A Case Study on Providing Immersive Visualization for Neuronal Network Data Using COTS Soft- and Hardware

Marcel Krüger*
RWTH Aachen University

Qin Li
RWTH Aachen University

Torsten W. Kuhlen
RWTH Aachen University

Tim Gerrits
RWTH Aachen University

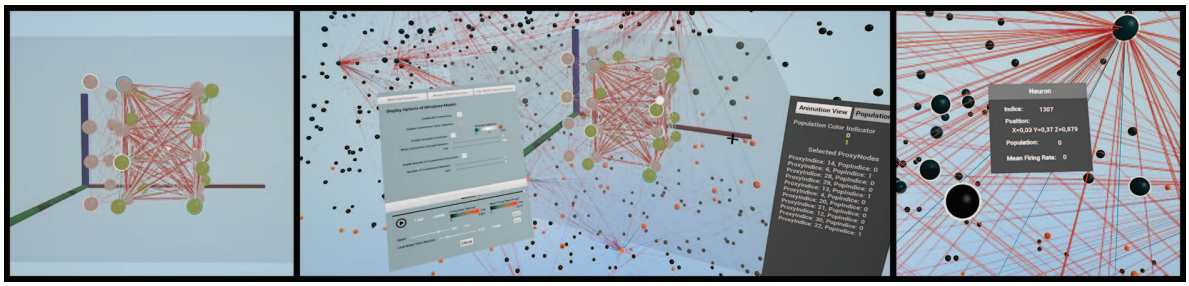

Figure 1: An immersive visualization software for the analysis of neuronal network data based on Unreal Engine. Left: A world-in-miniature model provides an overview visualization of neuronal activity. Center: Selection and filtering regions of interest change node renderings of a linked view. Right: Selection of single nodes in the main view shows node information.

Abstract

COTS VR hardware and modern game engines create the impression that bringing even complex data into VR has become easy. In this work, we investigate to what extent game engines can support the development of immersive visualization software with a case study. We discuss how the engine can support the development and where it falls short, e.g., failing to provide acceptable rendering performance for medium and large-sized data sets without using more sophisticated features.

Index Terms: Human-centered computing—Visualization—Visualization techniques—; Human-centered computing—Virtual reality—

Measuring and studying the functionality of neurons in the human brain is necessary to understand various phenomena and conditions of human development. This includes research on early intellectual development or mental disorders such as Alzheimer's and Parkinson's disease [7, 9, 14, 16]. Typical neural models contain more than 10^{10} neurons and 10^{14} synapses distributed across different brain regions. Given the complexity and sheer quantity of neurons, as well as the 3-dimensional location of the data, advanced analysis methods are required to reduce mental overload.

Immersive visualization approaches have proven to be a valuable tool for interacting with spatial data and allowing for engaging analysis. However, using virtual reality techniques requires low latency computation and interactive rendering frame rates while following suitable design patterns to avoid overwhelming the user [13]. Therefore, such applications have often been developed by specialized labs writing custom software that is fine-tuned to specific needs. This requires a lot of time and knowledge in several domains, ranging from efficient computations to rendering abstraction and interaction techniques. As affordable yet powerful consumer hardware has made developing and using virtual reality applications more readily available to researchers and consumers alike, the development of immersive visualization software — also for complex medical applications —- should also become more accessible. Game engines such as Unity [15] or Unreal Engine(UE) [3] use highly optimized code and processing strategies and further include ready-to-use techniques for interaction and GUI building. This allows users to create VR applications from scratch and sometimes even without any programming skills necessary.

In this work, we want to investigate the influence of game engines on two distinct questions:

1. Can domain experts create immersive applications without the help of immersive visualization labs?

2. Can immersive visualization labs gain an advantage from using game engines?

While these questions seem to tackle two independent challenges at first, they both share a common characteristic: A first indication for answers to both questions can be derived from investigating the ease of developing an application without deep knowledge or extensive research in immersive visualization. Therefore, we present a case study on how commercial-of-the-shelf (COTS) hard- and software can be used naïvely to offer immersive visualization of neuronal simulation. We develop an interactive, immersive visualization application built with Unreal Engine to analyze regions of interest in a brain neural network. We discuss how the game engine provides support and functionality for developing such software and where limitations and challenges arise.

1 Related Works

Visualization of neuronal networks is often limited to answering specific narrow research questions, which has led to a variety of tools. There exist a number of 2D abstractions, see, e.g., Keiriz et al, [8] representing connections as Hinton diagrams [11] or heatmaps to show the temporal changes within a network [13]. There also exist desktop applications that combine 2D abstractions and 3D renderings such as ViSimpl [4] or NEST Desktop [1]. The limitation to 2D display technology, however, fails to provide an intuitive understanding of neuron locations [10] and topological features, and does not scale well for large simulations. Therefore, immersive visualizations provide a promising approach in brain analysis [6] and 3-dimensional graph analysis [18]. Jaeger et al. [7] provide a useful overview of tasks and challenges for the immersive analysis of brain data and provide a prototype application based on Unity extended by

*e-mail: krueger@vis.rwth-aachen.de

979-8-3503-4839-2/23/$31.00 ©2023 IEEE
DOI 10.1109/VRW58643.2023.00050

custom R and D3 functions for processing and visualization. Most existing immersive visualization tools develop their own custom software [8,10,12,17] which is highly tailored toward their respective goals and require the integration of several aspects such as rendering, interaction, and data management. We investigate how off-the-shelf game engines can support the development of such software without any external dependencies or custom extensions.

2 Use Case

Our case study is set to represent a workflow for domain scientists who want to analyze computational neuroscience simulation data with commercial-of-the-shelf hard- and software. The data describes simulated neuronal activity taken from a NEST [5] simulation, which computes network activity on a Brunnel network. It contains 3126 neurons divided into two populations of inhibitory and excitatory neurons; they span 981876 neuron connections and generate 96396 spiking events over the simulation time. To give a spatial and temporal understanding of the data an application is required that allows users to observe the activity and structure of the network. Schneiderman's mantra was chosen to reduce the amount of data that is visible at the same time as this provides two advantages simultaneously: Not only does the "overview first, detail on-demand" approach provide an opportunity for performance improvements, but at the same time can help reduce the mental load in the analysis of the network.

The resulting application consists of three components: a miniature model, a main view, and a control menu as seen in Figure 1. The data in the miniature view represent the whole network's data at an abstract level within a fixed-size, movable, transparent cube to provide an overview of the network. Its size and position are based on the user's field of view to allow for suitable interaction. Groups of neurons with statistical equivalence, i.e., type and distance, are binned together to create subsets that can be displayed on demand in the main view. Each subset is represented by a scaled node at an averaged position as well as optional summarized connections, thus reducing the number of objects to render. The node color intensity indicates the activity level of the whole region, thickness and saturation of connecting lines indicate the number of connections and connection strength between regions, respectively. Connections can also be filtered based on a set of user-defined parameters via the user interface seen in Figure 3. By selecting a node in the miniature view, users select a subset of interest that is linked to the main view. The main view displays all neurons and connections in the selected subset(s) as well as the activity of individual neurons. The same visualization and filtering options can be applied as in the miniature model. Targeting neurons in this view activates an information panel providing more precise information on the neuron. The user can move into any region or observe the whole network from a distance. Movement is realized by right-hand-directed steering, allowing the user to observe the spatial relations while moving through the network. If larger distances need to be overcome quickly, a proxy representation of the user in the miniature model can be moved to a new position. Upon release, the user is teleported to the destination. The selection of neurons is provided in two modes: individual nodes can be selected in the main view and miniature view via standard ray casting. For group selections, a selection volume is shown in the miniature model. The position of the selection volume can be controlled through the controller's movement.

A work machine equipped with an Intel Core i9-10900X 10C20T@3.70GHz with 32 GB RAM and an NVIDIA RTX 3090 was used, representing strong but not high-performance hardware. As for the display technology, we decided to use the HTC Vive Pro 2 as a representative higher-end head-mounted display.

3 Discussion

The main goals of the case study were to determine the degree of expert knowledge needed when developing applications with COTS game engines and if such engines can provide a good base for immersive visualization labs. Game engines can potentially accelerate the development of new immersive visualization methods and applications. Especially in an academic context, people fluctuation in university research labs is a continuous issue. High fluctuation, e.g., due to students joining only for the period of a thesis, means that it is important to get them up and running quickly to stay efficient and successful as a research group. Using established COTS software increases the chances that potential candidates already bring experience and familiarity with them before joining the lab. Additionally, it allows existing researchers in the lab to refer to existing tutorials, documentation, and code to ease the entry. To choose a suitable game engine for the workflow, we compiled the following primary requirements:

- **Wide Adoption** A wide adoption addresses the aforementioned advantages of the availability of resources and the chance that new workforce already brings some experience with them. Additionally, it increases the chance of long-term support.
- **Large Feature Set** A large feature set speeds up the development of new applications by providing existing functionality that can be used to compose new methods.
- **Performance** Especially in the context of immersive visualizations, the application's performance is paramount. Real-world applications are often built with large and complex data, and immersive environments must always provide a smooth experience regarding interaction and rendering.
- **Accessibility** The entry barrier should be as low as possible to allow new developers to achieve results fast while not compromising on performance. This is especially important in an academic context as, e.g., students joining for a thesis have limited time to develop and evaluate new methods.
- **Extensibility/Adaptability** The ability to easily extend the engine allows development for special requirements and methods that benefit from an implementation inside the engine's code. Adaptability allows changes to the engine itself in case the provided implementation is not sufficient for a given task.
- **Flexibility** The engine should make it easy to develop applications for a wide range of systems. This is especially important in a fast-changing market like today's COTS hardware market. A flexible engine allows to easily keep up with the market's state-of-the-art and cooperation partners' requirements in terms of hardware.

Based on the above criteria, we decided to use Unreal Engine(UE) 4.27 for the case study as it fulfills the aforementioned requirements. UE is an established game engine with wide adoption. The commercial backing promises long-term support by the developers and good resources to get new users up to speed. A large community provides additional resources besides the official resources. Especially user-generated documentation and code can be beneficial to reduce the training period and the need to implement the required functionality. UE provides a large feature set that covers most needs for creating new applications. A big library provides a lot of basic functionality needed in immersive visualizations, such as efficient math operations, spatial queries, object intersections, etc. Furthermore, a lot of tooling is provided to make the development of applications easier, e.g., profiling and packaging binaries for distribution. UE is a highly optimized game engine written in C++. The capability of the engine

can be observed in tech demos, benchmarks, and when looking at existing applications and games. Thus, making it a good fit for immersive visualization applications that often have similar requirements for performance and interactivity. Besides the availability of resources, another benefit is that UE is geared toward a variety of developers. Not only are experienced C++ developers considered as users, but also users with less or no knowledge of C++ by providing ”blueprints”. Blueprints are a visual programming language integrated into UE and its editor that allows code-less programming. C++ code can be combined with blueprints by allowing developers to expose custom C++ code that is usable in blueprints. This makes them useful for newer developers and an excellent tool for experienced developers that want to do rapid prototyping. Thus, UE strikes a good balance between performance and accessibility. Due to the access to source model, developers have full source code access, which makes extensibility and adaptability easy for experienced developers. Functionality can be added to the engine core, and the engine's inner workings can be examined, allowing one to extend or adapt the engine to one's liking and needs, e.g., when special hardware should be supported or performance-critical methods should be embedded as low as possible. Lastly, UE provides flexibility using a Render Hardware Interface (RHI), which abstracts render commands from the underlying rendering backends. Rendering is abstracted such that even custom rendering code can be written independently of the rendering backend by using Unreals RHI, thus allowing developers to run the software on a plethora of systems. Especially in today's heterogeneous market, this is a big win as many different devices are supported with the same application allowing one to choose the hardware best suitable for the task at hand. Applications can be operated on standalone head-mounted displays running on Android or tethered on Windows with DirectX or Linux with Vulkan without any changes to the application. Functionality for application input, output, and networking are abstracted analogously to provide multiplatform support.

To gather experience with the advantages, disadvantages, and limitations of using UE for applications, we went for a naïve implementation of the application described in the use case. We decided to stick to resources that are easily accessible to new developers to create an application from the perspective of an inexperienced user. Therefore, we deliberately did not apply optimization techniques that may be proven techniques for immersive visualization experts but may be unknown to domain scientists of other disciplines. The available resources gave a good introduction to the engine and installation process and made it easy to start with development quickly. The application setup was easily done via the UE Editor, and we were able to run it on the HTC Vive Pro 2 in a matter of minutes. Resources describing how to implement the features needed for our application were broadly available. The first successful results were quickly achieved and allowed us to load the data from disk and render neuron data via simple primitives with provided UE functions, proving that UE provides both flexibility and accessibility. Due to UE's tooling, we were able to quickly change the representation of the visualization to our liking. This was primarily achieved using the graphical material editor that allows users to develop shaders in a visual editor. Changes to the shader are hot-reloaded and can be previewed on reference geometry to observe the effect. Thus allowing us to influence the visual representation of the application with ease and without specific knowledge of graphic APIs. Similarly, we could quickly and comfortably implement different interaction techniques needed for the use case. The engine provides many out-of-the-box features that can be composed to implement standard immersive visualization interactions. The UE Editor allowed us to iterate quickly over different interaction methods and visualizations. Changing parameters from the UE Editor with direct visual correspondence and no need for recompiling allowed fine-tuning of interactions without friction. Similarly, we were able to visually

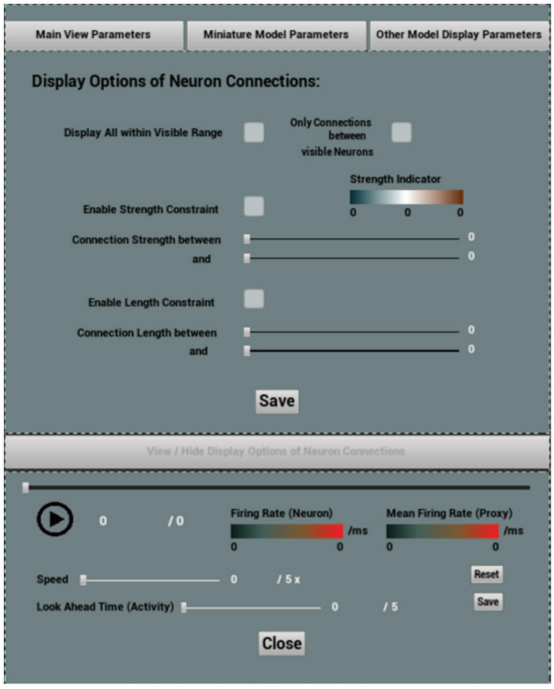

Figure 2: A complex and interactive UI for display and filter operations created within a drag and drop UI editor in Unreal Engine.

build a 2D User Interface(UI) in the WYSIWYG Editor via UE's UI framework called UMG. UE provided us with many different UI elements for input, output, and layout of the UI that are easily embeddable in the immersive visualization as spatial UI. This allowed us to easily create a rather complex UI with different elements and tabs.

While the aforementioned features allowed us to prototype our application rapidly, we experienced the first performance issues rather quickly. One bottleneck was the naive spawning and destroying of objects whenever they should be visible or hidden. To stick to a naïve implementation, neurons were created and deleted every time they were toggled from invisible to visible and vice versa. Even small numbers of on-demand object creations lead to obtrusive lags in the user experience. Table 3 shows the time for spawning different amounts of neurons as new game objects. Each neuron is programmed as a game object with interactive behavior, i.e., it can be selected/deselected and interacted with. The neurons' logic, the interaction, and the visual representation, e.g., the spiking animation, were implemented in blueprints. We observed that even for small numbers like 176 neurons, the spawning of objects on demand can take a considerable amount of time. With 0.193s, the time to create the objects takes roughly 18x the frame budget when assuming the standard of 90 frames per second. When spawning roughly half of the dataset at once, it takes nearly 2 seconds to spawn the objects, thus rendering the application impracticable. A similar but less criti-

#Nodes	Time to Spawn	FPS
200	109ms	90
720	383ms	90
1938	1097ms	45
3125	1890ms	28

Table 1: Response time for spawning distinct numbers of neurons simultaneously and the resulting framerate while active.

cal result can be observed when toggling the visibility of connections between neurons (c.f. Table 2). The cost of spawning connection objects is smaller compared to neurons since a connection has less functionality and is thus a smaller object. However, due to the high connectivity in the represented network, connections outnumber neurons. Resulting in disruptive lags when toggling meaningful amounts of connections. This behavior is especially pronounced

#Connections	Time to Spawn
2718	16ms
4877	29ms
9053	64ms
17469	133ms

Table 2: Response time for spawning distinct numbers of connections simultaneously.

when spawning objects through blueprints iteratively while setting the object's properties such as position, color, etc. Upon further analysis, it could be seen that the most severe bottlenecks are traceable to blueprint performance. Especially, tasks that require iterating over data, such as filtering, sorting, and creating objects, are noticeably slower in blueprint implementations. This is significantly limiting for immersive visualization applications as iterating over the data is a common and recurring requirement in a lot of applications. After implementing basic functionality to spawn the neurons and their connections for the main view, the functionality of the miniature view was implemented next. To reduce the number of simultaneously rendered neurons, we implemented a linked view that shows the activity in a spatially binned region as a representative neuron in a miniature view. It was quickly seen that these basic approaches helped to alleviate the issue but were still not enough to completely remedy the problems. More involved methods would have been needed to improve the application's performance further. E.g., using instanced rendering, object pooling, optimized shaders, or implementations of the features in C++. While various advanced techniques are available in the engine, the documentation often fails to explain precisely when and how to use them. Thus, those techniques were deliberately not used in the case study as the availability and necessity of these features would likely not be apparent to inexperienced users. Many online resources are written in the context of game development and show implementations that do not scale well for large applications. The simplicity of the proposed solutions often outweighs the increased complexity and difficulty to implement efficient solutions. While this is fine for a few objects in a game, the cost-benefit ratio quickly changes when applied to thousands of objects in a visualization. Documentation of advanced features is often sparse and sometimes best documented in code. Even though the source code is available, the complexity of the engine makes quick changes often prohibitive. This is especially true for features touching the core functionality, e.g., the rendering. However, these advanced features are necessary to maintain performance in immersive visualization.

In summary, the naïve implementation of the use case has shown that even highly performant engines like UE can quickly run into performance issues when not used correctly. This is especially critical in combination with the perceived accessibility of the engine. Users could easily get the false impression that the provided and prominently featured functions like blueprints can also be easily applied to immersive visualization applications. Thus, early design decisions can become a bottleneck when scaling the data size. Additionally, UE provides a lot of different building blocks to create immersive applications but not ready-to-use solutions. Interaction techniques like IntenSelect [2], which would be very useful for our use case due to the interaction with many densely packed objects, were not readily available. Thus, pre-existing knowledge in immersive visualization is needed to be aware of such techniques and to be able to implement even basic interaction techniques.

4 Conclusion

In this case study, we examined the benefits and limitations of commercial-off-the-shelf software for developing immersive visualization applications. To observe the advantages and disadvantages, we implemented a linked-view visualization for data from computational neuroscience. To investigate whether new developers in the lab or other domain scientists could implement such an application on their own, we deliberately stuck to a naïve implementation. Unreal Engine's functionality allowed us to implement interaction techniques easily, create UI elements and build a first implementation quickly. Compared to implementations on custom software, a lot of development time was saved using these features. However, it was also shown that even small data sets proved challenging for the application. This was mainly caused by the fact that features of the engine which are easy to use, often do not scale very well. Features that provide the required performance are often not documented well or prominently featured in online resources. Additionally, UE only provides building blocks for most cases that the developer must combine. In the case of interactions, for example, this requires knowledge of the immersive visualization domain to provide the best solution for the applications. Our findings suggest that implementing immersive visualization applications, while becoming easier, is still a challenging task that needs expert knowledge. Core challenges like providing performant implementations and system design are still difficult to solve at scale. Additionally, advanced user interactions and navigation techniques are often not provided out of the box. Thus, domain scientists will still benefit significantly by working with experts from immersive visualization labs. On the other hand, those experts can efficiently use the functionality provided by game engines like Unreal Engine. Existing COTS software, like game engines, often provides powerful and optimized features that can be utilized by experienced developers aware of advanced methods. Furthermore, it provides many convenient features that make development easier and more convenient for developers of all levels. Especially open source engines are often extensible and adaptable, thus, providing the opportunity to customize the engine to the lab's needs. While using such software makes the development of applications more accessible to new developers, such as students joining the lab, the accessibility can also be a disadvantage. New developers must be sensitized to make them performance-aware early on to prevent them from over-reliance on the engine's optimization. Therefore, educating new developers on the advantages and disadvantages of using such tools and providing concrete guidelines for applications in immersive visualization is essential. Experience obtained with specific tools must be turned into best practices to guide users through the development and stop preventable bottlenecks from occurring. With these limitations in mind, our case study suggests that game engines can be an excellent base for immersive visualization labs to build upon. The performance, usability, and flexibility that modern engines are built with can accelerate the development and do not lessen the need for immersive visualization researchers and labs. Taking advantage of existing COTS software and hardware allows labs to advance scientific knowledge by focusing their expert knowledge on immersive visualization methods instead of technical implementations.

Acknowledgments

This project/research has received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under the Specific Grant Agreement No. 945539 (Human Brain Project SGA3).

REFERENCES

[1] J. Buchertseifer, S. Spreizer, and B. Weyers. Nest desktop, Feb. 2022. doi: 10.5281/zenodo.6320318

[2] G. De Haan, M. Koutek, and F. H. Post. Intenselect: Using dynamic object rating for assisting 3d object selection. In *Ipt/egve*, pp. 201–209, 2005.

[3] Epic Games. Unreal Engine. `https://www.unrealengine.com`.

[4] S. E. Galindo, P. Toharia, O. D. Robles, and L. Pastor. Visimpl: multi-view visual analysis of brain simulation data. *Frontiers in Neuroinformatics*, 10:44, 2016.

[5] M.-O. Gewaltig and M. Diesmann. Nest (neural simulation tool). *Scholarpedia*, 2(4):1430, 2007.

[6] C. Hänel, P. Pieperhoff, B. Hentschel, K. Amunts, and T. Kuhlen. Interactive 3d visualization of structural changes in the brain of a person with corticobasal syndrome. *Frontiers in neuroinformatics*, 8:42, 2014.

[7] S. Jaeger, K. Klein, L. Joos, J. Zagermann, M. De Ridder, J. Kim, J. Yang, U. Pfeil, H. Reiterer, and F. Schreiber. Challenges for brain data analysis in vr environments. In *2019 IEEE Pacific Visualization Symposium (PacificVis)*, pp. 42–46. IEEE, 2019.

[8] J. J. Keiriz, L. Zhan, M. Chukhman, O. Ajilore, A. D. Leow, and A. G. Forbes. Exploring the human connectome topology in group studies. *arXiv preprint arXiv:1706.10297*, 2017.

[9] D. Lancour, J. Dupuis, R. Mayeux, J. L. Haines, M. A. Pericak-Vance, G. C. Schellenberg, M. Crovella, L. A. Farrer, and S. Kasif. Analysis of brain region-specific co-expression networks reveals clustering of established and novel genes associated with alzheimer disease. *Alzheimer's research & therapy*, 12(1):1–11, 2020.

[10] S. Marks. Immersive visualisation of 3-dimensional spiking neural networks. *Evolving Systems*, 8(3):193–201, 2017.

[11] E. Nordlie and H. E. Plesser. Visualizing neuronal network connectivity with connectivity pattern tables. *Frontiers in Neuroinformatics*, p. 39, 2010.

[12] C. Nowke, M. Schmidt, S. J. Van Albada, J. M. Eppler, R. Bakker, M. Diesrnann, B. Hentschel, and T. Kuhlen. Visnest—interactive analysis of neural activity data. In *2013 IEEE Symposium on Biological Data Visualization (BioVis)*, pp. 65–72. IEEE, 2013.

[13] B. Pester, O. Winke, C. Ligges, R. Dachselt, and S. Gumhold. Immersive 3d visualization of multi-modal brain connectivity. 2021.

[14] C. K. Tamnes, A. M. Fjell, Y. Østby, L. T. Westlye, P. Due-Tønnessen, A. Bjørnerud, and K. B. Walhovd. The brain dynamics of intellectual development: Waxing and waning white and gray matter. *Neuropsychologia*, 49(13):3605–3611, 2011.

[15] Unity Technologies. Unity. `https://unity.com`.

[16] L. J. Van Eldik and W. S. T. Griffin. S100β expression in alzheimer's disease: relation to neuropathology in brain regions. *Biochimica et Biophysica Acta (BBA)-Molecular Cell Research*, 1223(3):398–403, 1994.

[17] A. von Kapri, T. Rick, T. C. Potjans, M. Diesmann, and T. Kuhlen. Towards the visualization of spiking neurons in virtual reality. In *Medicine Meets Virtual Reality 18*, pp. 685–687. IOS Press, 2011.

[18] C. Ware and P. Mitchell. Visualizing graphs in three dimensions. *ACM Transactions on Applied Perception (TAP)*, 5(1):1–15, 2008.