IMPORTING DATA FROM SHAPEFILES & PATHFINDING ALONG GENERATED NODES

Victor Brestoiu
Grade 12, The Woodlands School, Peel District School Board (Mississauga, ON)

ABSTRACT:
The Shapefile format is a particular standard for storing GIS (Geographic Information System) data, designed and developed by the Environmental Systems Research Institute (ESRI). The purpose of this project was to extract the binary data describing the City of Lethbridge from ESRI Shapefiles, and then to demonstrate an ability to utilize and modify this data. The utilization component centered on pathfinding and visually drawing the data, while the modification component involved the creation of a new, human-readable file type which contained the processed Shapefile data. These goals were accomplished by converting the Shapefile data into custom ‘Node’ objects in C++ code. These nodes form the basis for further development, as more attributes can easily be added to them as needed. The implemented pathfinding is a matter of picking a starting and ending node, and travelling across their adjacent nodes until a shortest path is found, a search algorithm called A* (read: A Star). Although further work is necessary for a robust product, this platform is already highly modular and is freely available open source.

INTRODUCTION:
Transit networks are an integral part of most large cities, as they enable vast amounts of people to efficiently reach their destinations, as well as rapid transportation of crucial goods and services (Transportation Economics Organization). Since geographic data is critical to the operation of a transit network, being able to easily parse, visualize, and modify this data would be a massive boon to engineers working to optimize or extend these networks. The aim of this project, thus, was to build the foundational tools that would be required by such engineers, in a modular manner such that these utilities can either be used as-is for a very rudimentary but generally functional platform or developed into a more advanced and higher quality product to aide in future additions or modifications to the transit system.

There has been a significant amount of research in recent years on pathfinding, especially in computer games (Bulitko, 2011). Games, especially real time strategy games, are strong drivers of research in this field, as they require the implementation of an increasing numbers of AI agents with ever increasing complexity. Complexity is defined in terms of the number of variables that must be considered and the

1RTS Games: Real-time strategy games such as StarCraft, or Supreme Commander, etc.
2AI Agents: Artificial intelligence agents – computer controlled entities - in a sense each agent is a mini-computer that reacts to stimuli around them, making ‘intelligent’ choices
varieties of tasks that must be accomplished by the agent, and all at an increasing speed compared to previous generations of games and their respective agents. (Buro & Churchill, 2015).

A commonly used data structure for modeling pathways is called a graph (Bollobas, 1998). In this data structure, individual nodes make up a ‘web’ of connections, with each node concerned only about its immediate neighbours – a lot like intersections in a system of roads. Graphs have been studied since the advent of computers, and a fair amount of progress has been made in terms of how to optimally traverse a graph (Bollobas, 1998). Pathfinding algorithms, in the context of graphs, are a set of instructions for how the computer should maneuver this network of nodes (Pohl, 1970). The primary algorithm of concern here is A*, which is a heuristic variant of Dijkstra’s algorithm. (Khantanapoka & Chinnasarn, 2009) The premise behind Dijkstra’s algorithm is quite straightforward: the computer must calculate the cost of moving from the current node, to one of its neighbours. This calculation usually, but not always, represents the distance between the two nodes. A* is simply a slightly smarter process, in which the computer also rates a node’s neighbours on how far they are from the target (and not just how far they are from the source). (Khantanapoka & Chinnasarn, 2009) Because of this heuristic, A* is faster, and under some conditions, is guaranteed to be both complete (meaning it will find a path between two nodes if a path exists) and optimal (meaning that the path found will be the shortest possible path) (Chiang, 2011) (Zeng & Church, 2009). Another way of considering Dijkstra’s is as a special case of A*, whereby all nodes have the same heuristic cost (Zeng & Church, 2009).

MATERIALS AND METHODS

The entire project was completed digitally. The code was written in C++ and developed using Visual Studio 2015 Community Edition³, as well as SFML⁴ for the ability to draw the data to the screen. Before using the final data to complete the project, which was provided by Dr. Benkoczi, sample Shapefiles available on Statsilk⁵ were the primary sample used to test the functionality of the code.

Code was written in C++ to read the Shapefiles as described in ESRI’s documentation.⁶ Major obstacles were primarily related to the binary format of the file, as well as the minimal verification methods built into the files themselves. Custom (albeit basic) functions were created to import the data, with care being taken to follow the documentation with regards to large or small endianness⁷, as well as the proper double-precision floating-point format as described by the IEEE 754 standard⁸. The latter was particularly difficult – future attempts at doing this should take advantage of the operating system’s innate ability to handle doubles. The issue was eventually resolved by reading the required bytes and then casting the bytes as an IEEE 754 double-precision value through C++’s standard “memcpy” (memory copy) function, instead of trying to do so manually.

Furthermore, extra care was needed to validate all data being read from the file. Initially, this project was coded with the assumption that all data are valid – this is unreasonable, and must be accounted for, especially given that Shapefiles have bounding box data as one of the only validation measures in place. Approximately 10% of the data in any given file are invalid, almost always as a “NaN” value⁹, amongst the final data that were used for this project.

RESULTS

The extraction of the geographic Shapefile data was definitively accomplished. By drawing the acquired data and comparing with satellite images and Google Maps, it was clear that the data matched. However, not all of the data matched perfectly - in many cases, and for as of yet unknown reasons, some valid data was being discarded. Specifically, many buildings which should have been rectangular were being displayed as triangular, with three of the points identical, and

³ Visual Studio 2015 Community: A development environment for software created by Microsoft
⁴ SFML: Super-Fast Multimedia Library: An open-source platform for various multimedia written in and for C++
⁵ Statsilk: An online repository of free Shapefiles
⁶ ESRI Documentation: A PDF created by ESRI detailing how the Shapefile format works
⁷ Endianness: The principle of storing numbers in computers; either the most significant or least significant digits are stored first
⁸ Double-precision standard: Using 64 bits to store a number instead of 32; this results in much higher precision
⁹ NaN: Not a Number; used interchangeably with infinity
the fourth one missing. It is possible that some of the data had conflicting boundary boxes, or that the Shapefiles were not finalized, or that the component responsible for interpreting the Shapefiles had very niche bugs.

The export of custom ‘Nodes’ worked perfectly, with the program being capable of exporting Shapefiles into so-called “nodegraphs”, and then re-importing these nodegraphs without issue. A test nodegraph was also made, by hand, and imported flawlessly into the program, thereby demonstrating the custom file type’s human readability and compatibility with the program.

Finally, the pathfinding along the nodes was also tested, more extensively than the other features, and with promising results. The only problem at all was that under very specific circumstances, certain nodes would think they were adjacent with other nodes, though they clearly were not, resulting in paths that did not follow the streets represented by the nodes. This problem was not present when testing the nodegraphs instead of the Shapefiles, implying a very particular bug.

DISSCUSSION
To begin, the project achieved its first goal of being able to read the main Shapefile types used by the City of Lethbridge – essentially, Type 1, a point, Type 3, a line (two or more points), and Type 5, a polygon (a closed line). Using SFML’s built-in Vertex class to draw lines, it can draw pixel-perfect (or very close to it) representations of the Shapefile’s data. Depending on the user’s input, the program can generate a list of pathing nodes based on the input file. Pathing nodes fundamentally consist of a unique program-wide identifier, the node’s x and y coordinates, and a list of pointers to the node’s neighbours. They also contain additional metadata about the point such as its colour, but these are not relevant to the main premise of the project. The pathing nodes represent intersections between streets in the City, and are used to actually calculate the shortest path between any two points on the map.

Before continuing, it is important to mention a limitation of the program at this point: it assumes the integrity of the Shapefile’s data. Though it can compare coordinate point values with a Shapefile’s

Figure 1: The “Real World” View
Represented above is an urban area, with buildings demarked by the grey boxes, and roads with white spaces in between. The Red and Green dots represent the start and end positions respectively of the path finding operation.

Figure 2: Real World with Node Overlays
Represented above is the same urban area as in Figure 1, but the intersections are now represented as blue dots, and the connections between them as blue lines.
given bounding box, and will discard points that fall outside that box, or have NaN values for either of their coordinates, the program has no error checking or data interpolation to compensate for the discarded data. This effect is most commonly visible when triangular shapes are formed instead of boxes – the program connects the points it believes are adjacent, though it does not account for points that may have since been discarded.

Any Shapefiles of type 3 can currently have representative pathing nodes generated, as type 3 represents “PolyLine” geometry – simply a line made up of some number of segments. This type was chosen because the data for the City’s roads was in this format, and because the other formats do not lend themselves to pathfinding. Type 3 geometry nicely represents roads – a line from point A to point B, with possible intersections, and possible bends. On the other hand, type 1 represents “Point” geometry, and points evidently cannot be linked – if they were, the resultant geometry would be called a line. Type 5 represents “Polygon” geometry, which is similar to a “PolyLine,” but must always form a closed shape – a polygon. This geometry is not a problem for A*, but handling this case of files is not relevant, largely because roads are best represented by line segments rather than closed polygons.

Once a Shapefile is loaded into memory and converted into pathing nodes in a graph, the A* algorithm can be called to find the shortest distance between any two nodes in the graph.

The target points can be placed anywhere on the screen after which the algorithm will find the closest pathing nodes to the user’s desired start and end positions. Then, using A*, it rapidly sorts through the relevant nodes, and generates the shortest path between the input points. Throughout tests, on a laptop computer running Windows 8.1, the longest runtime was under 30 seconds to find a path from one corner of Lethbridge to the other. After a few tests on other machines, it seems that for an ‘average’ path across...
the City’s diagonal, not only will a path be found in less than 30 seconds, but it will in fact be the optimal path (so long as all data are valid), since it meets the criteria needed for A* to be admissible (Chiang, 2011). This is an important distinction because some amount of the data loaded from the Shapefiles is discarded. With the tested files, an average of 3-5% of points were discarded. In some instances, this relatively small percentage of discarded data resulted in missed connections – roads that should have been connected to one another but were not, or roads that clearly should not be connected, but are.

The last significant flaw in the code is that occasionally the path found is too good – the path is in fact the shortest path, but following it would be a traffic violation. This occurs most prominently with highway on and off ramps, as highways tend to be encoded as multiple parallel streets in the tested data. This, combined with the current method of merging nodes within 1 meter of one another, results in A* on occasion getting onto the highway and then making a sharp (illegal) U-turn to reach a point on the other side of the highway.

However, the program has two significant features that annul these flaws. The first is that the interpreted data can be saved to a custom file type that contains the information needed to recreate the connection between nodes. This file is human readable, and thus modifiable with relative ease. This has enabled a testing of A*, which has demonstrated that it does, in fact, always find the optimal path between any two given nodes on the graph, if such a path exists. The other significant feature is an extension of the former – the program provides a modifiable platform for converting Shapefile data to data of just about any other type, by proxy of this custom format. Currently, only the named custom format is built-in, but extensions to CSV, XML, or JSON files are easily possible should they be needed.

CONCLUSION

The targeted goal of interpreting ESRI Shapefiles and managing the resultant data has been achieved. Although the final code base is fairly robust, there are still some flaws and room for further growth. For example, one improvement that would have been quite useful to include is the functionality of ‘points of interest’, whereby the algorithm attempts to find the shortest path between points A and B by way of points C, D, E, etc. This sort of service would make it so city planners can place and manipulate individual bus stations while still seeing the ‘big picture’ of the whole route. It could also help route buses through actual points of interest in the city, should the bus route planners wish to do so. Overall, this is still doable by manually going through each combination, but it would evidently be preferable if the computer could do it automatically.

Furthermore, by merging what has been done in this project with more advanced concepts such as flow fields (Blum & Li, 2008), it may be possible to provide a service not just to bus route planners, but also to urban developers and planners as well, enabling them to have a clearer, visual, understanding of how people will move within and use the city.

ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESRI</td>
<td>Environment Systems Research Institute</td>
</tr>
<tr>
<td>GIS</td>
<td>Geographic Information System</td>
</tr>
<tr>
<td>SFML</td>
<td>Super-Fast Multimedia Library</td>
</tr>
<tr>
<td>A*</td>
<td>Pronounced “A star.” Refers to a computer algorithm</td>
</tr>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>Nan/NaN</td>
<td>Not a number/ not a numerical value</td>
</tr>
</tbody>
</table>

ACKNOWLEDGEMENTS

To begin, I would like to extend a major thank you to Dr. Robert Benkoczi, without whom this project would not have been possible, as well as the City of Lethbridge, and the University of Lethbridge for access to the data for which this project was made. Further thank you to Ms. Lauren Sykes who coordinated this program, and Ms. Liz McBryan who provided guidance and support. Additional acknowledgement goes to SFML (and its community), created by Laurent Gomila, and made available with the zlib/png license.
REFERENCES


