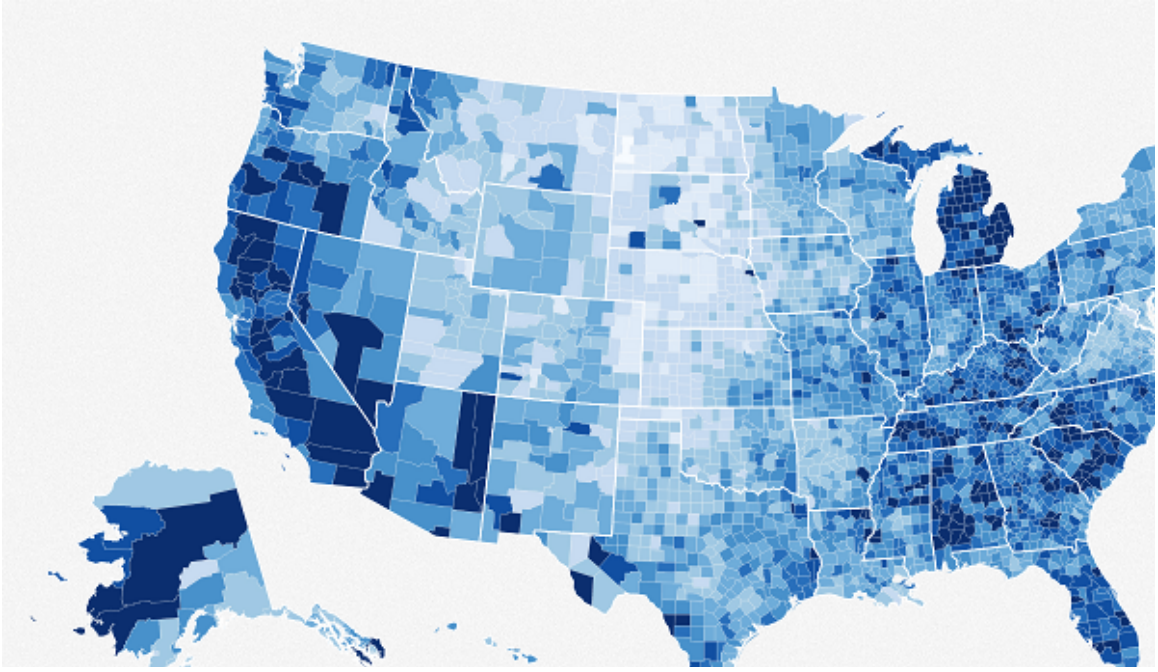
Quadstream

Algorithms for Multi-scale Polygon Generalization

Curran Kelleher
December 5, 2012

Introduction

The goal of this project is to enable the implementation of highly interactive choropleth maps on the Web that support smooth zooming and panning with variable resolution. Choropleth Maps, also known as "Thematic Maps", assign color to geographic regions based on some data. Numerous visualization libraries and toolkits support creation of choropleth maps. For example, here is a choropleth map created using D3:



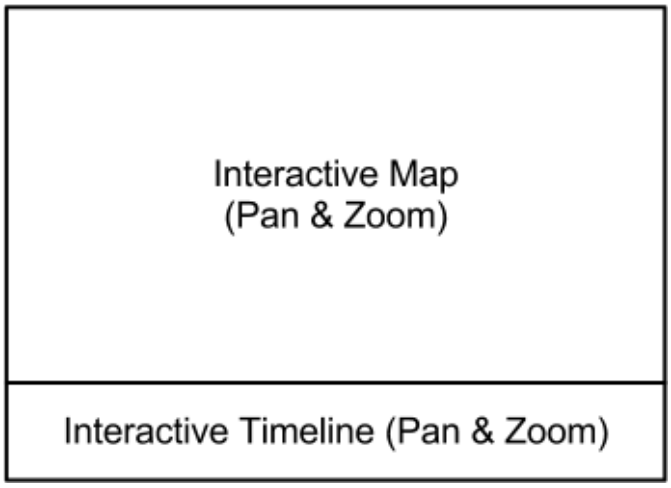
The target data to be visualized includes public data sets that have

- many various measures that could be visualized using color
- temporal variance (e.g. yearly data), and
- hierarchical structure (e.g. Continents, Countries, States).

The desired choropleth map therefore must have the following properties:

- instant response when the user changes either
  - the measure to visualize
  - the color map used
  - the time slice viewed
- support for zooming and panning to explore the hierarchy

Here is a sketch of what the user interface might look like for time navigation:



Examples of public data available include:

- CO2 Emissions Per Capita
- Energy Consumption Per Capita
- Life Expectancy

When developing choropleth maps to run in a Web Browser,

one must consider bandwidth limitations. To keep download sizes small, existing solutions typically either use a fixed polygon generalization, or image tiles rendered on the server side. Fixed generalization leads to rough looking polygon boundaries when zooming in. Use of server-side image tiles demands a round trip to the server for re-rendering. Our desired choropleth map tool cannot use fixed generalization, as we require high quality polygon boundaries at any zoom level. The image tile approach cannot be used user interactions that cause re-rendering must be instant, and not require a round trip to the server.

The contribution of this project is a set of algorithms and data structures for building client-side rendered choropleth maps with multi-scale polygon generalization. This approach fulfills all the requirements for our choropleth map application, and can be re-used by researchers and software developers looking for a solution for creating a highly performant interactive Web-based choropleth map with smooth zooming and panning and support for hierarchical polygons.

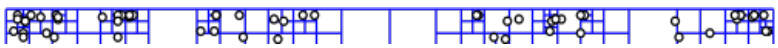
Related Work

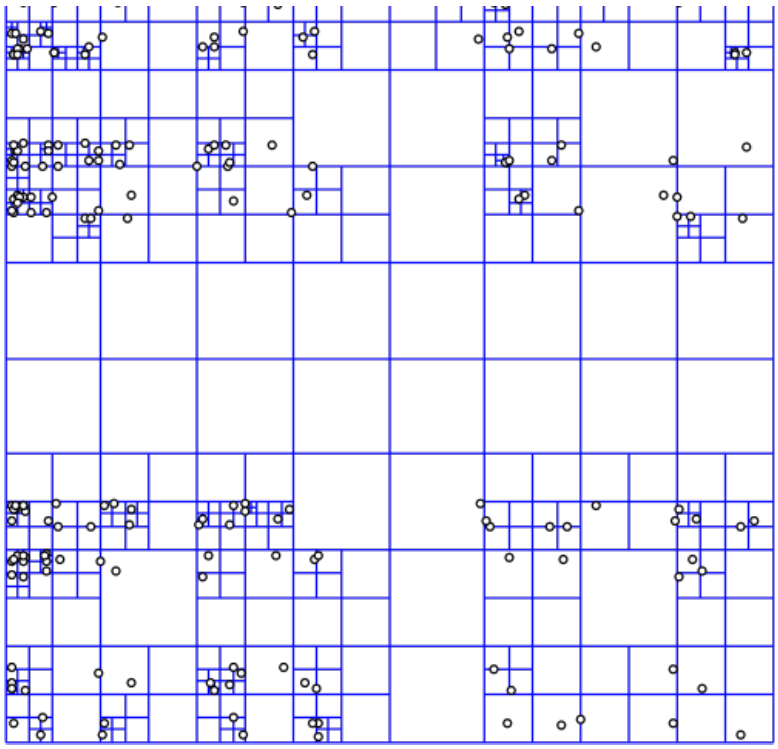
In this section prior work is surveyed in the areas of multi-scale structures, line and polygon generalization algorithms, and web-based choropleth map visualization tools.

Multi-Scale Structures

Multi-scale data structures and algorithms have been devised to cope with data at multiple scales (i.e. varying "zoom level" or "level of detail"). These approaches often involve recursive partitioning of a space into a data structure that can be efficiently queried. Multi-scale structures related to this work include the Quadtree, R-Tree, Reactive Tree, and BLG Tree.

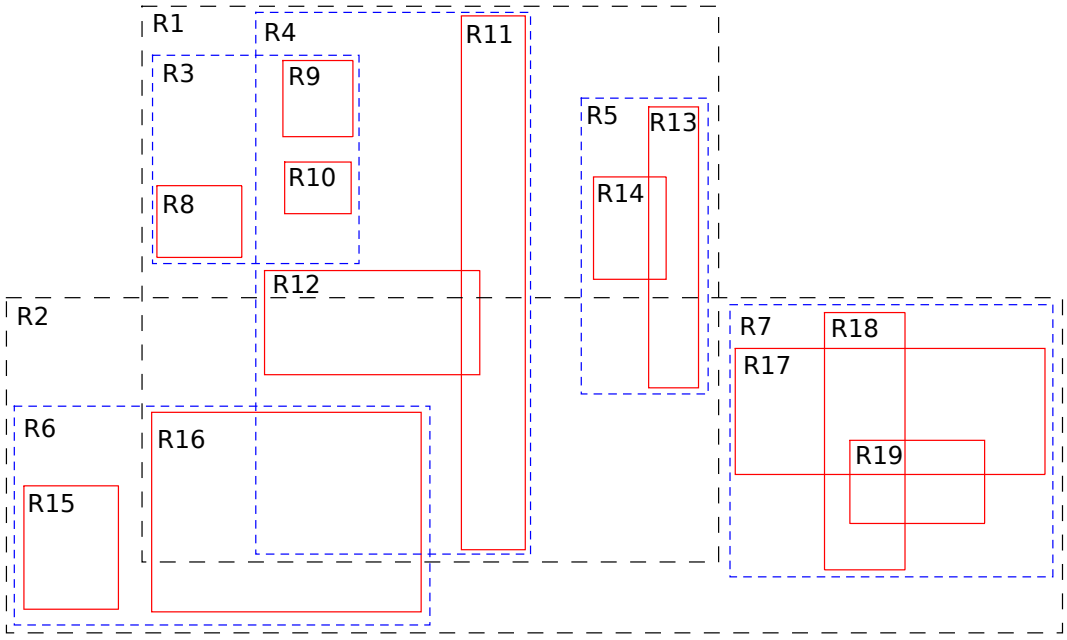
The quadtree data structure recursively subdivides a bounding box into four quadrants, or "buckets". For a point quadtree, each bucket is subdivided as long as it contains above a certain threshold number of vertices.

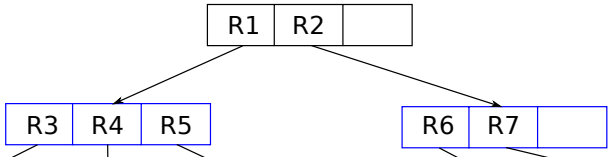


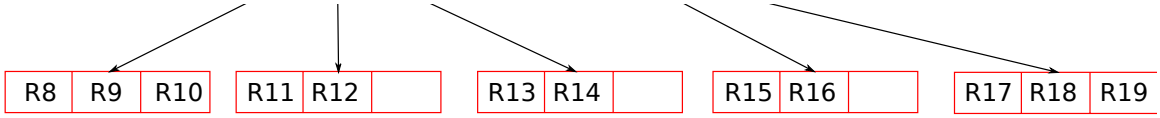


An example point quadtree, from Wikipedia.

The R-Tree data structure is a rectangle-based tree structure designed to support efficient rectangle intersection or containment testing.



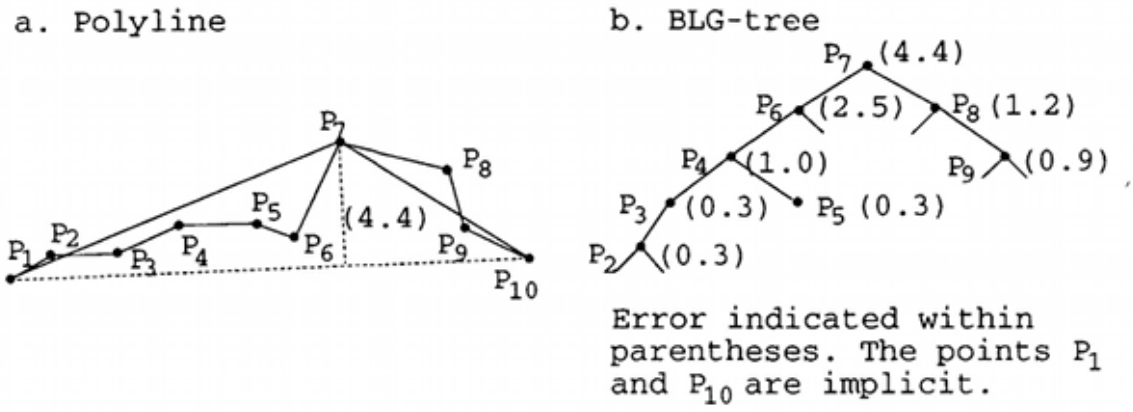




An example R-Tree, from Wikipedia.

The Reactive Tree data structure aims to provide a multi-scale solution for dynamic generalization using simplification (removal of vertices in lines and polygons), aggregation (summarizing many features with fewer), symbolization (representing polygons with lines or points), and selection (omission of features).

The Binary Line Generalization Tree (BLG-Tree) is one component of the Reactive Tree that supports efficient rendering of pre-computed line generalizations at multiple scales. The BLG-Tree is a binary tree in which an inorder traversal yields the original ordering of the vertices, and the depth of each node corresponds to its "importance level", a value determining at which scale the vertex should be present. To render a polygon or line at a given resolution, thr tree is traversed only to the depth required to reach the importance threshold determined by the scale of the viewing region.



An example BLG-Tree shown with its corresponding line, simplified using the Douglas-Peucker algorithm. From the Reactive Tree Paper.

Line Generalization Algorithms

Line generalization refers to the process of modifying high resolution lines or polygons for presentation at a lower resolution. Some line generalization algorithms function by choosing original vertices to "keep", and others introduce

vertices that were not there in the original data. The two most relevant line generalization algorithms for this work are the Douglas-Peucker Algorithm and the Li-Openshaw Algorithm.

The Douglas-Peucker algorithm appears to be the most widely used algorithm for line generalization. It works by recursively subdividing the list of vertices in the line until the "error" (perpendicular distance) is below a certain threshold value.

The Li-Openshaw Algorithm works by imposing a raster grid over the line and selecting vertices such that each cell of the raster grid that contains vertices is represented by a single vertex. There are several variations of this algorithm; one that introduces points not in the original data (such as the cell center, or average between enter and exit points), and one that uses the original vertices.

Web-based Map Visualizations

Many Web-based choropleth map tools exist today. Most, if not all, of them use either fixed line generalization (most likely using the Douglas-Peucker algorithm) or use an image tile approach in which images are rendered on the server side, cached, and delivered to clients when needed.

Here is a selection of several tools:

- GapMinder - Though not a choropleth map, this tool is a widely known example of Web-based map visualization.
- Tableau Public - This tool supports choropleth maps with panning and zooming, in conjunction with other visualizations, and uses a server-side-rendering approach.
- D3 - This is a library for implementing client-side rendered visualizations, including choropleth maps, using Scalable Vector Graphics (SVG). D3 choropleth maps use a fixed polygon generalization.
- CNN Economy Tracker - This visualization tool for economic data uses a fixed generalization and is a perfect example where zooming and panning would be useful (as the data is present for US State and US County levels, but not presented in the tool).
- Weave - This project supports dynamic resolution choropleth maps with zooming and panning, but the

algorithms involved have not been published.

Quadstream

Our solution, which we call "Quadstream", involves the combination of the Quadtree recursive subdivision pattern with the Li-Openshaw line generalization algorithm. The result can be partitioned into small files. The client can consume these files as they are neede to fill in detail on demand, as the user zooms and pans.

The Quadstream algorithm works in two phases; the publishing phase and the consumption phase.

Publishing Phase

The publishing phase is the process by which original input files (e.g. ESRI Shapefiles or GeoJSON files) are transformed into a set of files that can be published to the Web and consumed by Quadstream clients. This is a one-time process for each shape set to be published.

The algorithms for the publishing phase do the following:

1. Preprocess vertices - input polygons are assigned integer identifiers, and for each polygon, vertices are assigned integer identifiers that define their winding order for rendering. Vertices that fall on the same (x, y) coordinates are aggregated. The resulting data structure is a collection of vertex objects with the following properties:
    - (x,y) coordinates
    - memberships, a list of objects with
        - polygonId
        - vertexId
2. Build the quadtree - each vertex is assigned to a node in the quadtree using a multi-scale variant of the Li-Openshaw algorithm. After this phase, a quadtree is available in which each occupied node contains a single vertex. This quadtree can be traversed to a limited depth to compute the Li-Openshaw generalization for a specified scale. The vertices in this quadtree can be inserted into a BLG-Tree in which their importance

value corresponds to their level in the Quadtree.

3. Partition into files - to partition the vertices into files, a transformation of the quadtree is computed in which the vertices are moved several levels up in the tree (and vertices at the root stay at the root). Let us call the number of levels vertices are moved up `fileDepth`. The effect of this transformation is that all the vertices present in the subtree`fileDepth` levels down from the root all end up in the root node, and subnodes contain the leaf nodes of their original subtree down `fileDepth` levels. Each node in the transformed tree is output as a file containing an array of vertex objects, named according to its (level, i, j) address in the tree. The value of `fileDepth` can be tweaked to optimize the size of the files for best performance.

The algorithm that computes the quadtree is the main novel contribution of this work, and can be summarized with the following pseudocode.

```
N = makeHashMap()
for each vertex v in V
  for each level l in [0 ... L]
    k = key(v, l)
    if N does not contain k as a key
      put(N, k, v)
      break out of inner loop
```

In this pseudocode,

- N stands for "Nodes" of the quadtree,
- V stands for the preprocessed "Vertices", and
- L stands for the maximum "Level" for quad subdivision,
- `makeHashMap()` creates a hash table, and
- `key(vertex, level)` computes the address of the quadtree node at the given `level` that the given `vertex` falls into. This address is of the form (level, i, j) where i and j define the integral grid coordinates of the quadtree partitioning.

The running time of this algorithm is worst case O(n * L), because each vertex is visited once, and at most all levels between 0 and L are tested for each vertex. As L is a constant, the analysis simplifies to O(n).

Consumption Phase

The consumption phase is the phase of the Quadstream system that executes when a Browser-residend Quadstream client reads the files created in the publishing phase from the server and presents an interactive map with zooming and panning to the end user.

In the client, an R-Tree is initialized that will contain the bounding rectangles of the polygons. Each polygon is represented in the client using a BLG-Tree. Initially, the root file is downloaded, which contains a coarse generalization of all polygons (and their bounding boxes, so the R-Tree can be initialized). As the user zooms and pans, the files containing the vertices that should be rendered for the changing view rectangle are downloaded and inserted into their corresponding BLG trees.

Our goal is to achieve a rendering cycle that can execute at 60 Frames per second to support smoothly animated zooming and panning. Each rendering cycle, the following steps occur:

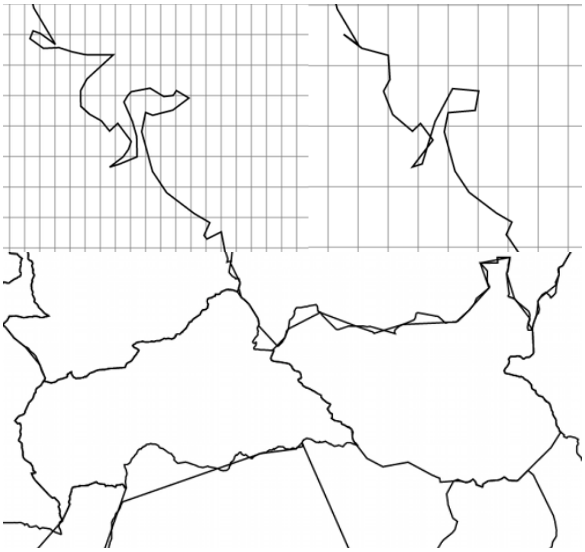
- The R-Tree is queried to find the list of polygons that are
    - partially or totally inside the viewing rectangle, and
    - large enough to see.
- For each polygon in the resulting list, its corresponding BLG Tree is traversed to a limited depth to find the vertices that define the polygon at the appropriate scale defined by the viewing rectangle.
- Each list of vertices is intersected with the viewing rectangle to eliminate vertices outside the viewing rectangle.
- Some data value is looked up for each polygon, and
- each polygon is rendered to the display using HTML5 Canvas.

Issues

The above sections articulate the ideal implementation, however the actual implementation fails to address the following:

- Partitioning across multiple files (the partitioning is computed, but in memory)
- Use of an R-Tree to index polygons, and
- View rectangle intersection computation.

The algorithm itself has two severe issues; it creates non-simple polygons in some cases, and it fails to handle the case when two polygons share a border but the vertices to not match up exactly.



Examples of where the algorithm delivers poor results: non-simple polygons, and borders that do not match exactly.

Future Work

In terms of the implementation, future work includes using multiple files, using an R-Tree, and computing the intersection with the viewing rectnagle. Additional desired features include the ability to work over a hierarchy of polygons (e.g. Countries, States, Counties) and use of a 3D sphere projection.

This project is part of a larger project, the Universal Data Cube Visualization System, which aims to provide a public resource pool with, Data Sets, Visualization Authoring Tools, and Visualizations. This system uses Semantic Web standards such as the RDF Data Cube Vocabulary to represent data.

Future work includes integrating the Quadstream system with the UDC data representation system such that public data exposed using the UDC framework will be automatically

made browsable using Quadstream-based choropleth maps. This involves the assignment of RDF URIs to Quadstream polygons as identifiers, and matching of identifers occuring in the UDC data with identifiers used for polygons.

Ideally a distributed Web-based ecosystem of developers and users can evolve in which users and developers publish their simplified polygons to the Web, and others can access them. Eventually, a system can evolve that contains all boundaries of Continents, Countries, States, Counties (and international equivalent), Cities, and Towns.

This hierarchy can serve as a backbone for assembling a collection of all useful public data, including statistics for health, economics, education, pollution, industry, and many more. The resulting tool would function as a telescope into the world through the lens of data and interactive graphics, and could be used by anyone with an Internet connection. It is our hope that this tool stands to revolutionize education, journalism, and public policy processes.

Links:

- Journal of Progress
- Mid-project Presentation 1
- Mid-project Presentation 2
- Final Presentation
- Pseudocode
- Circle Generalization Demo with Addresses (source)
- Quadtree Subdivision Demo (source)
- Pan Zoom Demo with Test Data (source)
- World Map Demo (source, documentation)
- 3D Earth Demo (source)
- Natural Earth
    - Raster Data
    - Country Polygons
        - Originally an ESRI Shapefile
        - Converted to GeoJSON with GDAL
            - Resulting file ~25MB.

References

- Li, Z. "Algorithmic Foundataion of Multi-Scale Spatial Representation". CRC Press. 2007.

- Samet, S. "Foundations of Multidimensional and Metric Data Structures". Morgan Kaufmann Publishers. 2006.
- Samet, S. "Applications of Spatial Data Structures". Addison Wessley Publishing. 1990.
- Wikipedia: R-Tree Quadtree KD-Tree Douglas-Peuker Algorithm
- Larger project: the UDCViS