NetV.js: A Large-Scale Graph Visualization Library

Dongming Han, Jiacheng Pan, Rusheng Pan, Jiehui Zhou, Xiaodong Zhao[a,b], Wei Chen[a,*]

[a] *State Key Lab of CAD&CG, Zhejiang University, Hangzhou, Zhejiang, China*
[b] *Zhejiang Lab, hangzhou, zhejiang, China*

Abstract

Network visualization plays an important role in many fields, such as social media networks, protein-protein-interaction networks, and traffic networks. In the meantime, a dozen of visualization designer tools and programming toolkits are widely used in implementing network visualization applications. With network data growth, exiting tools can not help users rapidly and efficiently construct large-scale network visualization. In this paper, we present NetV.js, an open-sourced, JavaScript-based, WebGL-based library that supports the rapid large-scale network rendering, interaction, and visualization.

Keywords: Network visualization, Web-based visualization

1. Introduction

图数据在现实世界广泛存在。图数据的可视化在很多领域都有其重要作用，比如在金融数据可视分析中可视化欺诈交易，探索社交媒体网络中的信息传播，以及展示生物网络中的蛋白质互相作用等等。 为了提升用户和开发者构建可视化的效率，一系列可视化构建工具被提出。特别是d3的提出，降低了基于web的可视化构建的难度，丰富了可视化社区。因此，很多图数据可视化工具也基于web进行开发。其中，节点链接图是最为广泛使用的可视化形式。[pan]

Visualization of graph data plays an important role in many fields, such as showing fraud transactions in financial data analysis [1], exploring information propagation in social media network [2], and visualizing protein-protein-interaction in biological network [3]. A series of visualization authoring tools have been developed to facilitate visualization generation. Notably, the proposal of d3.js [4] reduces the difficulty of web-based visualization authoring and enriches the visualization community. Thus, a great part of graph visualization tools are also web-based. Node-link diagrams are widely used among many graph visualization solutions, because they reveal topology and connectivities [5].[pan]

节点链接图可视化拥有其特点，比如每一条链接往往链接了两个节点，用户只需控制节点的位置，其链接的位置会相应变化；节点链接图依赖于布局算法，从图数据到可视化的映射，需要用户通过布局算法来处理节点的位置摆放；针对节点的交互（比如点击和拖拽）非常普遍。开发者在使用通用的可视化工具进行开发时，需要pay effort to 处理这些问题。比如在使用D3.js进行图数据可视化时，开发者需要使用其数据驱动文档的思想将对应的数据映射到对应的可视化元素上，开发者直接面向可视化元素进行开发，没有抽象的模型来支持他控制整个图数据，可能会导致编程错误的产生。 一些图可视化构建工具，比如xxxx，良好地支持了对图数据的可视化，相对于通用的可视化工具而言，他们通过解决节点链接图的特殊需求，封装图可视化的相关接口，隐藏一部分开发者不关心的接口（比如对边的位置控制），暴露一部分图可视化特性接口（对相邻节点的访问），来提升自身的易用性。[pan]

The node-link diagrams visualization has its speciality. For example, one link in a node-link diagram connects two nodes. User only need to control the nodes' positions and related links change their positions correspondingly; node-link diagrams strongly rely on layout algorithms; interactions on nodes such as clicking and dragging are frequently needed. Developers need to pay efforts to deal with these specialities using general visualization authoring tools such as d3.js [4], p5 [6], and stardust [7]. For example, when a developer uses d3.js to create a node-link diagram, s/he needs to map data elements to graphical marks using the data-drive-documents idea. The developer directly handles the visual elements and there is no hook/handler for s/he to handle the underlying graph model. It may lead to bugs. Some graph visualization authoring tools such as Cytoscape.js [8] and Sigma.js [9] support graph visualization efficiently. Compared to general visualization tools, tools for graph visualization encapsulate related interfaces of graph visualization, hide some unrelated interfaces (e.g. controlling the position of a link) for developers, and expose some graph-related interfaces (e.g. accessing neighborhoods of a node). They improve the usability through full-fill requirements of node-link diagrams.[pan]

随着数据规模的增长，图可视化也需要处理更多graphical marks（比如节点和链接）。然而，现有大部分图可视化工具难以处理有较多graphical marks。根据我们的实验，

*Corresponding author
Email address: chenvis@zju.edu.cn (Wei Chen)

它们在渲染较大规模数据集上存在延迟，这将会降低用户的accessibility。[pan]

With the growth of data scale, graph visualization authoring tools need to handle more graphical marks (e.g. nodes and links). However, most of existing tools are unable to handle a large number of graphical marks. According to our experiments in Section 4, existing graph visualization authoring tools have delays in rendering large-scale graph data (with more than ...). It reduces the user accessibility.[pan]

据我们所知，尚未存在一款工具，能够解决开发者易用性以及用户可访问性的问题。我们探索了图可视化的相关设计需求，设计并实现了NetV，一款基于web的高性能图可视化工具。其通过设计一系列图可视化相关的功能和接口来提高开发者易用性，并调用了GPU的高性能渲染能力来提高渲染效率以增加用户的可访问性。 通过和其他工具的对比实验，我们验证了NetV的accessibility和usability。我们对该工具进行了开源以便开发者访问和贡献代码。[pan]

To best of our knowledge, no existing tool can address developer usability and user accessibility in the same time. We explored the design requirements of the node-link diagram visualization, designed and developed NetV.js, a web-based high-performance node-link diagram visualization library. It improves developer usability through a serials of node-link diagram related functions and interfaces and increases user accessibility by utilizing the high-performance rendering ability of the GPU. We also evaluated the usability and accessibility of NetV.js through several comparative experiments. NetV.js is now open-sourced (`https://netv.zjuvag.org/`) for developers to access and contribute their own code.[pan]

2. Related Work

Many network grammars and frameworks have been provided to developers for designing network visualization applications. In the beginning, developers use conventional programming languages to construct network visualization applications, such as C Sharp, C++, Javascript, and Python. Developers need to have proficient programming skills and understand the implementation mechanism. In the meantime, they also spend a lot of time developing and debugging. To make it easier for developers to program and develop quickly, visualization grammars and frameworks give granular control of visual channels of visualizations such as D3.js [4], ECharts [10], Vega [11], and Vega-Lite [12]. Developers can use more concise tools to construct visualizations.

However, when developers construct network visualizations, they need to carefully select complex and different API to construct a network, because these APIs are designed for full visualization rather than network visualization applications. Cytoscape.js [8], sigmajs [9] and Gephi [13] are used to construct network visualizations. They encapsulate a series of API which are elaborated for the network data. Developers can use them to construct network visualizations quickly. Moreover, large-scale data brings new challenges. These grammars and tools can only render thousands of elements. To address this issue, PixiJS [14], P5 [6] and Stardust [7] used GPU-based acceleration technology to render a large number of elements. However, these GPU-based tools are also not designed for network visualizations. They still have redundancy and complex API that have nothing to do with network visualizations. It leads to a decrease in rendering efficiency and an increase in learning costs.

Our NetV.js focus on large-scale network visualization. It uses a GPU-based rendering engine to support large-scale data and design-friendly concise programming interfaces for network visualization construction.

3. Design of NetV.js

3.1. Design Requirements of NetV.js

为了探索NetV.js的设计空间，我们采访了3个图可视化相关的专家，调研了一系列图可视化的工具，包括Gephi，Cytoscape.js，Sigma.js，GraphViZ，总结了如下高性能节点链接图可视化的设计需求:[pan]

To explore the design space of NetV.js, we interviewed three graph visualization experts, investigated six graph visualization tools including Gephi [13], Pajek [15], SNAP [16], Sigma.js [9], GraphViZ [17], and Cytoscape.js [8]. We summarized the following design requirements for high-performance node-link diagram visualization: [pan]

- **R1** **需要有抽象的图模型来帮助控制图可视化**: 为了简化开发者对于可视化的操作，NetV.js需要一个抽象图模型来控制图可视化而非直接控制图可视化的元素。该模型需要支持图的以下几点特性: [pan]
  - R1.1 **链接关联节点**: 每条链接都会关联到两个节点是图结构最基本的特性，该特性使得绘制节点链接图可以不关心链接的位置，开发者可以专注于节点的位置修改，链接会相应联动。[pan]
  - R1.2 **邻节点和邻接边的可访问性**: 我们在调研过程中发现，很多可视化系统中支持了高亮某个节点的邻接边和邻节点的交互。所有专家都一致同意为图模型增加邻节点和邻接边可访问功能的重要性。[pan]
  - R1.3 **基本图论算法的支持**: 部分算法提供了一些图论的基本算法，比如计算某些度量（如节点度数，节点centrality，图直径等），或是获取两个节点之间的最短路。专家们都一致同意图论算法对于可视化的重要性，但他们部分同意实现图论算法的必要性，其中一个专家认为该功能不属于可视化渲染库所关心的需求。[pan]
- **R2** **需要对拥有大量可视化元素的节点链接图提供高刷新率**: 根据我们对三位图可视化专家的采访，他们认为【对超过10万元素的大规模图有30fps以上的渲染速度】才能认为其对大规模节点链接图提供了高性能渲染的能力。[pan]
- **R3** **支持基本的节点和链接的样式**: 开发者往往需要在节点链接图上编码不同信息，虽然在大部分大规模图可视化案例中，圆形的节点和直线边最受欢迎，但仍然有很多节点链接图支持了不同形状的节点和

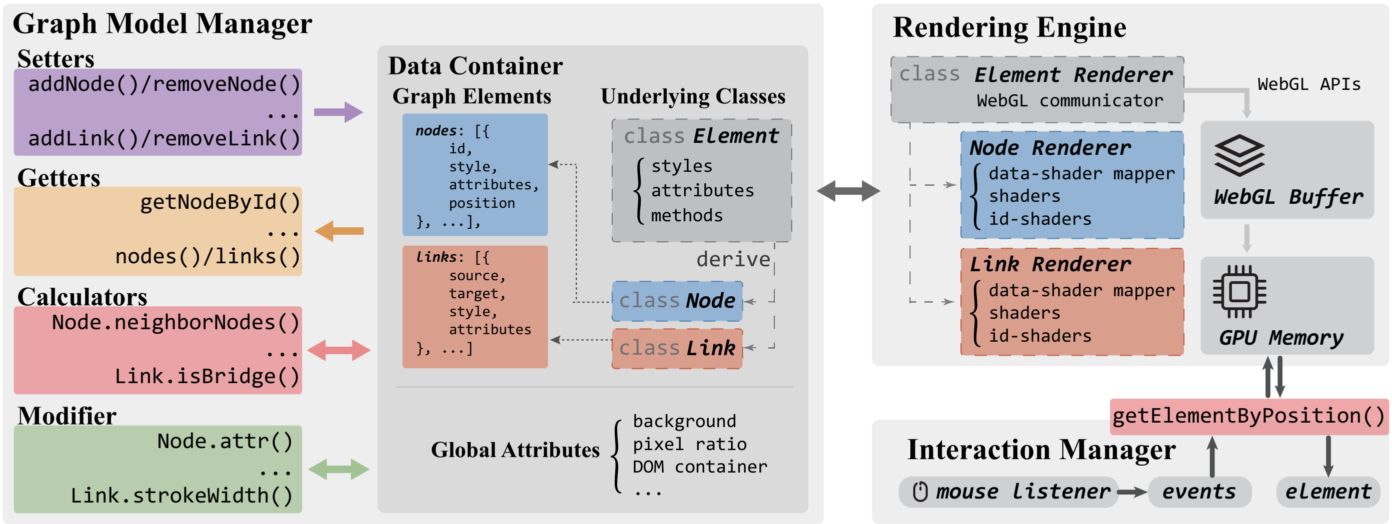

Figure 1: NetV.js designs: NetV.js consists of three parts: core engine, plugins, and library interface.

不同样式的边。因此，为节点和链接赋予不同的形状和样式，能使得开发者有更多可以编码信息的空间。最常见的节点形状有圆形、方形、三角形等，而最常见的链接形状有直线、曲线等pan

R4 需要提供多种布局功能和自定义布局插件：节点链接图对于布局的依赖程度不言而喻。NetV.js需要提供一些基础的节点链接图布局。因为布局算法的多样性，NetV.js还需能使开发者按照既定接口接入自定义布局。pan

R5 需要提供自定义标签渲染：虽然在渲染大规模节点链接图时，为可视化元素赋予标签会导致visual clutter的问题，但开发者仍有可能采取某些策略来绘制标签，比如在屏幕内元素数量较少时自适应地绘制标签。所以，NetV.js有必要提供标签绘制的接口。pan

R6 需要提供基础交互：虽然对于节点链接图的交互多种多样，但根据我们的调研和专家访谈，我们发现它们基本上都可以被分解为对节点、链接以及画布的交互监听。我们认为NetV.js需要能够对节点进行拖拽、点击、鼠标悬浮的监听，对链接需要有点击和鼠标悬浮的监听，以及对画布需要能够有平移、缩放、点击的交互监听。我们还需要实现对节点链接图的可视化元素的选择功能，比如lasso。pan

3.2. Design Details of NetV.js

为了完成上述的需求，我们设计实现了NetV.js。NetV.js包含了三个主要部分：*Graph Model Manager*、*Rendering Engine*以及*Interaction Manager*。我们还将一些非核心的需求独立抽象为插件的形式方便开发者自定义调用，从而减少NetV.js的核心代码。pan

3.2.1. Graph Model Manager

为了能够有抽象的图模型来帮助图可视化，我们在NetV.js里设计实现了*Graph Model Manager*，一个图模型的控制器（图1）。该控制器的主体是一个Data Container，其存储了节点数据和链接数据。每个节点拥有一个唯一标志符id，其对应的样式会被存储在style属性中，而位置坐标责备存储在position中，其余的属性则会被存储在attributes中；同样的，每个link都拥有一个source和一个target，代表它所关联的两个节点，其对应的样式和其余属性会被存储在style和attributes中。我们为节点和链接建立了对应的class：Node和Link来存储每一个数据实体。他们派生自Element类。

我们为data elements (nodes和links) 设计了 增、删、改、查和计算的功能。开发者可以通过data setter来增加和删除elements，通过 data getter 则能用于查询element，通过 data modifier 修改数据的内容（如属性、样式等）。除此以外，我们还提供了部分常用的图计算的接口（R1.2和R1.3）因为该部分和可视化的关联并不紧密，我们只实现了部分功能，比如查询节点的邻节点（R1.2）等。

*Graph Model Manager*还包括了一些除了节点、链接以外的数据管理。比如对图模型整体的配置，画布的背景颜色，DOM容器等等。

3.2.2. Rendering Engine

为了满足**R2**，*Rendering Engine*调用了GPU的高性能渲染来绘制可视化。*Rendering Engine*的主要工作流程是将节点和链接的样式以及位置，映射到 WebGL的 shader 程序中，通过WebGL API 将数据输入 WebGL Buffer，以此沟通 浏览器内存和GPU内存。最终，开发者输入的样式和位置等，会通过 *Rendering Engine*交付给GPU，并绘制到屏幕上。

我们采用了一种被动式的buffer修改策略：当开发者修改样式时，*Graph Model Manager*不会主动访问 *Rendering Engine*以修改WebGL buffer的内容，而仅仅将被修改的节点存到一段修改元素缓冲池中。当开发者调用重绘制的API时，*Rendering Engine*遍历所有缓存池的元素以获取样式，然后根据 数据-shader的映射表，将样式映射到对应的WebGL Buffer上进行覆盖。相对的主动式的buffer修改策略，则是当开发者修改样式时，*Graph Model Manager*直接访问 *Rendering Engine*以修改WebGL buffer的内容，需要从*Graph Model Manager*访问 *Rendering Engine*的相关接口，导致调用栈加深，降低效率。

为了解决R1.1，我们禁止了开发者直接修改link的位置，而是当开发者修改node的位置时，将该节点和其所有邻接边都放入缓冲池，后续则会同步修改邻接边的表现。

我们的 *Rendering Engine*还为节点和链接增加了许多基础的样式来满足R3。 我们为节点设计了四种不同的形状，包括圆形，矩形，三角形和十字形。开发者可以控制圆形节点的半径，矩形节点的长款，三角形节点的三个顶点和十字形顶点的粗细和大小。所有形状的节点都可设置填充颜色、边框颜色和边框宽度。 我们为链接设计了两种不同的形状，分别为直线和曲线。开发者可以控制曲线的曲率。所有的链接都可以设置颜色、宽度和虚线。

3.2.3. Interaction Manager

*Interaction Manager*为 NetV.js提供了一系列基础的交互以满足R3。通过为画布增加对鼠标事件的监听，我们实现了对画布的缩放、平移、点击的交互。对于链接，我们实现了鼠标进入、鼠标离开和点击的交互。对于节点，我们实现了鼠标进入、鼠标离开、点击、拖拽的交互，其中针对拖拽事件，我们将其分离为拖拽开始，拖拽中和拖拽结束等不同步骤。其中，点击交互还可以细分成鼠标按下和鼠标松开两个步骤进行监听。

为了能够在屏幕上找出鼠标交互事件发生在哪个元素或者是画布上，我们在 *Rendering Engine*中增加了一个隐藏层。该隐藏层将元素的唯一标志符编码到屏幕像素上。于是，通过获知鼠标交互事件发生的屏幕像素的编码，我们就能获知交互的目标。

3.2.4. Layout Plugin

4. Experiment

We conducted the frame per second (FPS) experiment[1] to test the rendering performance of our NetV.js with other popular tools and libraries which are supports network rendering, including D3-SVG, D3-Canvas, Cytoscape.js, Sigma.js (WebGL), and Stardust.js. In particular, NetV.js, Sigma.js, and Stardust.js use WenGL to render data. To simulate real-world network data, we set the density of the network as 20, it means the ratio of the number of edges to the number of nodes is 1 to 20. The display refresh rate is 144Hz, the GPU is GTX 1060 with 6G.

From the line chart (Figure 2), NetV.js, Stardust.js, and D3-Canvas can render around 100,000 elements. Our NetV.js can render more than 1 million elements with FPS greater than 1.

5. Conclusion

This paper presents NetV.js, an open-sourced, JavaScript-based, WebGL-based library that supports the rapid large-scale network rendering, interaction, and visualization. In the future, we plan to extend NetV.js to support heterogeneous networks. We also plan to explore more visualization components and visual analysis algorithms for analyzing network data.

[1] https://github.com/ZJUVAG/NetV.js/tree/benchmarks/benchmarks

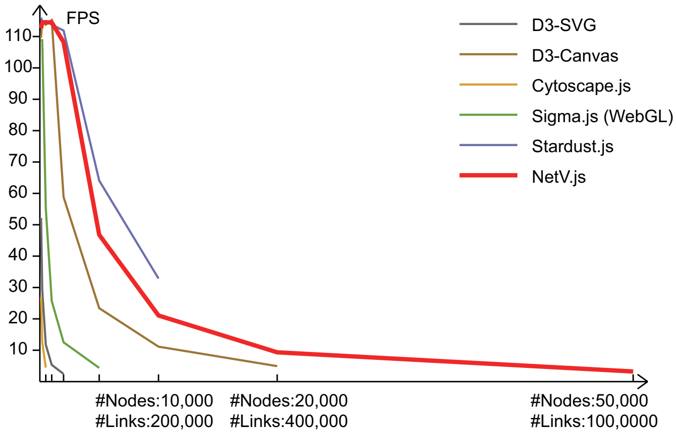

Figure 2: The Frame per Second (FPS) experiment.

References

[1] W. Chen, F. Guo, D. Han, J. Pan, X. Nie, J. Xia, X. Zhang, Structure-based suggestive exploration: A new approach for effective exploration of large networks, IEEE Trans. Vis. Comput. Graph. 25 (1) (2019) 555–565. doi:10.1109/TVCG.2018.2865139.
URL https://doi.org/10.1109/TVCG.2018.2865139

[2] M. A. Smith, B. Shneiderman, N. Milic-Frayling, E. M. Rodrigues, V. Barash, C. Dunne, T. Capone, A. Perer, E. Gleave, Analyzing (social media) networks with nodexl, in: J. M. Carroll (Ed.), Proceedings of the Fourth International Conference on Communities and Technologies, C&T 2009, University Park, PA, USA, June 25-27, 2009, ACM, 2009, pp. 255–264. doi:10.1145/1556460.1556497.
URL https://doi.org/10.1145/1556460.1556497

[3] N. T. Doncheva, Y. Assenov, F. S. Domingues, M. Albrecht, Topological analysis and interactive visualization of biological networks and protein structures, Nature protocols 7 (4) (2012) 670.

[4] M. Bostock, V. Ogievetsky, J. Heer, D^3 data-driven documents, IEEE Trans. Vis. Comput. Graph. 17 (12) (2011) 2301–2309. doi:10.1109/TVCG.2011.185.
URL https://doi.org/10.1109/TVCG.2011.185

[5] M. Ghoniem, J.-D. Fekete, P. Castagliola, A comparison of the readability of graphs using node-link and matrix-based representations, in: Proceedings of IEEE Symposium on Information Visualization, 2004, pp. 17–24. doi:10.1109/INFVIS.2004.1.

[6] J. K. Li, K. Ma, P5: portable progressive parallel processing pipelines for interactive data analysis and visualization, IEEE Trans. Vis. Comput. Graph. 26 (1) (2020) 1151–1160. doi:10.1109/TVCG.2019.2934537.
URL https://doi.org/10.1109/TVCG.2019.2934537

[7] D. Ren, B. Lee, T. Höllerer, Stardust: Accessible and transparent GPU support for information visualization rendering, Comput. Graph. Forum 36 (3) (2017) 179–188. doi:10.1111/cgf.13178.
URL https://doi.org/10.1111/cgf.13178

[8] M. Franz, C. T. Lopes, G. Huck, Y. Dong, S. O. Sümer, G. D. Bader, Cytoscape.js: a graph theory library for visualisation and analysis, Bioinform. 32 (2) (2016) 309–311. doi:10.1093/bioinformatics/btv557.
URL https://doi.org/10.1093/bioinformatics/btv557

[9] J. Coene, sigmajs: An R htmlwidget interface to the sigma.js visualization library, J. Open Source Softw. 3 (28) (2018) 814. doi:10.21105/joss.00814.
URL https://doi.org/10.21105/joss.00814

[10] D. Li, H. Mei, Y. Shen, S. Su, W. Zhang, J. Wang, M. Zu, W. Chen, Echarts: A declarative framework for rapid construction of web-based visualization, Vis. Informatics 2 (2) (2018)

136–146. doi:10.1016/j.visinf.2018.04.011.
URL https://doi.org/10.1016/j.visinf.2018.04.011

[11] A. Satyanarayan, R. Russell, J. Hoffswell, J. Heer, Reactive vega: A streaming dataflow architecture for declarative interactive visualization, IEEE Trans. Vis. Comput. Graph. 22 (1) (2016) 659–668. doi:10.1109/TVCG.2015.2467091.
URL https://doi.org/10.1109/TVCG.2015.2467091

[12] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, J. Heer, Vega-lite: A grammar of interactive graphics, IEEE Trans. Vis. Comput. Graph. 23 (1) (2017) 341–350. doi:10.1109/TVCG.2016.2599030.
URL https://doi.org/10.1109/TVCG.2016.2599030

[13] M. Bastian, S. Heymann, M. Jacomy, Gephi: An open source software for exploring and manipulating networks, in: E. Adar, M. Hurst, T. Finin, N. S. Glance, N. Nicolov, B. L. Tseng (Eds.), Proceedings of the Third International Conference on Weblogs and Social Media, ICWSM 2009, San Jose, California, USA, May 17-20, 2009, The AAAI Press, 2009.
URL http://aaai.org/ocs/index.php/ICWSM/09/paper/view/154

[14] C. G. I. Graphics, Learn pixi. js.

[15] V. Batagelj, A. Mrvar, Pajek, in: Encyclopedia of Social Network Analysis and Mining, 2014, pp. 1245–1256. doi:10.1007/978-1-4614-6170-8_310.
URL https://doi.org/10.1007/978-1-4614-6170-8_310

[16] J. Leskovec, R. Sosič, Snap: A general-purpose network analysis and graph-mining library, ACM Transactions on Intelligent Systems and Technology (TIST) 8 (1) (2016) 1.

[17] J. Ellson, E. R. Gansner, E. Koutsofios, S. C. North, G. Woodhull, Graphviz and dynagraph – static and dynamic graph drawing tools, in: GRAPH DRAWING SOFTWARE, Springer-Verlag, 2003, pp. 127–148.