**Lines:** 92

# Browsing large graphs with MSAGLJS, a graph draph drawing tool in JavaScript

Lev Nachmanson and Xiaoji Chen

Microsoft Research, US,
levnach@hotmail.com, cxiaoji@gmail.com,
Msagljs github home page: https://github.com/microsoft/msagljs

**Abstract.** There has been progress in visualization of large graphs recently. Still, interacting with a large graph in the browser with the same ease as browsing an online map, inspecting the high level structure and zooming to the lower details, is still an unsolved problem, in our opinion. In this paper we describe MSAGLJS's approach to two aspects of this problem. Firstly, we give a novel algorithm for edge routing, where the edges do not overlap the nodes. The algorithm does not necesserely creates optimal paths but is efficient and creates visually appealing paths. Secondly, to facilitate graph vizualization with DeckGL, namely fast zoom, and pan operations, and keeping the number of entities on the screen below a specific bound all the time, we use tiling. Our tiling procedure is simple and efficient. It is the second contribution of the paper.

## Introduction

## Related work

Links to large graph visualization

[1]

[2]

[3]

[4]

[5]

[6]

machine learing approach [7]

[8]

[9]

## Edge routing

The edge routing starts, as in [10], by building a spanner graph, an approximation of the full visibility graph. The spanner, see Fig. 2, is built on a variation of a

Yao graph, which was introduced independently by Flinchbaugh and Jones [11] and Yao [12]. A Yao graph is defined by the set of cones with the apices at the vertices. The cones have the same angle, usually in the form of $\frac{2\pi}{n}$, where $n$ is a natural number. This way the cones with the apex at a specific vertex partition the plane as illistrated in Fig. 1. For each cone at most one edge is created connecting the cone apex with a vertex inside of the cone.

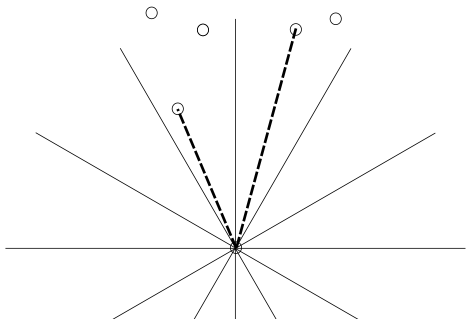

**Fig. 1.** Yao graph

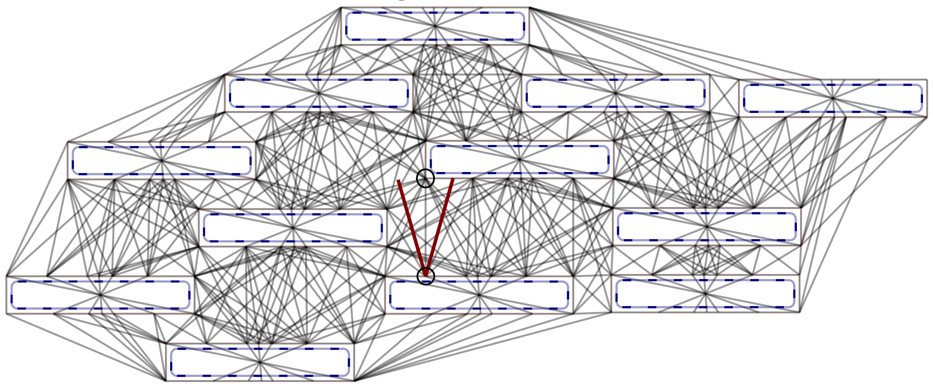

**Fig. 2.** Spanner graph is built using the idea of Yao graphs. The dashed curves are the original node boundaries. Each original curve is surrounded by a polygon with some offset to allow the polyline paths smoothing without intersecting the former.
The edge marked by the circles is created because the top vertex is inside of the cone and it is the closest among such vertices to the cone apex. The apex of the cone is the lower vertex of the edge.
MSAGLJS uses cone angle $\frac{\pi}{6}$, so the edges of the spanner can deviate from the optimal direction by this angle. Therefore the shortest paths on the spanner have length that is at most the optimal shortest length multiplied by $\frac{1}{\cos(\frac{\pi}{.}6})) \simeq 1.155$.

The approach of [10] first builds a polyline path through the spanner, and then applies some local modifications to shorten and smoothen the path. For shortening it tries to shortcut a vertex, as illustrated in Fig 3. To smoothen it

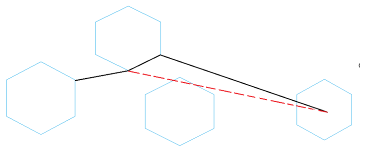
**Fig. 3.** Unsuccessful shortcut

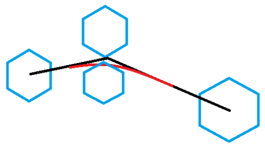
**Fig. 4.** Fitting a Bezier segment into a polyline corner

fits Bezier segment into the polyline corners, using the binary search to find the larger fitting segments,see Fig 4. While anylyzing performance of edge routing in MSAGLJS, we noticed that for a graph with more than 10000 edges these heuristics become the major bottleneck.

The reason for this was that we queured if a curve intersects any node of the whole graph. In spite of optimizing these operations with R-Trees [13], about %90 of the edge routing running time was spent on them. In addition, when the naive shortcutting of polyline corners fails the resulting path is not visually appealing, as shown in Fig. 3.

We replace global inefficent calculatitons with faster local procedures.

## Local path optimization

The idea of using a local optimization for paths is not a new one. The authors of [14] used it for hierarchical layouts, where the polygon containing the path is available. They write: "If $\mathcal{P}$ does not contain holes ... we can apply a standard "funnel" algorithm [15, 16] for finding Euclidean shortest paths in a simple polygon". We build polygon $\mathcal{P}$ containing $\mathcal{L}$ that is suitable for the funnel algorithm application mentioned above. Our method works for any layout. Let us describe it.

We call obstacles $\mathcal{O}$ the set of polygons covering the original nodes, see Fig. 2. Before routing edges we calculate a Constrained Delaunay Triangulation [17] on $\mathcal{O}$ and call it $\mathcal{T}$. Then for each edge of the graph we proceed with the following steps.

We route a path on the spanner, as illustrated by Fig. 5. Let us call this path $\mathcal{L}$.

Let $\mathcal{S}$ be the obstacle containing $\mathcal{L}$'s start point, and $\mathcal{E}$ be the obstacle containing $\mathcal{L}$'s end point. To obtain $\mathcal{P}$, let us consider $\mathcal{U}$ , the set of all triangles $t \in \mathcal{T}$ such that either $t \subset \mathcal{S} \cup \mathcal{E}$, or $t$ intersects $\mathcal{L}$ and is not inside of any obstacle. The union of $\mathcal{U}$ gives us $\mathcal{P}$. The boundary of $\mathcal{P}$ comprizes all edges $e$ of the triangles from $\mathcal{U}$ such that $e$ is adjacent to exactly one triangle from $\mathcal{U}$, see Fig. 6.
To apply the funnel algorithm [15, 16], we need to have a triangulation of $\mathcal{P}$ such

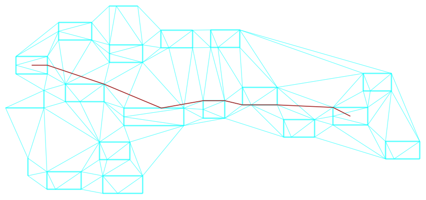
**Fig. 5.** Path $\mathcal{L}$ with $\mathcal{T}$, a fragment.

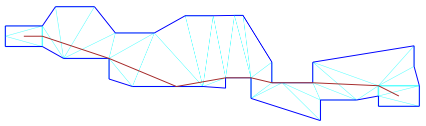
**Fig. 6.** Polygon $\mathcal{P}$ containing $\mathcal{L}$.

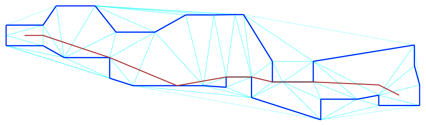
**Fig. 7.** New triangulation of $\mathcal{P}$.

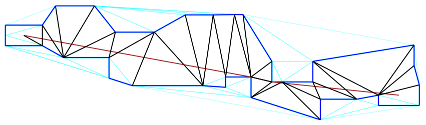
**Fig. 8.** Optimized path.

that every edge of the triangulation is either a boundary edge of $\mathcal{P}$, or a diagonal of $\mathcal{P}$. In our situation $\mathcal{U}$ does not have this property, containing triangles with vertices at the centers of $\mathcal{S}$ and $\mathcal{E}$. We create a new Constrained Delaunay Triangulation of $\mathcal{P}$, where the set of constrained edges is the boundary of $\mathcal{P}$, see Fig. 7.

Finally, we apply the funnel algorithm to obtain the path which is the shortest in the homotopy class of $\mathcal{L}$, see Fig. 8.

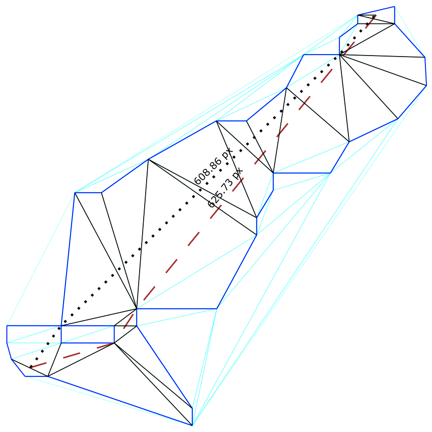

**Fig. 9.** $\mathcal{P}$ is not simple polygon. The dotted path is shorter than the dashed one, that was found be the routing.

The polygon $\mathcal{P}$ is not neccesserely simple, as shown in Fig. 9. In this example the path that we calculate with the funnel algorithm is not the shortest path inside of $\mathcal{P}$.

## Performance comparison

In Table 1 the time is presented in seconds.

| graph name | nodes | edges | old time | new time |
|---|---|---|---|---|
| social network | 407 | 2639 | 1.0 | 1.4 |
| b100 | 1463 | 5806 | 5.6 | 5.785 |
| composers | 3405 | 13832 | 510.5 | 17.5 |
| p2p-Gnutella04 | 10876 | 39994 | 375.4 | 293.8 |

## References

1. "Graphexp." https://github.com/bricaud/graphexp.
2. "Graphviz." http://www.graphviz.org/.
3. "Regraph." https://cambridge-intelligence.com/regraph/.
4. "Skewed." https://graph-tool.skewed.de.
5. "Circos." http://circos.ca/.
6. H. Gibson, J. Faith, and P. Vickers, "A survey of two-dimensional graph layout techniques for information visualisation," *Information visualization*, vol. 12, no. 3-4, pp. 324–357, 2013.
7. O.-H. Kwon, T. Crnovrsanin, and K.-L. Ma, "What would a graph look like in this layout? a machine learning approach to large graph visualization," *IEEE transactions on visualization and computer graphics*, vol. 24, no. 1, pp. 478–488, 2017.
8. Z. Lin, N. Cao, H. Tong, F. Wang, U. Kang, and D. H. Chau, "Interactive multi-resolution exploration of million node graphs," in *IEEE VIS*, 2013.
9. "Cosmograph." https://cosmograph.app.
10. T. Dwyer and L. Nachmanson, "Fast edge-routing for large graphs," in *Graph Drawing: 17th International Symposium, GD 2009, Chicago, IL, USA, September 22-25, 2009. Revised Papers 17*, pp. 147–158, Springer, 2010.
11. B. Flinchbaugh and L. Jones, "Strong connectivity in directional nearest-neighbor graphs," *SIAM Journal on Algebraic Discrete Methods*, vol. 2, no. 4, pp. 461–463, 1981.
12. A. C.-C. Yao, "On constructing minimum spanning trees in k-dimensional spaces and related problems," *SIAM Journal on Computing*, vol. 11, no. 4, pp. 721–736, 1982.
13. A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pp. 47–57, 1984.
14. D. P. Dobkin, E. R. Gansner, E. Koutsofios, and S. C. North, "Implementing a general-purpose edge router," in *Graph Drawing: 5th International Symposium, GD'97 Rome, Italy, September 18–20, 1997 Proceedings 5*, pp. 262–271, Springer, 1997.

15. B. Chazelle, "A theorem on polygon cutting with applications," in *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pp. 339–349, IEEE, 1982.
16. J. Hershberger and J. Snoeyink, "Computing minimum length paths of a given homotopy class," *Computational geometry*, vol. 4, no. 2, pp. 63–97, 1994.
17. B. Delaunay, "Sur la sphere vide, bull. acad. science ussr vii: Class," *Sci. Mat. Nat*, pp. 793–800, 1934.