Virtual Road Graph Generation

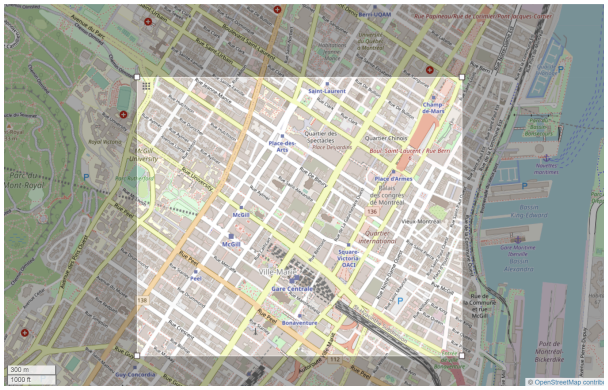
Cameron Angle & Oliver van Kaick

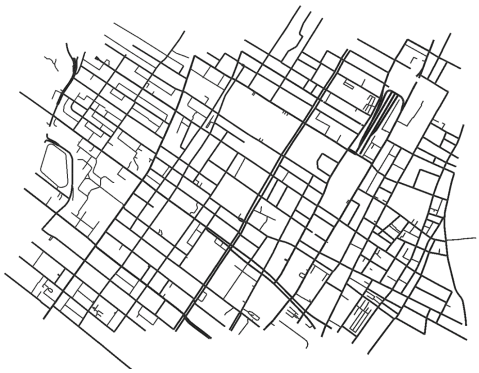
Abstract

In this project, we are developing a method to generate training data for a neural network capable of synthesizing realistic cities and road layout. Procedural generation of cities can be used in simulations and video games, and can also be used in an urban planning setting.

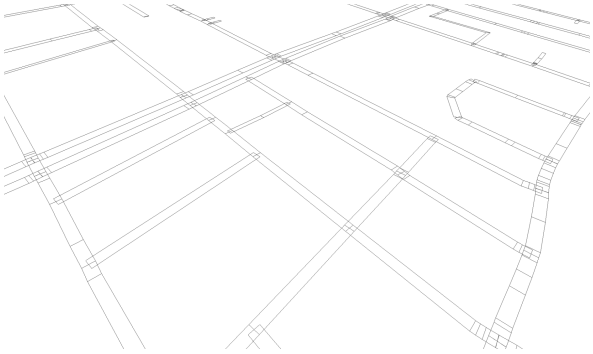
Pipeline Overview

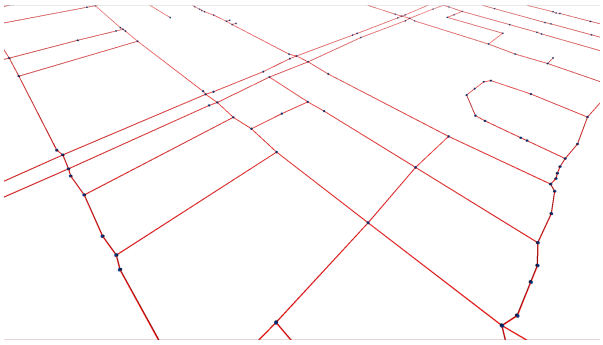
This script at a high level, is used to convert open street map data in .obj format to a graph format. The starting .obj file consists of various polygons making up a road network.





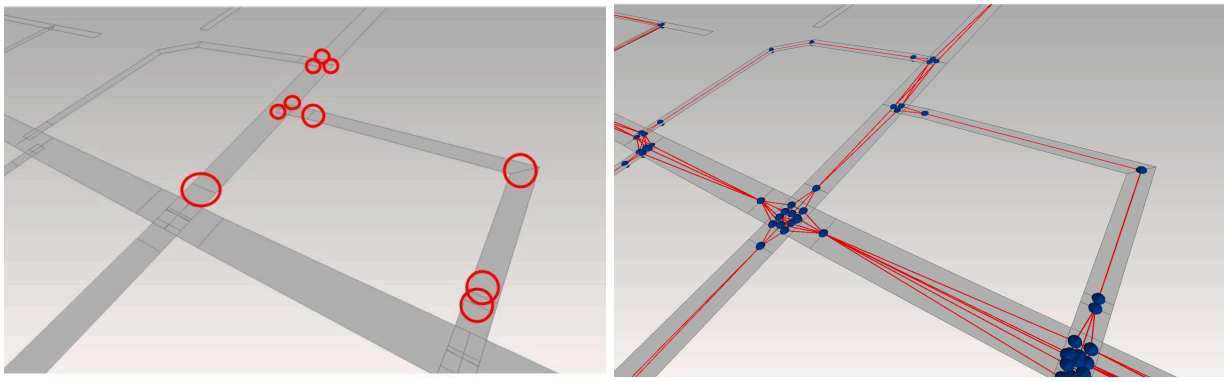
Polygon overlaps are detected, and through this nodes/connections are created to make a simpler graph representation. This graph will be used to train a neural network on road network creation.





Collision Detection

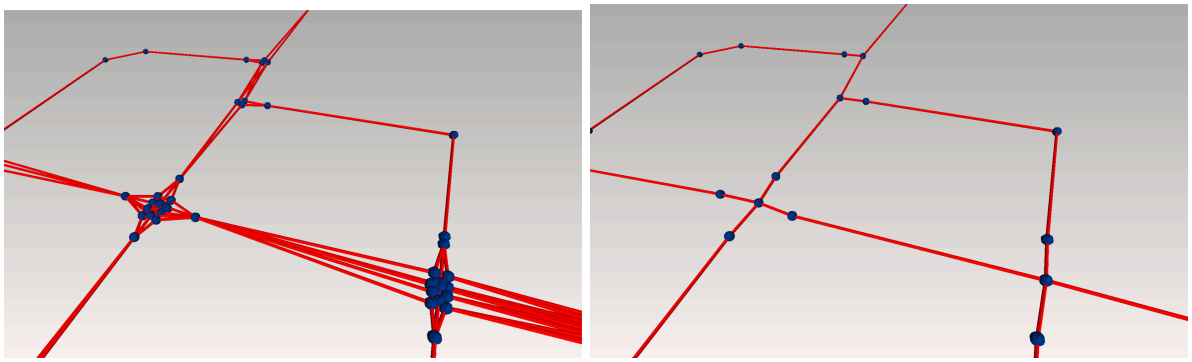
Polygon overlap and collision is detected using an R-tree data structure for bounding box connections, and the Shapely library for actual polygon connections. The main function is called “getGraphRtree()”, and it is passed the open street map .obj mesh obtained using the geomproc library.The function returns an adjacency list and a nodes list. Inside the function, a list of polygons are created and then fed into an R Tree data structure index, using the python Rtree library. Then, the tree structure returns all collisions of polygons in O(logn) time. This process is used a couple of times; firstly, in the data there are duplicate polygons that occupy the same space and have the same dimensions. These are removed in the “removeDuplicates()” function before any major processing takes place. Finally, collisions are detected in “computeConnections()”, where the majority of the graph is created.



Graph Creation

Inside “computeConnections()”, nodes in the graph are placed where two polygons connect/touch. A node “belongs” to exactly two faces/polygons in that sense, because it is created by the collision between them. Most polygons have around 2-3 nodes that belong to them, but this varies depending on if the polygon is near an intersection, where it will likely have 5-6. After all nodes have been created (avoiding duplicates), they are connected. Nodes are connected based on if they belong to the same polygon. So if nodes 1,2 belong to “face A” and nodes 2,3 belong to “face B”, then node 2 will connect to node 1 and node 3, since it will connect to every node that belongs to the same faces. In this case since it belongs to “face A” and “face B”, it connects to all nodes belonging to “face A” and “face B”.

Node Simplification



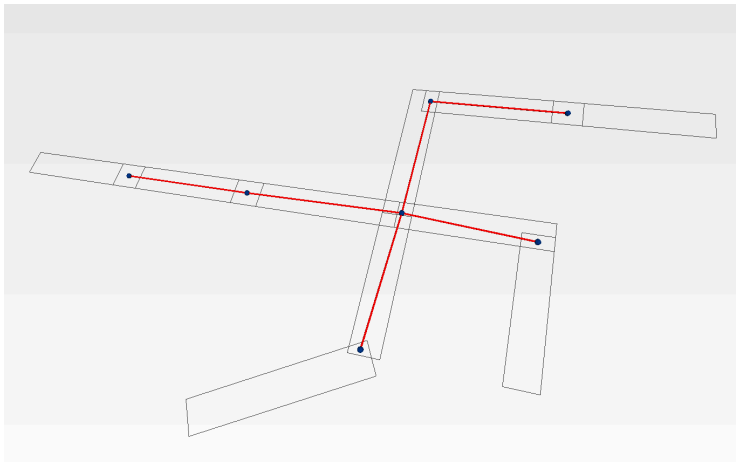
Particularly for traffic intersections, there's a lot of small separate faces that create a very messy look to the graph. The function “simplifyEdges()” takes all edges shorter than some minimum distance and simplifies them. Two nodes connected by a “short” edge will be merged into a single node in the middle of the previous two, where the first node in the connection moves to the midpoint and then the second node is deleted. All edges of the deleted node are rerouted to the first node, which now is in the middle of the previous short edge. Later, all instances of these “second nodes” are deleted from the adjacency list and nodes list.

Graph Visualization and Output

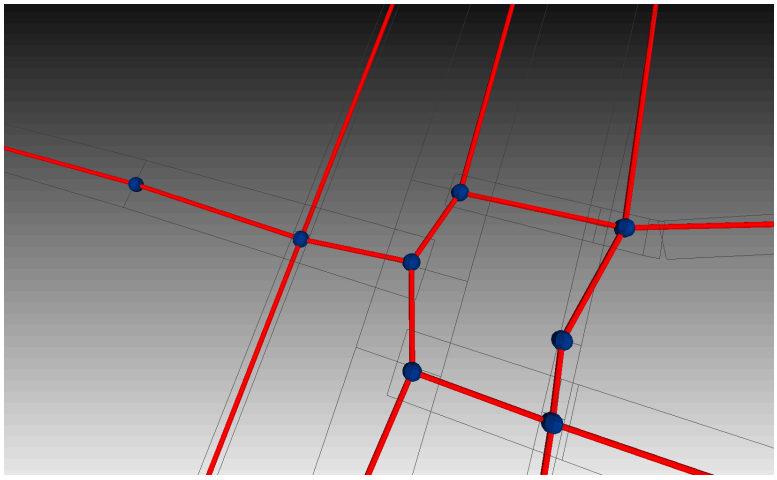
Using the geomproc library, the graph can be visualized with nodes and lines. The nodes being red, and lines/connections being blue. This is a useful tool to quickly determine errors or areas for improvement in the algorithm. Additionally, there is a more data oriented text output. First, the number of nodes is listed followed by all the nodes, where <Node> = <Number of vertices> <x1> <y1> <z1> … <xn> <yn> <zn>. Then the number of edges are listed, followed by all the edges, where <Edge> = <Index of node 1> <Index of node 2>. The program is set up to accept any number of files from folder “roadsToProcess”, and will output the .obj and text file to folder “processedRoads”. The folders need to be in the main directory with the geomproc library.

Future Work and Limitations

Overall, the algorithm struggles with certain edge cases and scenarios. For one, it cannot properly acknowledge dead end streets, since nodes are only created where streets intersect.

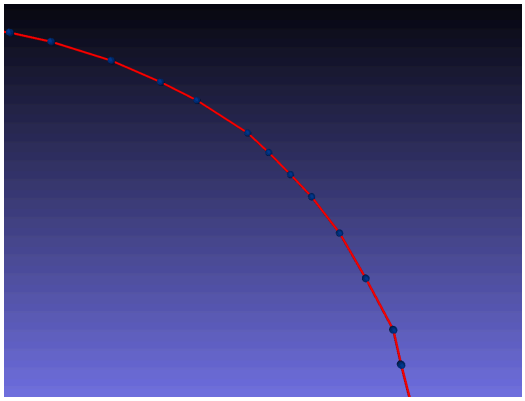


Another issue the program has is node placement in intersections. Often the program will place an “intersection node” off the center of a 4 way traffic intersection.



This can be accounted for by using a cluster algorithm to gather polygons in a traffic intersection cluster and accurately determine where the “visual middle” of the intersection is using area. There can also be more graph simplification where nodes can be deleted if there are only two edges connected to them, and those two edges are near parallel. Another issue is the amount

of node used in representing curves, and ideally this number could be brought down.



Overall there is still room for further graph simplification and better node placement/refinement.