The Journal of Open Source Software

The Plebian Graph Library: A WebGL based network visualisation and diagnostics package

Indrajeet Haldar [1]

1 Graduate School of Design, Harvard University, USA

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

Given a large network (greater than one million nodes), visualising and diagnosing network data has often proven challenging (Nowogrodzki, 2015). Although there is a wide range of statistical tools to draw inferences, the esoteric nature of the statistical analysis of networks limits the communication of the findings to researchers familiar with these research methods (Tobi & Kampen, 2018). Moreover, the more abstract nature of statistical solutions reduces the complexity of diagnosing network properties, especially network diffusion. It allows for limited documentation and analysis only from the perspective of mathematical models of this particular property of large-scale networks (Gao & Guan, 2012). The Plebian Graph Library (PGL) is a library that solves for the visualisation of large networks and their diagnostic study.

The Plebian Graph Library (PGL) is a JavaScript library (written in Typescript (Bierman et al., 2014)) designed to facilitate the visualisation and diagnostic analysis of large-scale network data in browsers using WebGL, using a backend provided by ThreeJS (Danchilla, 2012). Whether dealing with local datasets or data retrieved from online sources (APIs), PGL provides a versatile platform for conducting extensive network simulations, physical modelling, and visualisations whilst offering a range of diagnostic tools for organising network data using standard search algorithms (Mattson et al., 2013) such as network diffusions, Breadth-first search, Depth-first search and Dijkstra's search algorithm. With a rich set of diagnostic features, including network condensation, weighted edge pruning in highly connected graphs, and support for visualisation techniques like Kamada Kawai layouts (Kamada & Kawai, 1989), Hierarchical plots, Hive plots and Edge bundling (Bourqui et al., 2016), PGL empowers researchers to gain valuable insights from complex network structures. Additionally, PGL contains canonical example of the Zackary's Karate Club (ZKC) dataset (Zachary, 1977), and the Erdosh Reyni Random Graph model (Li, 2021) as a generator to study and compare network structures.

An illustrative case for the package is to diagnose large-scale network diffusion. Visualising a clustered network in 3D, where, for example, the network nodes are displaced vertically according to their recursive importance, i.e. eigenvector centralities (Lacobucci et al., 2017). A diffusion simulation is then run, and insights and diagnostics of diffusion sequences are gathered. For example, whether diffusion first occurs between high eigenvector centrality nodes across clusters or does the spread appear in the groups before spreading to other clusters, allowing for the visual study of the strength of weak ties behaviour (Granovetter, 1973). Exploratory research, analysis, communication, and documentation of these network behaviours, as mentioned above, would have been complex in a traditional visualisation library where the emphasis lies on validation instead of exploratory study and diagnostics.

Statement of need

The Plebian Graph Library (PGL) addresses several critical needs in large-scale graph data visualization. Existing software solutions for visualizing large datasets, such as Gephi (Bastian

et al., 2009), are limited to local machine installations, restricting accessibility and compatibility across various devices. Additionally, browser-based software libraries like Vis.JS (V., 2021) and D3[d3ieee], which rely on Scalable Vector Graphics (SVG), often lack the scalability to analyze complex network structures. This reliance on SVG imposes performance limitations and restricts visualizations to two dimensions.

In contrast, PGL offers a robust browser-based solution leveraging WebGL through the ThreeJS Library. This enables it to surpass traditional two-dimensional representations' limitations. PGL is primarily designed for client-side rendering, taking full advantage of the capabilities of WebGL to deliver dynamic and interactive visualizations directly within the browser. While it focuses on client-side rendering, the underlying graph algorithms of PGL can also be utilized in server-side processes, providing flexibility in application architecture.

A performance benchmark conducted against D3, an industry-standard visualization library, showcases PGL's capabilities. In this test involving approximately 5000 nodes and 200000 edges, D3-based SVG graphs only achieved a frame rate of 1.5 frames per second, bottoming at a frame every two seconds with a maximum of 12 frames per second. In contrast, PGL maintained a minimum of 52 frames per second and averaging 58 frames per second under similar conditions. This benchmark, performed on both Firefox and Chrome browsers (with negligible differences in performance) on a computer with an Nvidia RTX 2080 GPU, highlights PGL's superior performance and efficiency in rendering complex network visualizations.

Furthermore, PGL's three-dimensional rendering approach allows for a more comprehensive range of data stratification methods and facilitates more immersive and interactive visualizations. The ability to navigate information-dense networks in three dimensions significantly reduces visual noise and enhances clarity in diagnosing large-scale networks. Since its inception, PGL has played a crucial role in academic research, particularly in the study of large-scale social networks, as demonstrated in the academic thesis "On the Mathematics of Memetics" (Haldar, 2022), where it was used to generate primary inferences.

Usage

Existing network libraries like NetworkX (Hagberg et al., 2008) dictated the semantics of the graph library and borrowed some of the semantic ideas from ThreeJS. The overall structure is to define a Graph Object made of nodes and edges. Then, modify this graph based on some properties and update the relevant settings. Lastly, visualise the nodes as point clouds, boxes or cylinders, and draw out the edges (bundled or not). The following is a short example of the canonical ZKC dataset visualised in the library, simulated with Edge bundling.

First, initialize a node project and install the library using :

```
npm i plebiangraphlibrary
```

Then

```
// import the library
import * as PGL from "plebiangraphlibrary";

async function createVisualization() {
  // Load up the ZKC dataset
  const zkcSimulated = await PGL.SampleData.LoadZKCSimulated();

  // Attach the renderer to a div which is on an HTML that the script is linked too
  const canvas = document.getElementById("displayCanvas");
  // These are some basic options to set up a graph drawing object. Please refer to the
  const graphDrawerOptions = {
    graph: zkcSimulated,
```

```
    width: 800,
    height: 700,
    canvas: canvas,
  };

  // Initialize a graph with these settings
  const graph3d = new PGL.GraphDrawer.GraphDrawer3d(graphDrawerOptions);
  await graph3d.init();

  // Create the 3d elements for the graph
  // first describe a global scaling factor
  const bounds = 1;
  // Create all the geometry associated with node elements
  const nodeVisualElements = PGL.ThreeWrapper.DrawTHREEBoxBasedVertices(
    zkcSimulated,
    bounds,
    0xffffff,
    5
  );
  // add the node geometry to the scene
  graph3d.addVisElement(nodeVisualElements);
  // then create all the geometry associated with the edge elements
  const edgeVisualElements = PGL.ThreeWrapper.DrawTHREEGraphEdgesThick(
    zkcSimulated,
    bounds,
    0xffafcc,
    0.02
  );
  // add the edge geometry to the scene
  graph3d.addVisElement(edgeVisualElements);

  // by default the camera revolves around the graph, trigger the animation call
  function animate() {
    requestAnimationFrame(animate);
    graph3d.rendercall();
  }

  animate();
}

createVisualization();
```

Documentation

Package documentation is available on GitHub for the package. Guides for general guides and detailed descriptors of all the functions are also included. Further documentation is available at https://www.plebiangraphlibrary.com/. Examples are available at https://www.plebiangraphlibrary.com/examples.html. A boilerplate example walking through the installation and creation of an example library is documented here https://github.com/range-et/pgl_example, this is the same example documented above.

The Journal of Open Source Software

Acknowledgements

The Geometry Lab, under the Laboratory for Design Technologies at the Graduate School of Design at Harvard University, funded this project.

References

Bastian, M., Heymann, S., & Jacomy, M. (2009). Gephi: An open source software for exploring and manipulating networks. *International AAAI Conference on Weblogs and Social Media*. https://doi.org/10.1609/icwsm.v3i1.13937

Bierman, G., Abadi, M., & Torgersen, M. (2014). Understanding typescript. *European Conference on Object-Oriented Programming*, 257–281. https://doi.org/10.1007/978-1-4842-4979-6_2

Bourqui, R., Ienco, D., Sallaberry, A., & Poncelet, P. (2016). Multilayer graph edge bundling. *2016 IEEE Pacific Visualization Symposium (PacificVis)*, 184–188. https://doi.org/10.1109/PACIFICVIS.2016.7465267

Danchilla, B. (2012). Three.js framework. *Beginning WebGL for HTML5*.

Gao, X., & Guan, J. (2012). Network model of knowledge diffusion. *Scientometrics*, *90*(3), 749–762. https://doi.org/10.1007/s11192-011-0554-z

Granovetter, M. S. (1973). The strength of weak ties. *American Journal of Sociology*, *78*(6), 1360–1380. https://doi.org/10.1093/oso/9780195159509.003.0010

Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). Exploring network structure, dynamics, and function using NetworkX. *Proceedings of the 7th Python in Science Conference (SciPy2008)*, 11–15.

Haldar, I. (2022). *On the mathematics of memetics* [Master's thesis]. Graduate School of Design, Harvard University.

Kamada, T., & Kawai, S. (1989). An algorithm for drawing general undirected graphs. *Information Processing Letters*, *31*(1), 7–15. https://doi.org/10.1142/9789814434478_0005

Lacobucci, D., McBride, R., & Popovich, D. L. (2017). Eigenvector centrality: Illustrations supporting the utility of extracting more than one eigenvector to obtain additional insights into networks and interdependent structures. *Journal of Social Structure*, *18*(1), 1–22. https://doi.org/10.21307/joss-2018-003

Li, J. (2021). Brief overview of graph theory: Erdos-renyi random graph model and small world phenomenon. *University of Chicago*.

Mattson, T., Bader, D., Berry, J., Buluç, A., Dongarra, J., Faloutsos, C., Feo, J., Gilbert, J., Gonzalez, J., Hendrickson, B., Kepner, J., Leiserson, C., Lumsdaine, A., Padua, D., Poole, S., Reinhardt, S., Stonebraker, M., Wallach, S., & Yoo, A. (2013). Standards for graph algorithm primitives. *2013 IEEE High Performance Extreme Computing Conference (HPEC)*, 1–2. https://doi.org/10.1109/HPEC.2013.6670338

Nowogrodzki, A. (2015). Eleven tips for working with large data sets. *Nature*, *527*(7576), 105–107. https://doi.org/10.1038/d41586-020-00062-z

Tobi, H., & Kampen, J. K. (2018). Research design: The methodology for interdisciplinary research framework. *Quality & Quantity: International Journal of Methodology*, *52*(3), 1209–1225. https://doi.org/10.1007/s11135-017-0513-8

V., A. B. (2021). *Vis.js*. https://visjs.org/

Zachary, W. W. (1977). An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, *33*, 452–473. https://doi.org/10.1086/jar.33.4.3629752