial support from the OpenDreamKit Horizon 2020 project (#676541) and the DFG (project OAF; KO 2428/13-1, RA-18723-1). Furthermore, we are grateful for hardware support from and very helpful discussions about layout algorithms with Jonas Müller, Roberto Grosso, and Marc Stamminger.

2 Preliminaries

MathHub `https://MathHub.info` stores and manages mathematical archives and libraries across systems and is based on the MMT system. For TGView3D, it serves as the main source of theory graph interactions: on the one side, users can find links that open the respective theory graph in TGView3D; on the other side, TGView3D can link back to specific theories within the MathHub archives.

Theory Graphs are mechanisms for structuring large (formal and informal) knowledge collections used in many mathematical systems. They consist of graphs of **theories** connected by various kinds of **theory morphisms**. In OMDoc/MMT [Koh06; RK13] – the knowledge representation format we presuppose in this paper – Theories consist of groups of **declarations** that describe the respective theory and theory morphisms are truth-preserving mappings between theories.

The most important theory morphisms in OMDoc/MMT are **inclusions** for the inheritance hierarchy and **views** that express translations, interpretations, and representation theorems.

In theory graphs, inclusions are the most prevalent type of edges and induce a directed acyclic subgraph, which is important for understanding the primary structure and thus needs to be prioritized in the layout. Views may introduce cycles or connect very distant theories. There are also other kinds of morphisms

in OMDoc/MMT, but from the point of view of graph layouting, they fall into one these two categories. Additionally, both the nodes and the edges use MMT URIs for identification.

Running Example: The LATIN Graph The LATIN project [Cod+11] develops a modular representation of the logical languages of mathematical software systems as a meta-level theory graph. In LATIN, logics are theories and theory morphisms are logic inclusions, translations, relativizations, and even models. The LATIN logic graph has about 1000 theories and views; it can be found at `https://gl.mathhub.info/MMT/LATIN`.

3 Visualization and Layouting

3D Graph Visualization Before computing a graph layout, we need to define how the graph should be rendered. Since we often cannot see the end of edges in 3D, it would not be optimal to mark the direction of edges with a single arrow head. Therefore, we indicate direction with a gradient from light to dark, which also allows the user to estimate the length of the edge by analyzing how fast the color fades.

For theory graphs, we color include green, views blue and edges beginning from MMT namespaces red. We represent theories as spheres and the namespaces as cubes. We can further visualize the physical hierarchy in the graph by assigning the same color to nodes of the same cluster and adding a large, underlined label so that users can locate clusters by name and target them from afar. Additionally, we only render labels that are big enough to be readable, which also prevents the labels from occluding the graph.

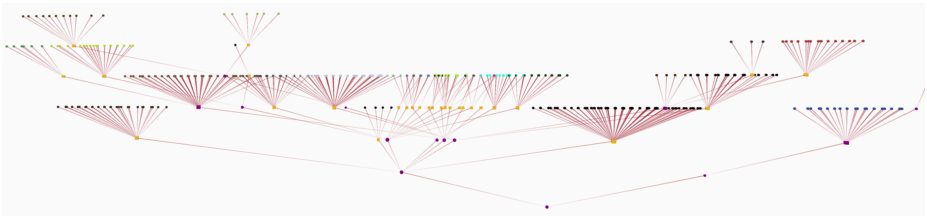


Fig. 1: LATIN namespace structure with static Layers

3D Layouting A key benefit of the 3D approach is that can utilize the third spatial dimensions to devise layout algorithms that cater to the structure of mathematical knowledge as, e.g., in theory graphs. The main goal here is to have a layout algorithm that, above all, can present the graph hierarchy without sacrificing the advantages of force-directed layout algorithms, i.e, that nodes can form groups and use the available space effectively.

Static layers in the style of layered graph drawing (cf. Figure 1) are problematic in combination with force-based approaches, as this limits nodes from

forming groups horizontally can lead to inefficient use of space, depending on the node/layer–distribution, e.g. all nodes could be forced into a single layer. Depending on the use case, though, this still is a valid solution in 3D as there are still two spatial dimensions left to be used by the force-directed algorithm. This applies to tree hierarchies like the hierarchic namespaces in our running example.

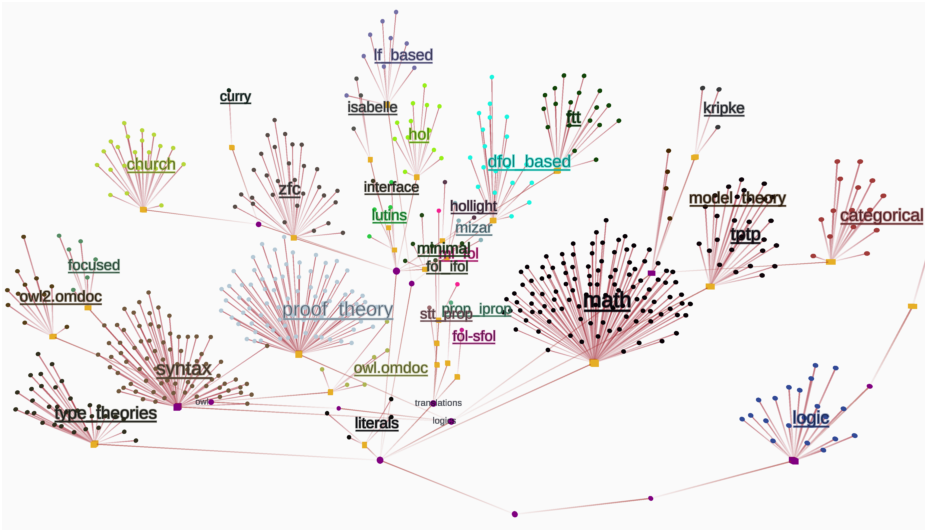


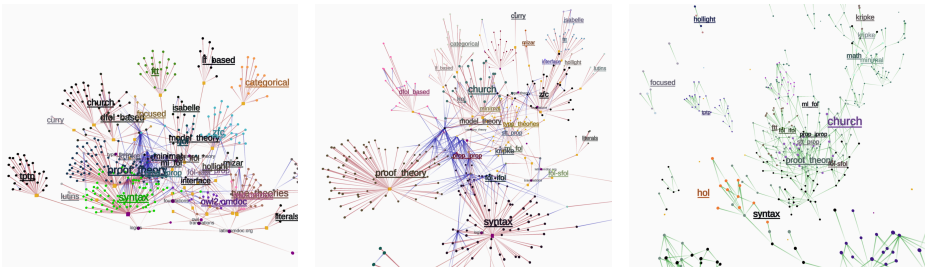
Fig. 2: LATIN namespace structure with hierarchic forces

DAGs and trees also allow a less restrictive **relative hierarchy**, i.e., a consistent edge direction going from bottom to top, which is also much better suited for combination with force-directed layout algorithms. Accordingly, we add a force that repels connected nodes either downwards or upwards, depending on whether they are successors or predecessors. This is already manipulates the layout sufficiently in many cases but we also added a way to explicitly guarantee that no successor/predecessor-pair is ordered incorrectly, independently from the number of iterations: The first step is to position the nodes so that the ordering is conform with the relative hierarchy — placing all nodes on the same layer, for example, is a valid initialization. Then, we restrict the node movement during each iteration so that successor and predecessor only "overtake" each other along the correct direction. The drawback is that this may affect the convergence of the layout negatively, so it is a matter of preference and concrete use case, whether this constraint should be active or not.

All in all, we achieve our goals to preserve the hierarchy while allowing the force-directed algorithm to work relatively freely and can also achieve good results in 2D if the structure is not too complex (cf. Figure 2). In particular, this also allows us to process non-hierarchical edges by simply ignoring them during hierarchical force computation. This typically results in the following or-

ganization: at the bottom of the graph, we have a tree structure according to namespaces or the physical file structure. On top of it, the DAG expands, with remaining relations affecting the horizontal ordering of the nodes by trying to achieve short edges.

4 Interaction



(a) Dense View Structure (b) View Structure in 3D (c) Inclusion DAG in 3D

Fig. 3: LATIN Graph: Interactive Layout Customization

The main benefit of 3D layouts is interactivity; screenshots or static images of 3D layouts naturally appear cluttered as the part in the front occlude more distant parts. By rotating the graph or changing the camera position, a single computed 3D layout can present an infinite number of 2D projections. For example, if we add the view edges to the namespace hierarchy of the LATIN graph (cf. Figure 3a), the layout seems cluttered and we can only distinguish the edges by color, looking at the graph from another perspective (cf. Figure 3b), however, changes this and we can recognize how the view edges occupy the inner part of the graph and connect different theories. In general, this is especially useful when the spatial dimensions also have different semantic meanings as in mathematical libraries, so that the front perspective rather highlights the hierarchy whereas the top perspective shows groups or other between-node relations.

Graph Exploration However, theory graphs such as the LATIN graph contain multiple different types of edges. Showing them all at once can result in confusing layouts, so we need further methods to support the user. We provide these within the TGView3D UI (cf. Figure 4). For advanced users like developers or library maintainers, we offer further features to edit the graphs like adding and removing nodes and edges via context menu, which we also demonstrate in Figure 4. In the top bar, we offer options to choose the graph to visualize and in the left bar options for configuring the visualization and other features.

In fact, we already have shown a very effective exploration method in Figure 3, namely filtering edges by type. This has two uses: first, simply hiding not required edges. Different types of relations in theory graphs have very specific

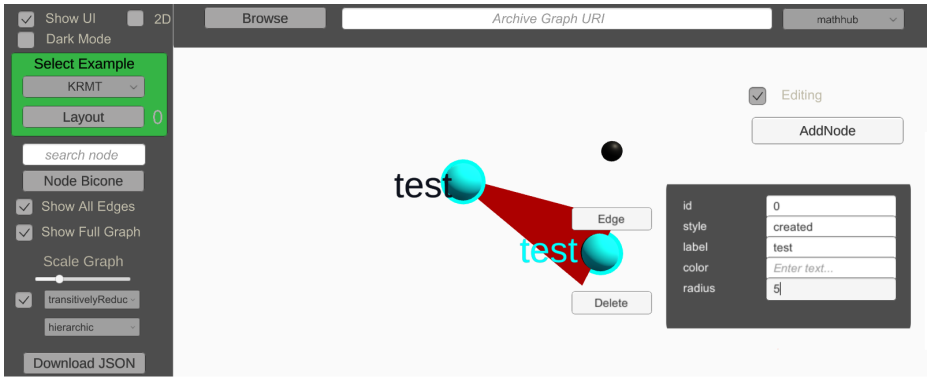
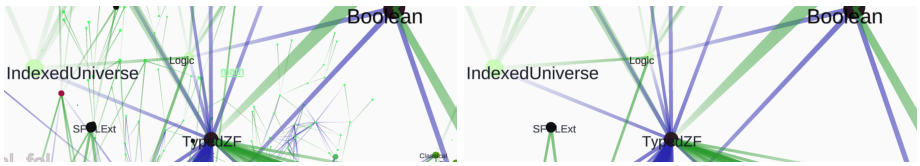


Fig. 4: TGView3D User Interface

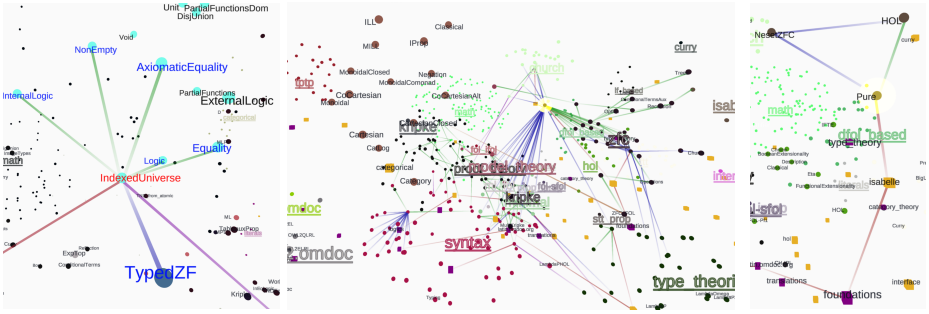
meanings and users often only want to work with one type of edges at a time, e.g., the file structure of the LATIN graph (cf. Figure 1 to navigate to a certain file, or its include hierarchy (cf. Figure 3c) to track inheritance relations. Second, it can be of interest, how certain combinations of edge types affect the layout. The user can then start from a certain layout, e.g. the file structure (cf. Figure 2), request another edge type like views and watch in realtime how the layout algorithm reorganizes the structure to achieve short edges — effectively moving related namespaces close to each other.



(a) Distant parts cause occlusion (b) Clear view after hiding distant parts

Fig. 5: LATIN Graph: Local Exploration of include/view structure

Alongside this way of analyzing global graph structures, it is often even more important to explore local structures in the graph as these actually represent the direct relations between theories. TGView3D offers several methods: Firstly, the user can **scale** the graph: this changes the node spacing and thus breaks up densely connected node bundles or, the other way around, makes it easier recognize them as such. In the case of our running example, the user may have identified the dense view structure in Figure 3a as area of interest and now decides to analyze it. Therefore, he enables the include hierarchy and zooms into the graph until he reaches the nodes in Figure 5a. To prevent distraction by the background, the user can hide distant parts of the graph (cf. Figure 5a) and then move through the graph in "slices". To obatin more information about the current nodes, the user can select a node, this highlights its direct neighborhood



(a) Neighborhood (b) Big Bicone of TypedZF (c) Small Bicone

Fig. 6: LATIN Graph: Subgraph Exploration

(cf. Figure 6a), even when the respective edge types are set as hidden. Based on this interaction, the user can now crawl through the graph and perform a custom selection. To come full circle and return to the global structure analysis, it is also possible to visualize the hierarchy position of a selected node within the graph. We call this the **node bicone**, as it hides all edges of nodes that do not belong to either the reachable subgraph or the dependency subgraph of the node — creating one dependency cone upwards one downwards. In particular, this also indicates whether a node is very central to the graph (cf. Figure 6b) or not (cf. Figure 6c).

Hierarchical Clustering Many graphs contains information about the source or the physical location of a node. This introduces a physical hierarchy by way of the division into packages, folders, files, etc. that is orthogonal to the logical hierarchy induced by inclusion/inheritance.

The physical structure is most helpful to form clusters of nodes, which significantly improves graph exploration. We have already described this partly in Section 3 but we also collapse whole node cluster into a single bigger node to reduce graph complexity.

To preserve as much information as possible, we then perform **edge propagation**: we add the edges of the collapsed nodes to the new cluster node. Given that such clusters can also contain further clusters, the user can explore the hierarchy by expanding and collapsing the graph interactively. This is extremely important to effectively reduce the graph to a size where humans can recognize clear structures. In our running example, we can see the first hierarchy level of the LATIN as clusters in Figure 7, which compresses the graph effectively and can act as starting point for further exploration.

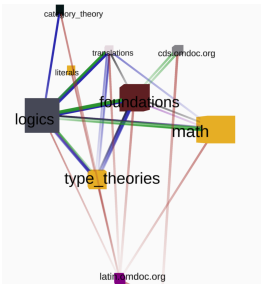


Fig. 7: LATIN Graph: Hierarchic Clustering

VR Interaction While humans usually are very accustomed to working with 2D content, these skills do not

necessarily transfer to 3D navigation with keyboard and
mouse, which may require a high number of simultaneous user inputs.

VR headsets, come with special input devices or even offer hand tracking and gesture support. Once the user is familiar with these, graph interactions can be implemented very intuitively. For example, the user can move the graph by performing push or pull gestures or literally grab nodes to select them. Exploring the 3D graph also becomes more efficient because of the stereoscopic nature of VR: the user basically has access to an infinite screen by simply looking around and actually perceives the depth of the 3D world.

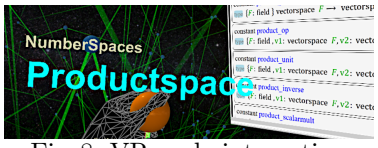


Fig. 8: VR node interaction

To access the multitude of filtering and exploration features, we attach a virtual, touchable UI to the right hand. Touching the labels opens a website in the browser that contains information about the respective node, which we then stream into the application as a virtual touch screen, shown in Figure 8.

Performance In interactive applications, performance is an important consideration. We have not conducted a systematic performance evaluation or an optimization of the code at this point. Therefore, we just provide some indicative timings in Table 1 (taken without clustering or other preprocessing steps on an AMD Ryzen 2600X with 100 iterations).

As we see, depending on the size of the graph, the initial layouting can become expensive; without multithreading, big graphs can take several minutes to load. In the next section we will show that this is only partly an issue as humans struggle to effectively process these big amounts of data. So we need information filtering methods that reduce graph sizes anyway.

#Nodes	23	223	739	3972
#Edges	52	637	2851	17769
Time [ms]	87	236	1635	39318

Table 1: Time measurements of base layout algorithm

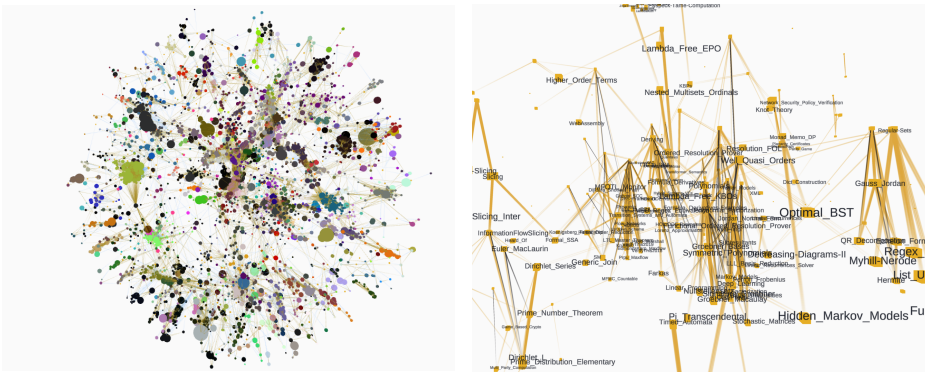
5 Case Studies

To test TGView3D in practice, we have experimented with multiple different data sets and extracted graphs from them. Concretely, we present four use cases here: formal libraries of proof assistant, narrative documents for lecture notes, and two domain-specific applications — attack graphs from argumentation frameworks [BGG18] and model pathway diagrams (MPDs) [Kop+18] from mathematical models of physical systems. Each of them profits from the core features of TGView3D but also introduces unique interactions.

5.1 Formal Libraries

Because of the vast amount of data in formal libraries such as the Coq libraries or the Isabelle AFP, it is usually impractical to show the full library as a static

graph. To cope with this, we make extensive use of hierarchical clustering. Especially graph visualizations of big theorem prover libraries can benefit from this because the sources are already carefully structured in this way. For example, a



(a) Clusters expanded once

(b) Clusters fully collapsed

Fig. 9: Isabelle AFP with clustering and edge propgatation (type ulo:uses)

top level setting as in Figure 9a can reveal insights about hubs or node groups and, starting from a single theory allows users to track dependencies or similar relations like in our running example, e.g. in Figure 6. In Figure 9, we can see how clustering allows us to handle huge libraries like Isabelle AFP: even though Figure 9a shows around 4000 nodes, it actually is already heavily clustered. Each node represents a cluster that may contain further clusters. The final clustering step (Figure 9b) reduces the enormous library to only a few hundred nodes, still preserving relations through edge propagation as described in Section 4.

Theory Graphs The main intended use of TGView3D is to visualize theory graphs. We have already analyzed such a theory graph in the form of the running example, so we will focus on the MMT integration here. The graphs are provided by the MMT system (which runs separately) and can be opened in TGView3D via HTTP. The MMT API further allows loading subgraphs, e.g., related to a project group, a single project or the MMT namespace. Hence, we do not need to load the entire archive and then extract the subgraph afterwards. This is used when accessing individual graphs from MathHub but also within TGView3D. For example, when nodes have edges that point "outside", i.e., that the target node is not part of the graph that is loaded currently, we can request to load in addition to or instead of the current one (cf. Figure 11). Note, however, that we do lose information as the outside pointing edges cannot fully replace the missing structure. For example, Figure 11a exhibits a strong connection between the namespaces church

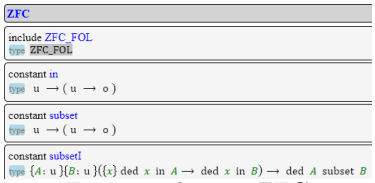
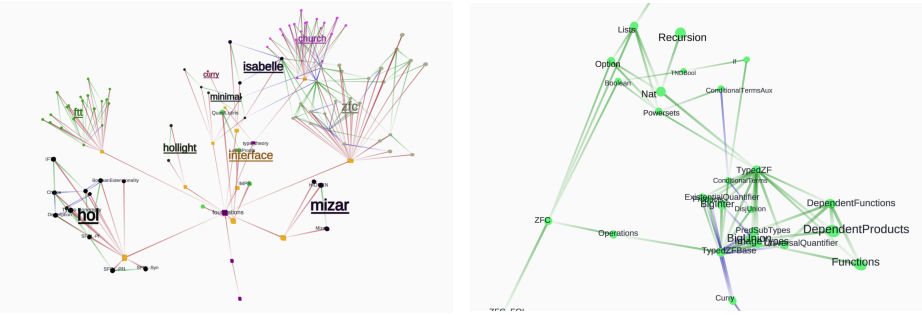


Fig. 10: Theory: ZFC

and zfc; even if add the outside pointing edges to Figure 11b, there will be no information regarding the internal structure of the church namespace within the isolated zfc subgraph. In turn, such an compact visualization allows the user to fully focus on the available structure which often can remain quite complex.

Finally, we can also use MMT URIs to access information about the node, e.g., what it represents and the underlying theory structure (cf. Figure 10).



(a) http://latin.omdoc.org/foundations (b) .../foundations/zfc

Fig. 11: LATIN Graph: MMT namespace subgraphs

RDF Graphs The **Upper Library Ontology** ULO[Con+19] specifies properties and relations for theorem prover libraries in general, including the library structure, semantic relations, cross-references, and metadata. This information is then the basis for different types of nodes and edges in TGView3D. Our system can visualize the ULO export of theorem prover libraries such as for Isabelle and Coq in [Con+19] and even mix the ULO information into the archive graphs presented above.

With TGView3D, we can now visually compare different libraries or try to get insights by exploring them. We can also use it to debug the representations of large libraries. For example, a small bug in the RDF generation led to a naming discrepancy that resulted in disconnected subgraphs and redundant nodes. This was quickly noticed in the visualization but would have been very difficult to detect otherwise, especially, considering that each of the 4000 nodes in Figure 9a can again contain up to hundreds of nodes an order of magnitude more edges.

5.2 Course Materials

A related, but different application of TGView3D for is **active course materials**: the second author has written all of his lecture materials (slides, course note, and background) in STEX, a semantic variant of LATEX, which can be transformed into OMDoc/MMT. The resulting documents mix document structures with aspects of OMDoc/MMT theory graphs and cross-link into other resources.

Note that any knowledge item covered in a course is subject to a triple context: *i*) the intrinsic **knowledge context** given by the theory graph of concept inheritance (includes) and interpretation (views), *ii*) the **document context** given by its position in the document structure (e.g. sectioning), and its relation to auxiliary materials, such as problems, master solutions, or clarifications/explanations on the course forum, and *iii*) the **course progress** (how far has the lecture already progressed).

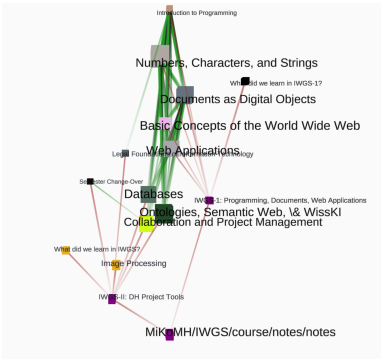


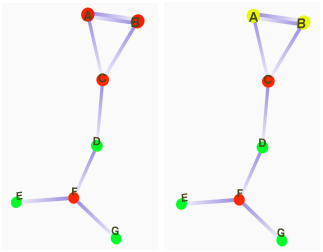
Fig. 12: IWGS lecture graph

This suggests exporting visualizations that allow students to navigate the joint context intuitively; Figure 12 shows an example that uses chapters for clustering. The graph has been partly expanded and contains around 600 nodes and 1200 edges. The knowledge context is theory-graph structured, so the discussion above applies. The course progress can be visualized by either hiding "future knowledge" or specifying clusters accordingly. Eventually, it will also be possible to jump from the graph to the respective chapters or contents within the document via URIs.

But there are also benefits for the document creator. On a technical level, he can easily catch incorrect relations inside the documents because the graph visualization is often broken in these cases, e.g. there are loops when there should be none or parts of the graph are fully disconnected. On a semantic level, the creator can obtain insights regarding how well the lecture is structured by analyzing how the hierarchy is presented in the graph visualization.

5.3 Argumentation Theory

An **argumentation framework** specifies classes of **attack graphs** consisting of **arguments** and **attack** relations between them; see [BGG18] for an overview. One of the interests of argumentation theory is whether an attack graph contains subgraphs containing only the arguments that should be accepted. These are called **extensions** under a given **semantic**. We can use TGView3D to visualize extensions of attack graphs by labeling (or coloring) nodes (and in some argumentation frameworks supporting/undercutting/...edges). The argumentation structures we show in this paper are formalized in OMDoc/MMT, so TGView3D is directly applicable. Figure 13 shows an example, where we use TGView3D to compare the grounded with the stable semantics. In the for-



(a) Grounded (b) Stable

Fig. 13: C attacks A and B, which attack each other

mer, A and B are rejected (red) and in the latter they are undecided (yellow).

(a) Attack graph: Roe v. Wade, (1973) (b) Reasoning with Tweety the penguin.

Fig. 14: Attack graphs with attacks and inclusions

Generally, the resulting graphs are rather small but they are a prime example for multidimensional graphs. Beside the attack edges, they also may have include edges to express logical reasoning (compare Figure 14b). Complex argumentation structures, in particular, often result in layouts with geometrical structures (Figure 14a).

Attack graphs profit from advanced interactions: modifying single edges or nodes affects whether arguments are accepted or not. While it is possible to perform these operations within TGView3D, we currently do not directly change the underlying OMDoc/MMT formalizations. Instead, the user can edit the OMDoc/MMT file accordingly and request the updated graph afterwards.

5.4 Model Pathway Diagrams

MPDs are a novel type of diagram for understanding mathematical models of physical systems and simulation algorithms that use these models. The nodes represent physical laws (cubes labeled with a MathML rendering of the main formula that constitutes the law) and the physical quantities (spheres); edges connect these laws with the quantities they involve. From the layout perspective, the key difference is that the topological structure of the graphs carries information about the possible simulation algorithms. Since such a careful – manual – layout helps understanding, modeling practitioners like to generate raw

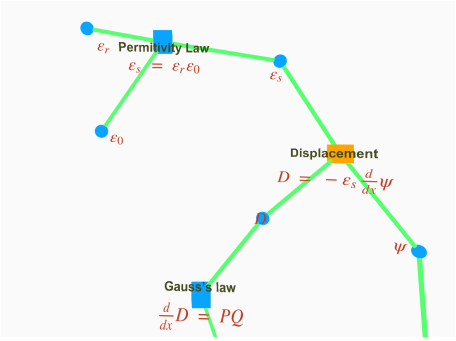


Fig. 15: A Model Pathway Diagram

layouts from OMDoc/MMT-encoded
models and then hand-tweak them for didactic clarity. Accordingly, we allow the user to reposition the nodes manually and store this layout within the graph file.

6 Conclusion and Future Work

We have presented an interactive 3D graph viewer that can handle hierarchical relations in multigraphs efficiently. Our layout algorithm makes use of the third spatial dimension to preserve the hierarchy and, simultaneously, optimize the node organization in a force-directed manner. Users can interact with theory graphs in 3D either traditionally via a screen and a mouse or via virtual reality devices. The proposed methods to explore the resulting structures globally and locally can break up the information density of large libraries.

Future work will focus reaching a wider audience. In addition to continuous improvements to the graph viewer itself, we want to create an ecosystem that simplifies the process of importing different kinds of graphs into TGView3D. TGView3D offers a variety of methods to visualize and explore different kinds of graphs. Extending this, we want to allow more customizability and offer preconfigured builds that are tailored towards domain-specific use cases. These can then also incorporate ways to interoperate more closely with the respective tools or libraries.

References

[3DGV] *3D force-directed graph component using ThreeJS/WebGL*. URL: https://github.com/vasturiano/3d-force-graph (visited on 03/11/2020).

[BGG18] P. Baroni, D. Gabbay, and M. Giacomin. *Handbook of Formal Argumentation*. College Publications, 2018. URL: https://books.google.de/books?id=_OnTswEACAAJ.

[BHJ09] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. "Gephi: an open source software for exploring and manipulating networks". In: *Third international AAAI conference on weblogs and social media*. 2009.

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

[Cod+11] Mihai Codescu et al. "Project Abstract: Logic Atlas and Integrator (LATIN)". In: *Intelligent Computer Mathematics*. Ed. by James Davenport et al. LNAI 6824. Springer Verlag, 2011, pp. 289–291. URL: https://kwarc.info/people/frabe/Research/CHKMR_latinabs_11.pdf.

[Con+19] Andrea Condoluci et al. "Relational Data Across Mathematical Libraries". In: *Intelligent Computer Mathematics (CICM) 2019*. Ed. by Cezary Kaliszyck et al. LNAI 11617. Springer, 2019, pp. 61–76. DOI: 10.1007/978-3-030-23250-4.

[DKM06] Tim Dwyer, Yehuda Koren, and Kim Marriott. "Drawing directed graphs using quadratic programming". In: *IEEE Transactions on Visualization and Computer Graphics* 12.4 (2006), pp. 536–548.

[FR91] Thomas M. J. Fruchterman and Edward M. Reingold. "Graph drawing by force-directed placement". In: *Software: Practice and Experience* 21.11 (1991), pp. 1129–1164. DOI: 10.1002/spe.4380211102. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.4380211102.

[Kal+19] Cezary Kaliszyck et al., eds. *Intelligent Computer Mathematics*. LNAI 11617. Springer, 2019. DOI: 10.1007/978-3-030-23250-4.

[Koh06] Michael Kohlhase. *OMDoc – An open markup format for mathematical documents [Version 1.2]*. LNAI 4180. Springer Verlag, Aug. 2006. URL: http://omdoc.org/pubs/omdoc1.2.pdf.

[Kop+18] Thomas Koprucki et al. "Model pathway diagrams for the representation of mathematical models". In: *Journal of Optical and Quantum Electronics* 50.2 (2018), p. 70. DOI: 10.1007/s11082-018-1321-7.

[Kwo+16] O. Kwon et al. "A Study of Layout, Rendering, and Interaction Methods for Immersive Graph Visualization". In: *IEEE Transactions on Visualization and Computer Graphics* 22.7 (2016), pp. 1802–1815. DOI: 10.1109/TVCG.2016.2520921.

[Liu+17] Jian Liu et al. "VMDV: A 3D visualization tool for modeling, demonstration, and verification". In: *2017 International Symposium on Theoretical Aspects of Software Engineering (TASE)* (2017), pp. 1–7.

[RK13] Florian Rabe and Michael Kohlhase. "A Scalable Module System". In: *Information & Computation* 0.230 (2013), pp. 1–54. URL: http://kwarc.info/frabe/Research/mmt.pdf.

[RKM17] Marcel Rupprecht, Michael Kohlhase, and Dennis Müller. "A Flexible, Interactive Theory-Graph Viewer". In: *MathUI 2017: The 12th Workshop on Mathematical User Interfaces*. Ed. by Andrea Kohlhase and Marco Pollanen. 2017. URL: http://kwarc.info/kohlhase/papers/mathui17-tgview.pdf.

[Sch07] Satu Elisa Schaeffer. "Graph clustering". In: *Computer science review* 1.1 (2007), pp. 27–64.

[STT81] K. Sugiyama, S. Tagawa, and M. Toda. "Methods for Visual Understanding of Hierarchical System Structures". In: *IEEE Transactions on Systems, Man, and Cybernetics* 11.2 (1981), pp. 109–125. DOI: 10.1109/TSMC.1981.4308636.

[UGE] *Unity Game Engine*. URL: https://unity3d.com (visited on 03/07/2019).

[Wen19] Makarius Wenzel. "Interaction with Formal Mathematical Documents in Isabelle/PIDE". In: *Intelligent Computer Mathematics (CICM) 2019*. Ed. by Cezary Kaliszyck et al. LNAI 11617. Springer, 2019, pp. 1–15. DOI: 10.1007/978-3-030-23250-4.

[WM08] Colin Ware and Peter Mitchell. "Visualizing graphs in three dimensions". In: *ACM Transactions on Applied Perception (TAP)* 5.1 (2008), pp. 1–15.

A Enlarged Versions of some of the Images

For convenience in a print-out, we re-iterate some of the images that are relatively small in the text. We hope to get an additional page in the final version, so that we can enlarge them there are well.

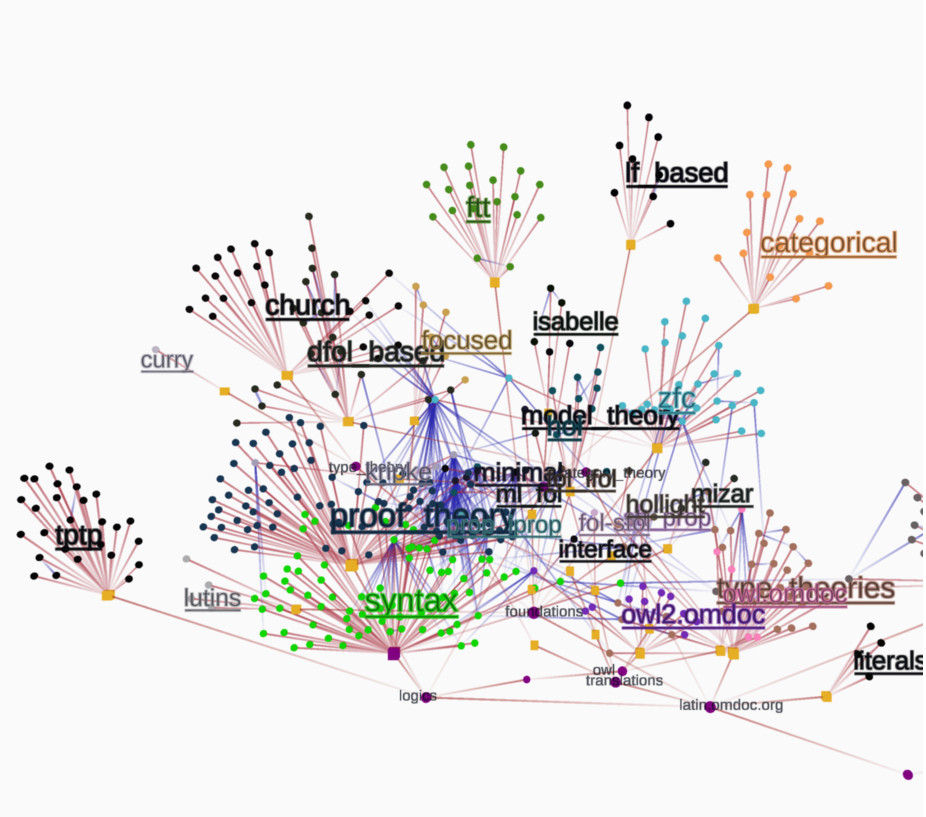


Fig. 16: Figure 3a "Dense View Structure"

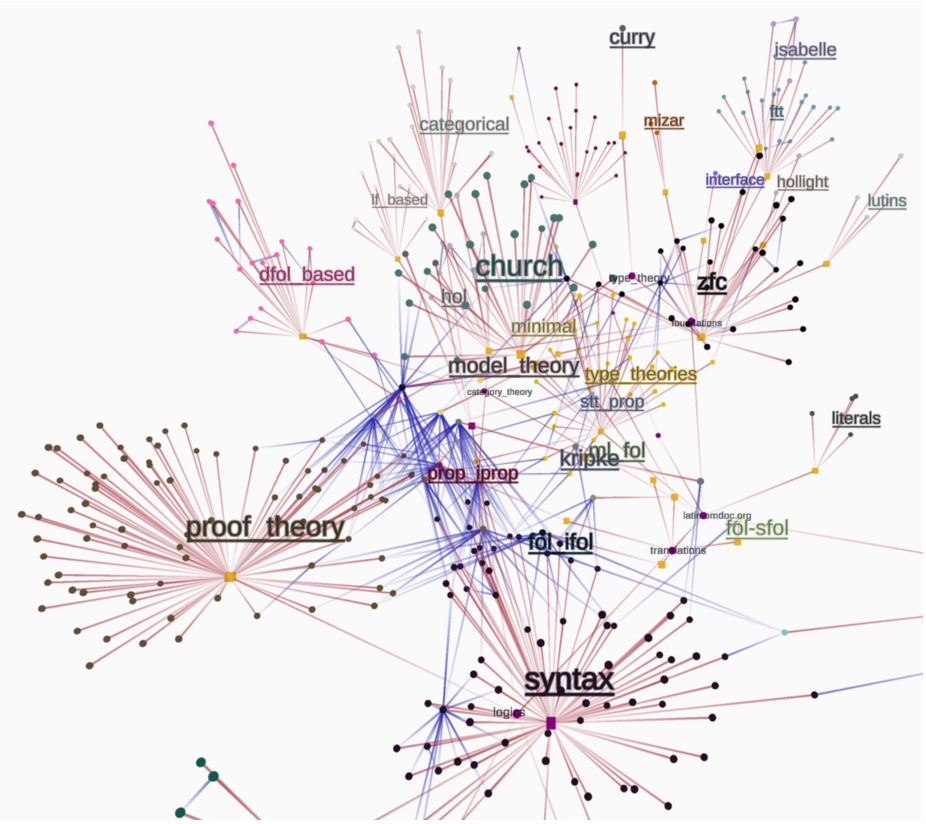


Fig. 17: Figure 3b "View Structure in 3D"

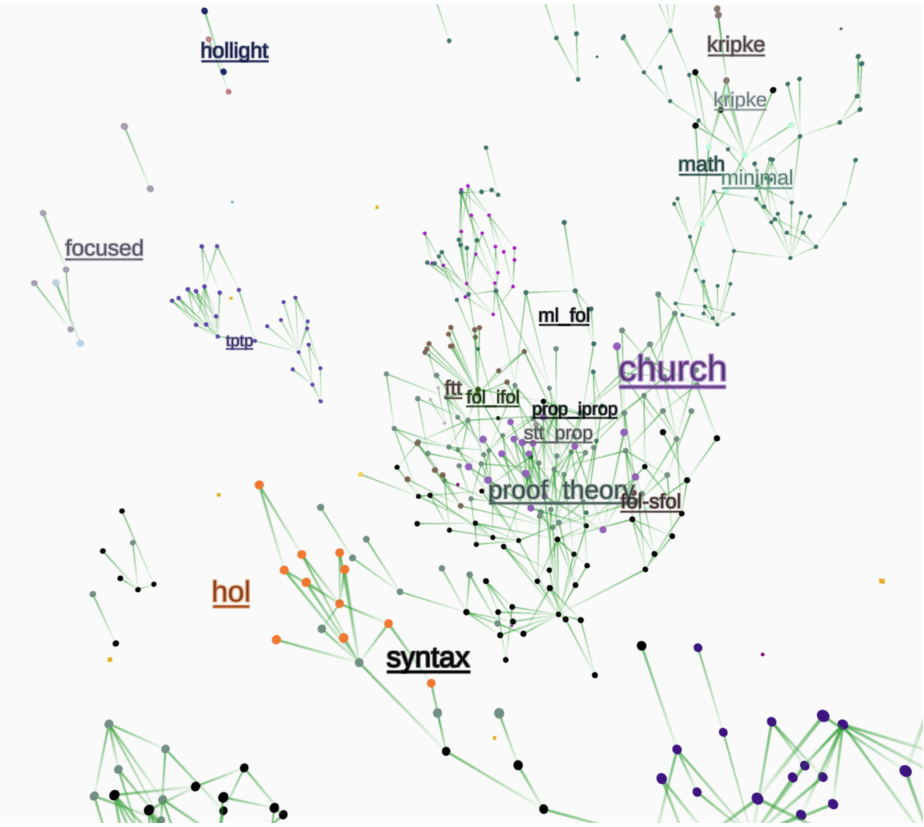


Fig. 18: Figure 3c "Inclusion DAG in 3D"

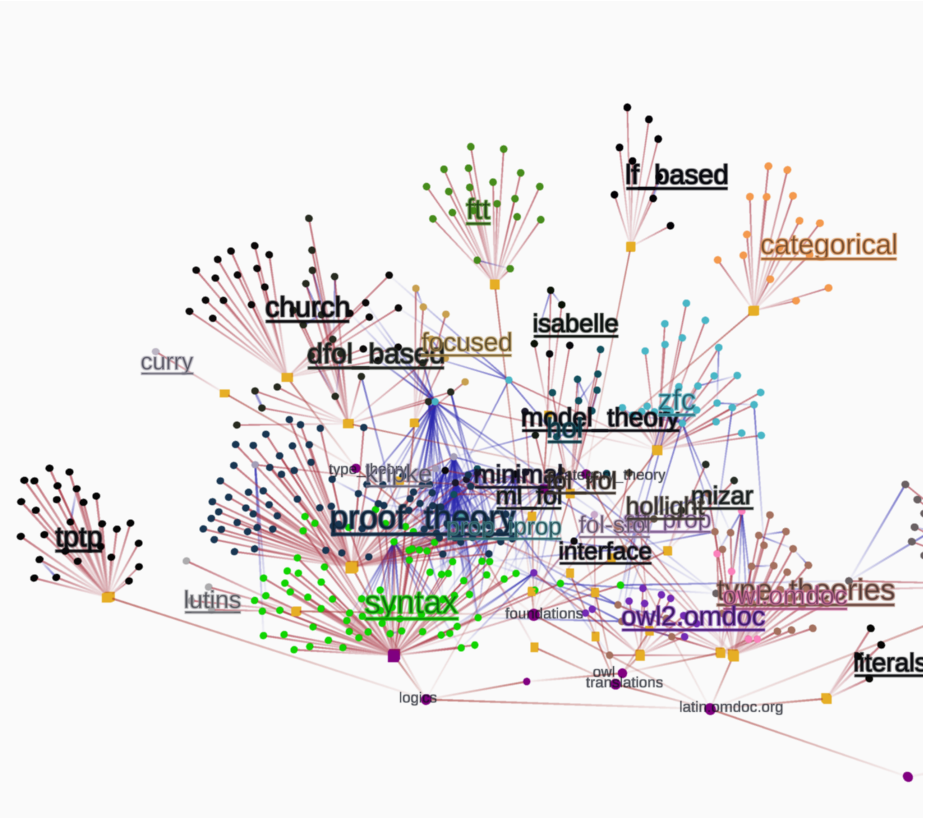


Fig. 19: Figure 3a "Dense View Structure"

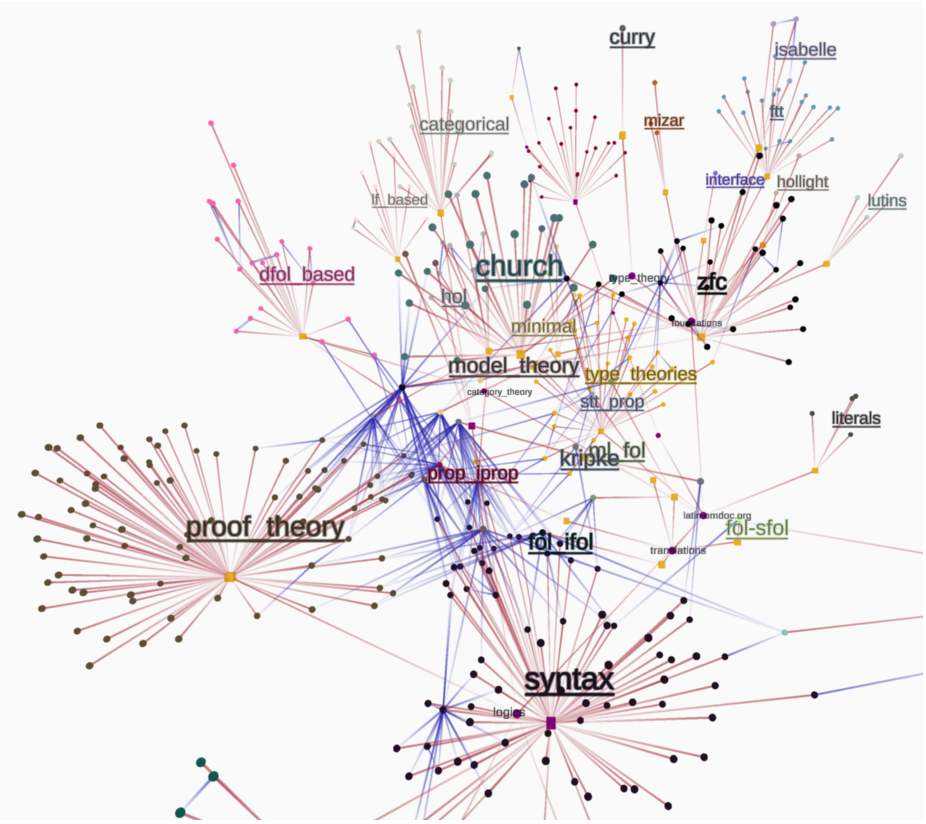


Fig. 20: Figure 3b "View Structure in 3D"

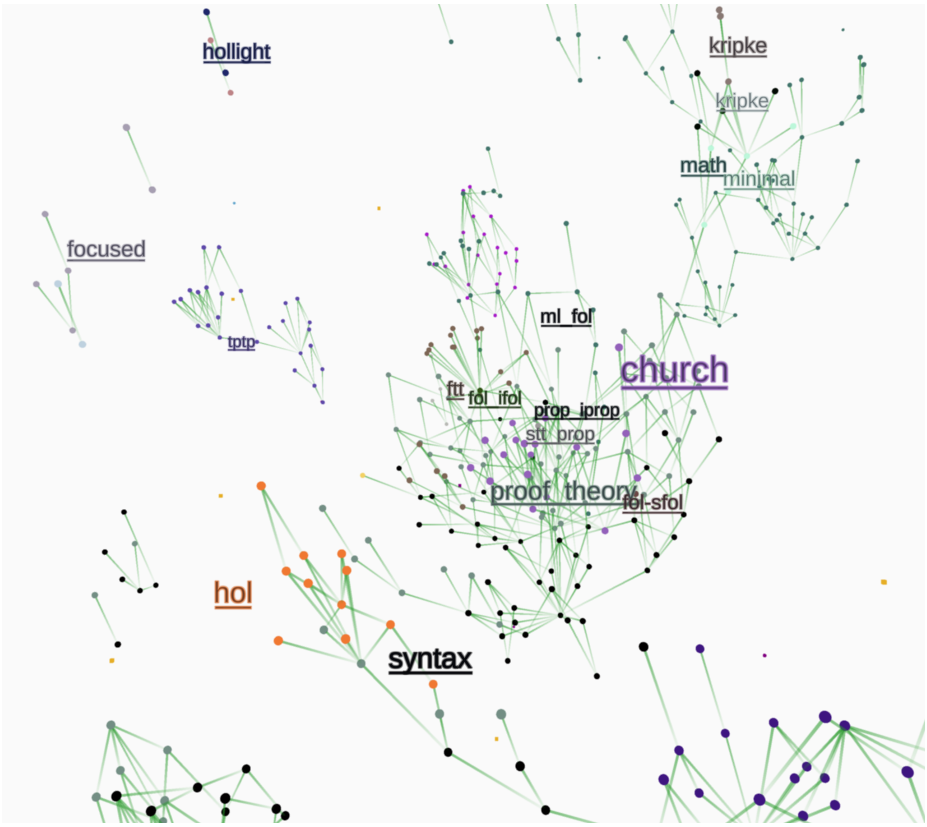


Fig. 21: Figure 3c "Inclusion DAG in 3D"

Fig. 22: Figure 8 "VR node interaction"

Fig. 23: Figure 9a “Clusters expanded once”

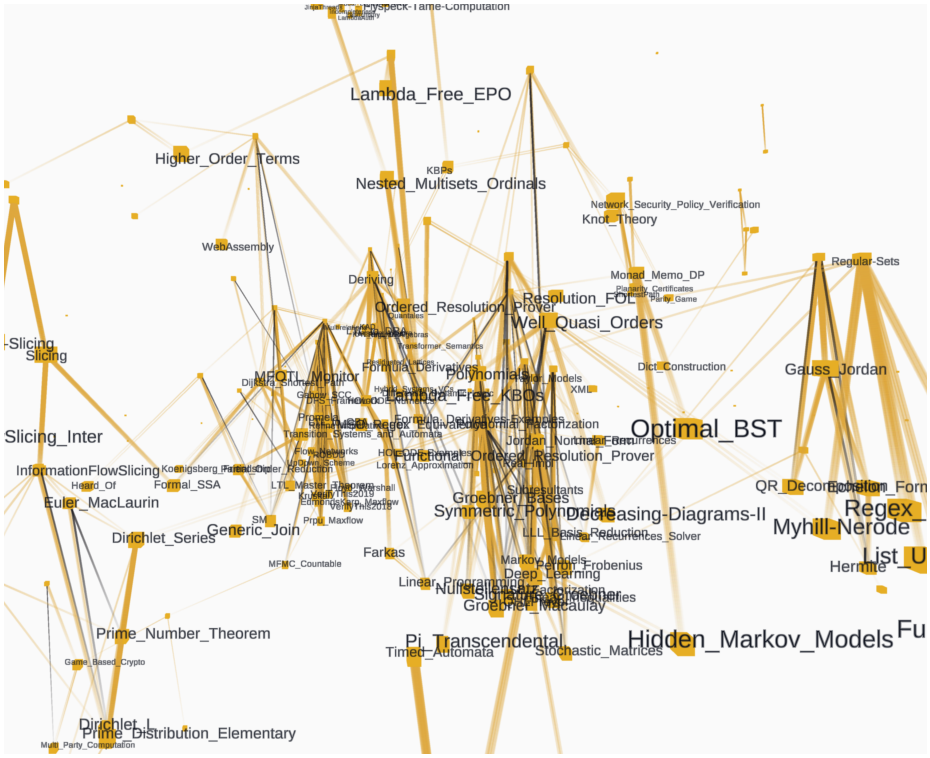


Fig. 24: Figure 9b “Clusters fully collapsed”

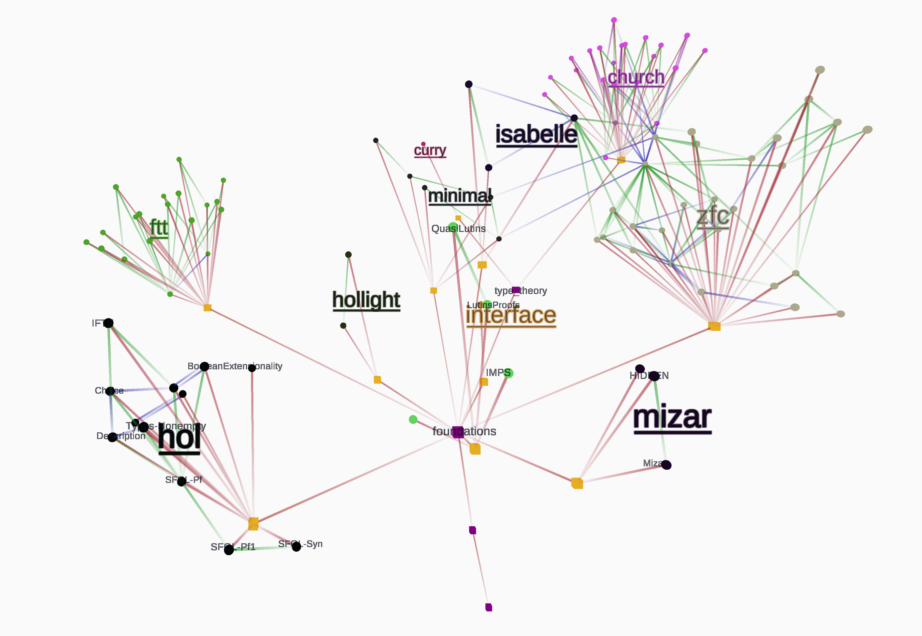


Fig. 25: Figure 11a "http://latin.omdoc.org/foundations"

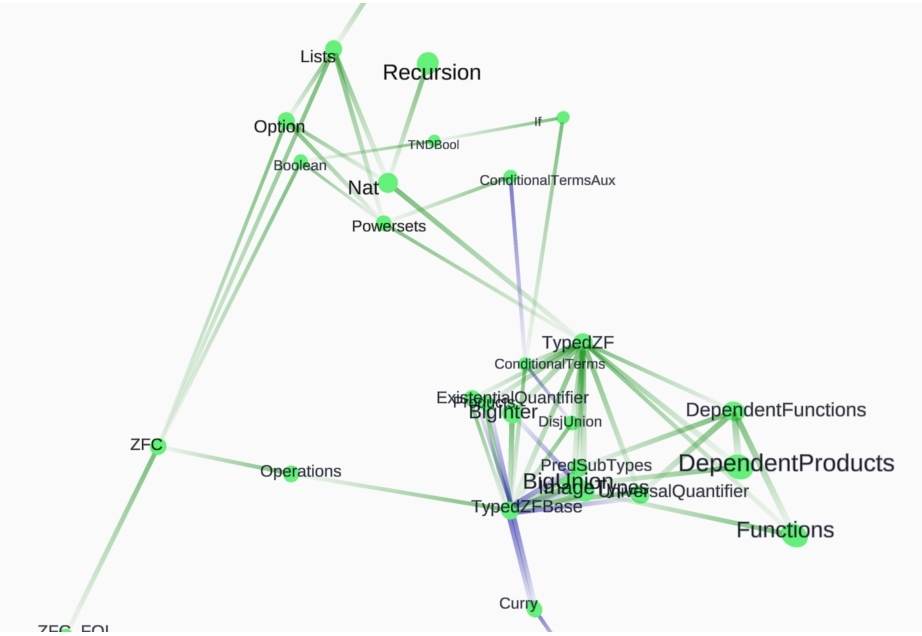


Fig. 26: Figure 11b ".../foundations/zfc"

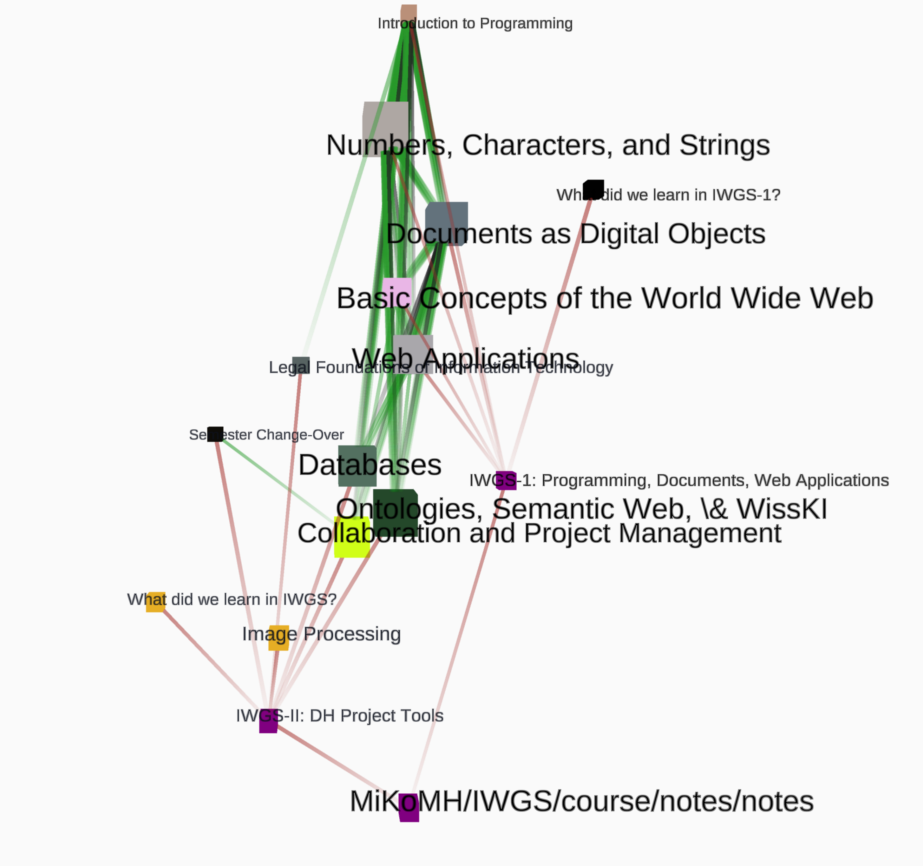
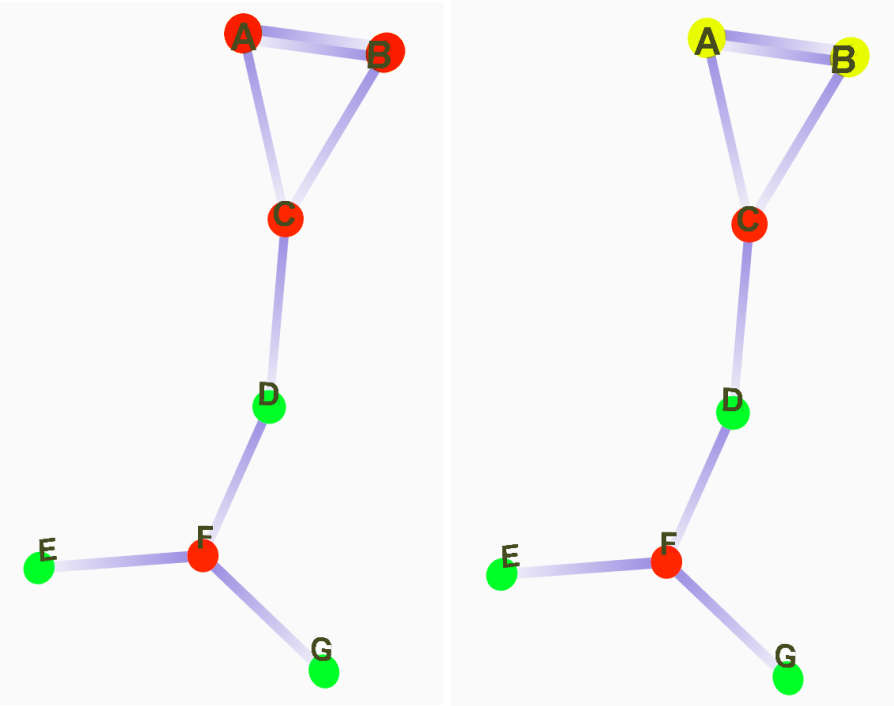


Fig. 27: Figure 12 "IWGS lecture graph"



(a) Grounded

(b) Stable

Fig. 28: Figure 13 "C attacks A and B, which attack each other"

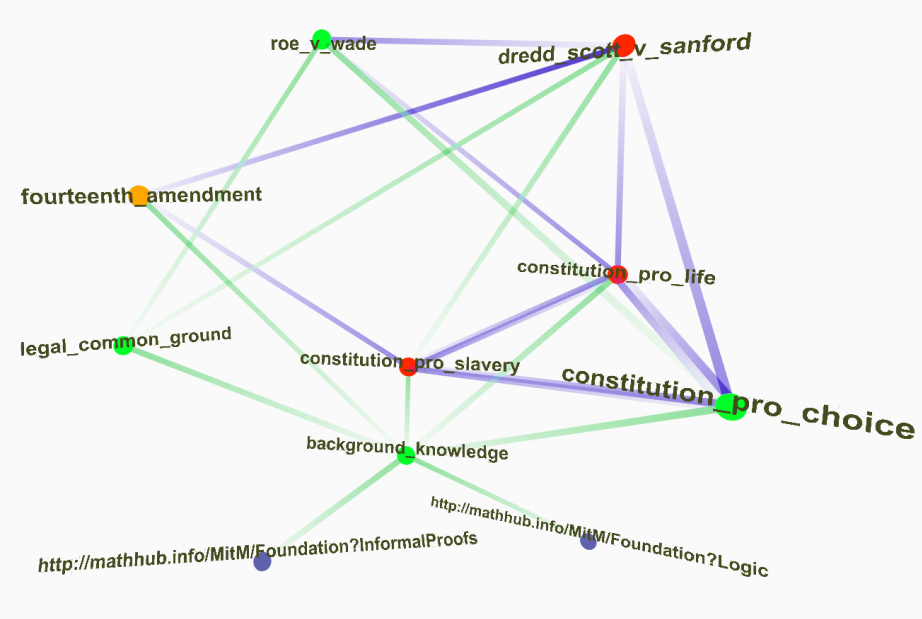


Fig. 29: Figure 14a: Attack graph: Roe v. Wade, (1973)

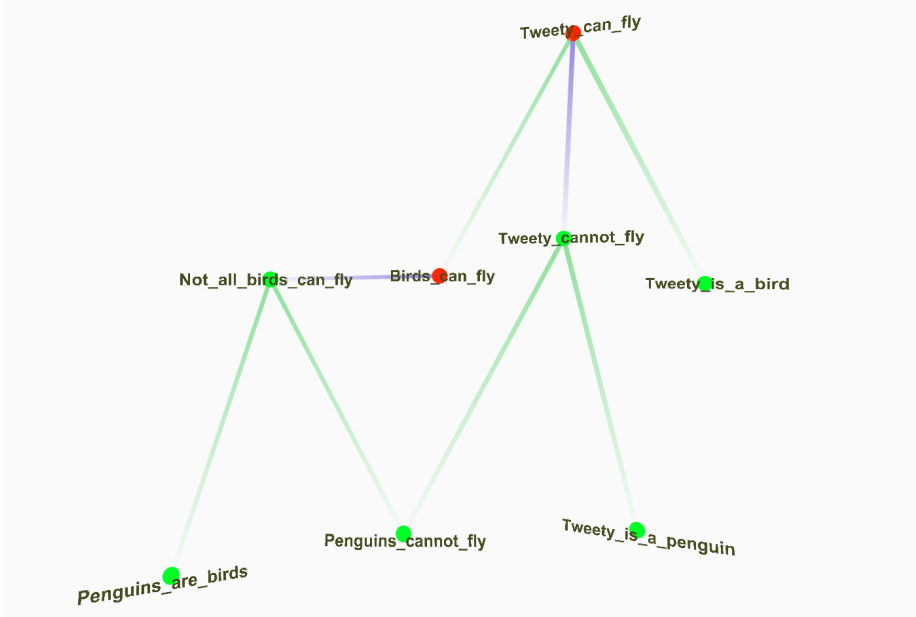


Fig. 30: Figure 14b "Reasoning with Tweety the penguin"

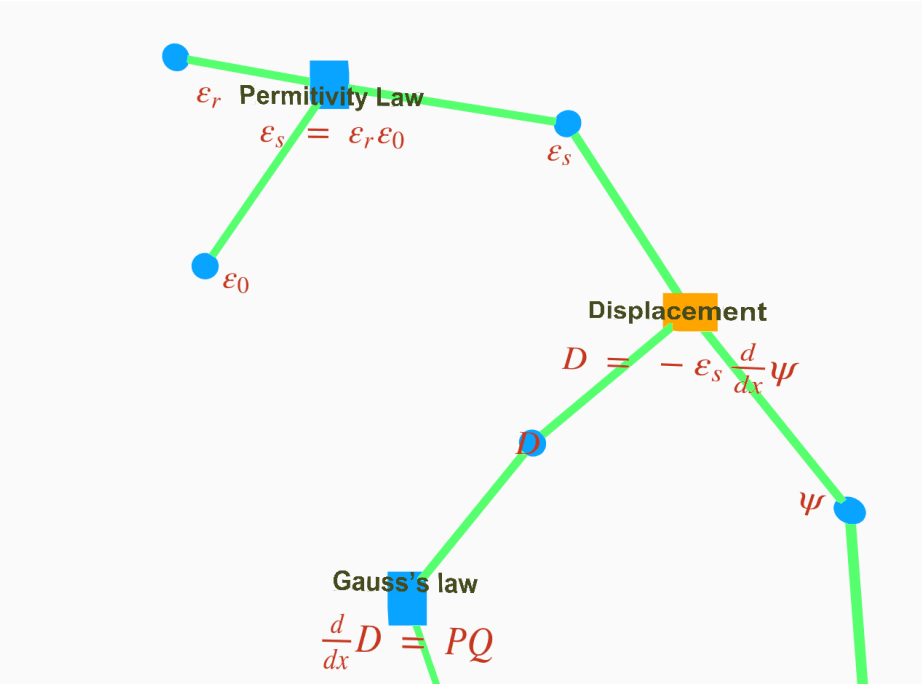


Fig. 31: Figure 15 "A Model Pathway Diagram"