Using Tcl to curate OpenStreetMap

Kevin B. Kenny
5 November 2019

The massive OpenStreetMap project, which aims to crowd-source a detailed map of the entire Earth, occasionally benefits from the import of public-domain data, usually from one or another government. Tcl, used with only a handful of extensions to orchestrate a large suite of external tools, has proven to be a valuable framework in carrying out the complex tasks involved in such an import. This paper presents a sample workflow of several such imports and how Tcl enables it.

Introduction

OpenStreetMap (https://www.openstreetmap.org/) is an ambitious project to use crowdsourcing, or the open-source model of development, to map the entire world in detail. In effect, OpenStreetMap aims to be to the atlas what Wikipedia is to the encyclopaedia.

Project contributors (who call themselves, “mappers,” in preference to any more formal term like “surveyors”) use tools that work with established programming interfaces to edit a database with a radically simple structure and map the features of interest to them. Most work with some combination of field survey (often simply done by visiting the features of interest on foot or bicycle while logging tracks on a GPS-enabled device such as a smartphone), tracing features from publicly-available aerial imagery, and editing the attributes of objects based simply on field notes and local knowledge.

One controversial approach to editing OpenStreetMap is to import data into the database from other sources. The reason for the controversy is that this approach is far and away the most difficult of the methods discussed here. It is fraught with pitfalls. These include the license compatibility of the data (does OpenStreetmap, and its users, actaully have binding and permanent permission to use and share the data?), the data quality (many government data sets actually contain enough inaccurate and downright incorrect data that the value added could well be negative), and the problem of ‘conflation.’ The last issue refers to the fact that imported data must be respectful of the mapper who has acquired and edited the same feature directly. At the very least, the imported features must be combined and reconciled with what has been previously mapped. In some cases, the only acceptable approach is to avoid importing the colliding object altogether.

This paper discusses some case studies in using Tcl scripts to manage the task of data import, including data format conversion, managing of the relatively easy data integrity issues such as topological inconsistency, identifying objects for conflation, and applying the changes. In many ways, it gets back to the roots of Tcl. There is no ‘programming in the large’ to be done here. The scripts are no more than a few hundred lines, and all the intensive calculation and data management is done in an existing ecosystem of tools. No Tcl extensions were developed in the course of the work described. Nevertheless, Tcl proves invaluable in helping this one mapper to keep his sanity through the import of several large, government-supplied data sets.

Background

The data model

As mentioned before, OpenStreetMap’s data model is radically simple. There are only three types of feature: *nodes*, *ways*, and *relations*.

A node represents a point on the surface of the Earth. It has a latitude and a longitude.

A way represents a linear feature. It comprises an ordered sequence of nodes. It is possible for the same node to appear multiple times in the sequence; this is generally regarded as poor practice, except that repeating the same node for the start and end of a way can be used to represent an area feature (a polygon).

A relation is used for anything more complicated than a node or way. It comprises an ordered sequence of nodes and/or ways. As with nodes within ways, the members within relations may appear more than once in the sequence. In addition, a node or way in a relation may optionally have a *role*. The role is simply a name; the meaning of a role is by convention.

One important type of relation is the *multipolygon*, which may contain only ways. The ways have the roles *inner* and *outer*, indicating whether they represent the outer edge of an area or the edge of a hole in the area. The relation is therefore expected to consist of one or more contiguous strings of outer ways, and zero or more contiguous strings of inner ones, with no crossings in the ways. Area features of arbitrary topology may therefore be referenced.

Each of the three types of object also has an unordered collection of *tags*. A tag is merely a pair of strings; a keyword and a value. Again, the interpretation of tags is a matter of convention. The 'correct' use of tags can be quite contentious; there are several mailing lists devoted to tagging strategy, which all tend to have endless arguments on them. A typical set of tags on a roadway might be:

```
highway=residential
lanes=2
maxspeed=50
name=Garden Drive
surface=asphalt
```

All the tags remain human readable but are drawn from a limited vocabulary so as to be of use to automated data consumers such as map renderers and navigation systems.

The ecosystem of tools

OpenStreetMap has a rich ecosystem of software tools devoted to its maintenance. There is some history of developing these in Tcl. For instance, Andrey Shadura (who at the time favoured the transliteration, Andrew Shadoura) carried out a 2010 project with sponsorship from Google Summer of Code to develop an editor for OpenStreetMap objects in Tcl/Tk [KUPR10]. This tool was released, but ultimately proved to be unpopular and was abandoned.

Essentially, the only problem with it appears to have been that embedding such a tool in Tcl does not provide enough of an advantage to make it compelling. A fully functional OpenStreetMap editor is a massive project, dealing with many outside file formats, sources of imaging, and plugins that provide specfic families of operations (such as editing buildings, managing street addresses, or mapping traffic turn restrictions).

Moreover, most of the editors actually host embedded Web servers, supporting a limited number of GET operations that allow panning and zooming the map, commanding a download from the OpenStreetMap server, requesting that the editor read data from a given URL (including file:// URL's, allowing the loading of local data), and so on. This interface proved adequate for most of the import tasks that were attempted.

Perhaps surprisingly, no attempt was made to represent geometry in Tcl at all. If Tcl were to contain the geometry, then Tcl would have to work with all the complexity of problems like "compute a multipolygon representing the intersection of these other multipolygons", "dilate this multipolygon by five metres", "compute the minimum set of node reorderings needed to give this multipolygon a consistent topology", and so on. Such a task would have to embed a computational geometry library such as GDAL (Geospatial Data Abstration Library – https://gdal.org/). Rather than develop such an extension, there was a relatively simply approach: store all the geometry in tables in a PostgreSQL database using the PostGIS extension (https://postgis.net/) and carry out all the geometry calculations there.

Making this decision opened up an entire Swiss Army Knife of tools for manipulating geographic data. Imported data, in practically any reasonable file format, could be loaded with the `ogr2ogr` command line tool, and data in the database could be reconverted to OpenStreetMap format by the existing Python script `ogr2osm.py`.

This decision also enabled the possibility of using a customary set of OpenStreetMap tools, `osmosis` and `osm2pgsql`, to maintain a local copy of a large subset of the OpenStreetMap database. The current setup keeps the complete data set for the North American continent, and updates it daily. With this data set at hand, arbitrary queries against a (possibly slightly stale) copy of OpenStreetMap are possible, a boon for determining the data relative to conflation.

So, what does this mean for Tcl?

All of these decisions are consistent with what has been traditionally considered to be The Tcl Way. Tcl was born as

a “glue” language for the purpose of integrating other tools, and it does that job well. Only a small number of Tcl extensions have been proven to be useful, and these do multiple duty:

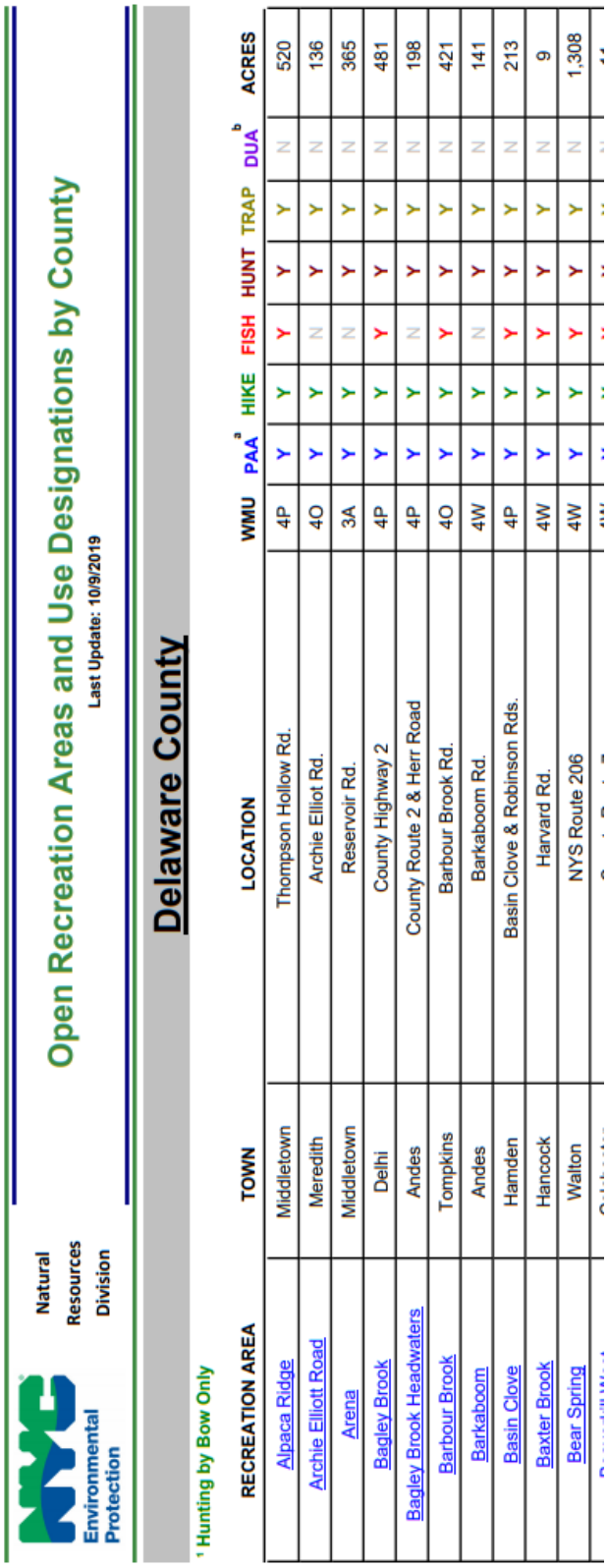
- `http`. More and more software functions by being commanded over a Web interface. As mentioned above, the popular OpenStreetMap editors are among these. Of course, `http` (together with `tls`) is still needed to retrieve the external data to be imported!
- `tdom`. Parsing and forming data in HTML and other XML dialects is again essential to modern interoperability.
- `tdbc::postgres`. As mentioned above, all the geometric heavy lifting is delegated to PostGIS.
- `clock:rfc2822`. This one turns up surprisingly often, in asking one server or another, “are my data up to date?”

Beyond these, the most powerful command that is used is undoubtedly `[exec]`. Many of the external systems have versatile command-line interfaces that are both fast enough and powerful enough to get the job done.

In short, the ideas presented here very much follow the principle of “let someone else do the heavy lifting!”

A typical workflow

The power of this minimalist approach to Tcl development is perhaps best illustrated by following the workflow of one typical import. In this case, the external data set that is to be brought in are the boundaries of some four hundred areas, of sizes ranging from a few thousand square metres to a few tens of square km, that are maintained by New York City outside its city limits to protect them from development (and therefore from polluting the watershed that supplies its drinking water). Since these areas are well-marked, and open to the public for recreations like hiking and bird-watching (and often for hunting, fishing and trapping as well), it seemed a good idea to map them in OpenStreetMap.



NYC Environmental Protection — Natural Resources Division

Open Recreation Areas and Use Designations by County

Last Update: 10/9/2019

Delaware County

[1] Hunting by Bow Only

RECREATION AREA	TOWN	LOCATION	WMU	PAA[a]	HIKE	FISH	HUNT	TRAP	DUA[b]	ACRES
Alpaca Ridge	Middletown	Thompson Hollow Rd.	4P	Y	Y	Y	Y	Y	N	520
Archie Elliott Road	Meredith	Archie Elliot Rd.	4O	Y	Y	N	Y	Y	N	136
Arena	Middletown	Reservoir Rd.	3A	Y	Y	N	Y	Y	N	365
Bagley Brook	Delhi	County Highway 2	4P	Y	Y	Y	Y	Y	N	481
Bagley Brook Headwaters	Andes	County Route 2 & Herr Road	4P	Y	Y	N	Y	Y	N	198
Barbour Brook	Tompkins	Barbour Brook Rd.	4O	Y	Y	Y	Y	Y	N	421
Barkaboom	Andes	Barkaboom Rd.	4W	Y	Y	N	Y	Y	N	141
Basin Clove	Hamden	Basin Clove & Robinson Rds.	4P	Y	Y	Y	Y	Y	N	213
Baxter Brook	Hancock	Harvard Rd.	4W	Y	Y	Y	Y	Y	N	9
Bear Spring	Walton	NYS Route 206	4W	Y	Y	Y	Y	Y	N	1,308

Figure 1. The index to the recreation area maps

Data intake

Once the legalities were sorted out (easy in this case owing to the fact that New York City has a strong "open government" law), the first issue to confront development was the inconvenient format in which the data were available.[1] The areas themselves were listed in a PDF file (Figure 1), and the file contained embedded hyperlinks, each of which designated another PDF file containing a map of one of the areas. There was clearly a serious job of "Web scraping" ahead!

The first task of the import was therefore to use `http` and `tls` to fetch this file to the local filesystem. Then, the command line tool, `httptohtml`, was used to make an HTML approximation to the file. The HTML was then imported into Tcl using `tdom`.

Some experimentation revealed that every line representing a recreation area appeared as a row of a table, where the row had ten cells and the first cell contained a hyperlink. The name of the recreation area was the text of the link, and the remaining cells contained access restrictions and other attributes. This information is enough to make a work list.

Of course, the work list doesn't help all that much if geometry will have to be transcribed manually. The author was still thinking he'd have to do a Freedom of Information demand, but out of curiosity began to examine a few example maps. One such map is attached to this paper as an Appendix.

As is often the case with maps, it proved essential to read all the seemingly unimportant notations in the border. One of them offered a clue: "Prepared by BWS [Bureau of Water Supply] GIS". At one point, that department had been used as a reference customer for the ArcGIS system. Perhaps the PDF contained the structured commentary that ARCgis uses to identify map layers and provide projection information, so that the data can be reconstructed from the PDF? GDAL revealed that this was indeed the case. The file was revealed to be in UTM (Universal Transverse Mercator) projection, and contained layers with informative names like "Buildings", "Elevation Contours", and "Rivers, Ponds, Lakes and Reservoirs". A layer named "PAA" revealed itself to be "Public Access Area", and the import was back in business!

The script for data intake was therefore modified to download each map (a cache was added so that the download would be preceded with a HEAD transaction, and skipped if an up-to-date copy of the file was available locally). The downloaded file was then run through the `ogr2ogr` program (a general-purpose data conversion tool from GDAL) to push it into PostGIS.

There were several more glitches following this. The full list of what happens in the PostGIS calculations is probably of interest only to a maintainer of the script, but a few examples will be illustrative:

- The larger areas were divided into multiple strings of line segments and had to be reassembled.
- Some of the areas appear to have been mapped with raw, noisy GPS traces. These had far too many vertices, and contained topological errors like gaps and self-intersections.
- When noisy data was used for boundaries shared between two areas, it wasn't necessarily the same noisy data, giving rise to gaps and overlaps.

PostGIS was used to "do all the heavy lifting" of fixing all these problems, and it turned out that the fixes gave rise to only hairline differences in the mapped results.

A Tcl procedure a couple of pages long then converted all the table cells to proper OpenStreetMap keyword-value pairs. This procedure ends with an invocation of the appropriate SQL `INSERT … SELECT` statement to accumulate all the imported maps into a single table.

Data conflation

Once developed, the script for data intake can be run repeatedly without a human in the loop. In fact, the author runs it sporadically, several times a year, to look for changes (occasionally the city will purchase new land, or close an area to the public or change its access constraints). The task of identifying what data in OpenStreetMap are in possible conflict with the new import, and resolving the

1 In a private conversation, one of New York City's GIS specialists indicated that making the data available in the original form that the city worked with was entirely feasible. The analyst suggested a Freedom of Information demand for the original data. Since such a demand is often perceived as a hostile act by the managers of the low-level functionaries, the author decided to ignore this advice until processing the data in the available format proved to be too onerous.

conflict, is something that requires a certain amount of human judgment.

The first part remains fairly simple. A SQL query looks in OpenStreetMap for objects that overlap an imported object significantly (hairline overlaps at the borders are allowed). These go in a new table, that also adds a `done` flag to indicate that an object has received attention and conflicts are resolved.

Then a Tcl script functions as a "one-person tasking manager" to go through the new/changed areas one by one and present them for review in an editor. For each element in turn that is not marked 'done', it does the following:

- Execute the Python script `ogr2osm.py`, to pull the row from the database and output a file in OpenStreetMap format with the area's geometry and attributes.
- Command the editor, using its embedded Web server, to download the data from the bounding box of the area from OpenStreetMap; the editor will now have the current data.
- Command the editor to select the existing version of the area (if there is one) and zoom to the selection.
- Command the editor to load the new version of the area from the local file, as a new layer.
- Await input from the user. For the New York City import, the updater is a command line application, and asks only if all conflicts were resolved successfully. On an affimative answer, it sets the 'done' flag and presents the next area.

With both areas loaded in the editor, it is reasonably straighforward to check for changes to either geometry or attributes, and save the new data back to the OpenStreetMap database. While the initial import took several weeks, the author can usually import a few months' worth of changes and resolve conflicts in a couple of evenings' work.

This manager allows the conflation to proceed over multiple editing sessions.

A more sophisticated task manager

While the simple command-line task manager was entirely up to the task of importing the New York City data, the author felt that it was worthwhile to build a better one for another database, this one from New York State Department of Environmental Conservation (DEC). This database describes all of DEC's public land, totalling well over 20000 km². The land uses and access constraints are much more heterogeneous (including things such as campgrounds, fish hatcheries, historic sites and maintenance depots as well as the expected state forests and wilderness areas). For this reason, the OpenStreetMap tags are less stylized, and there is more data to check.

For updating this data set, a similar workflow is used. (The intake is simpler since the entire set is available as a 'shapefile' exported from the ArcGIS system, and can simply be poured into the local database using the `ogr2ogr` program). Fixing topology and assigning tags proceeds much as before (with a considerably more complex Tcl script assigning the tags).

Identifying the objects in conflict is more complex, because New York's updates have included some involved 'horse trading' making changes such as breaking up an area and dividing its land among adjoining ones, while at the same time annexing additional newly-purchased land. A single new feature can overlap with multiple old features, and so there needs to be further work to match them.

This was best done by matching the features' names, which, alas, are not always spelt correctly in the databases. Fortunately, this problem was solved by a few glances at Rosetta Code. The cost of a matching was modelled as the Levenshtein distance between the two names (http://rosettacode.org/wiki/Levenshtein_distance#Tcl), and the 'best' pairing is then found by solving the 'stable marriage problem' (http://rosettacode.org/wiki/Stable_marriage_problem#Tcl). Once more, the Tcler's strategy of "let someone else solve the hard part of the problem" pays off!

The task manager has a similar structure to the one for New York City's recreation areas, but has a Tk GUI attached to it. Figure 2 shows the layout.

The workflow for this UI is only a little more complicated than for the command-line one. The mapper begins by selecting one of the changed areas from the listbox. Making this selection also has the effect of commanding the editor to download the appropriate area from OpenStreetMap, and to load up the new area's geometry into a new layer.

Optionally, the mapper can select “Load Differences” which creates a new layer showing the symmetric difference between the old and new geometry. This is particularly useful if the state has made a small change to a large area with a complex boundary. It both directs the mapper to the change and helps ensure that nothing was missed.

The mapper now gets in the bottom panel a list of tags that belong to the area. In the case shown in the screen capture, there are no differences in the tagging. If there were differences, the text under ‘Value’ would be replaced with a combobox showing both values, and the mapper would have to choose what value to apply.

The buttons at the bottom allow the mapper to investigate further by visiting the area’s web site, to select a collection of tags using the checkboxes and apply the checked ones, to copy the checked tags to the clipboard (the JOSM editor has functions to paste tags), and to mark the object as ‘finished’ (whereupon it will disappear from the listbox). Once again, editing can proceed in many sessions; the initial import of these data was conducted in sporadic evening sessions over several months.

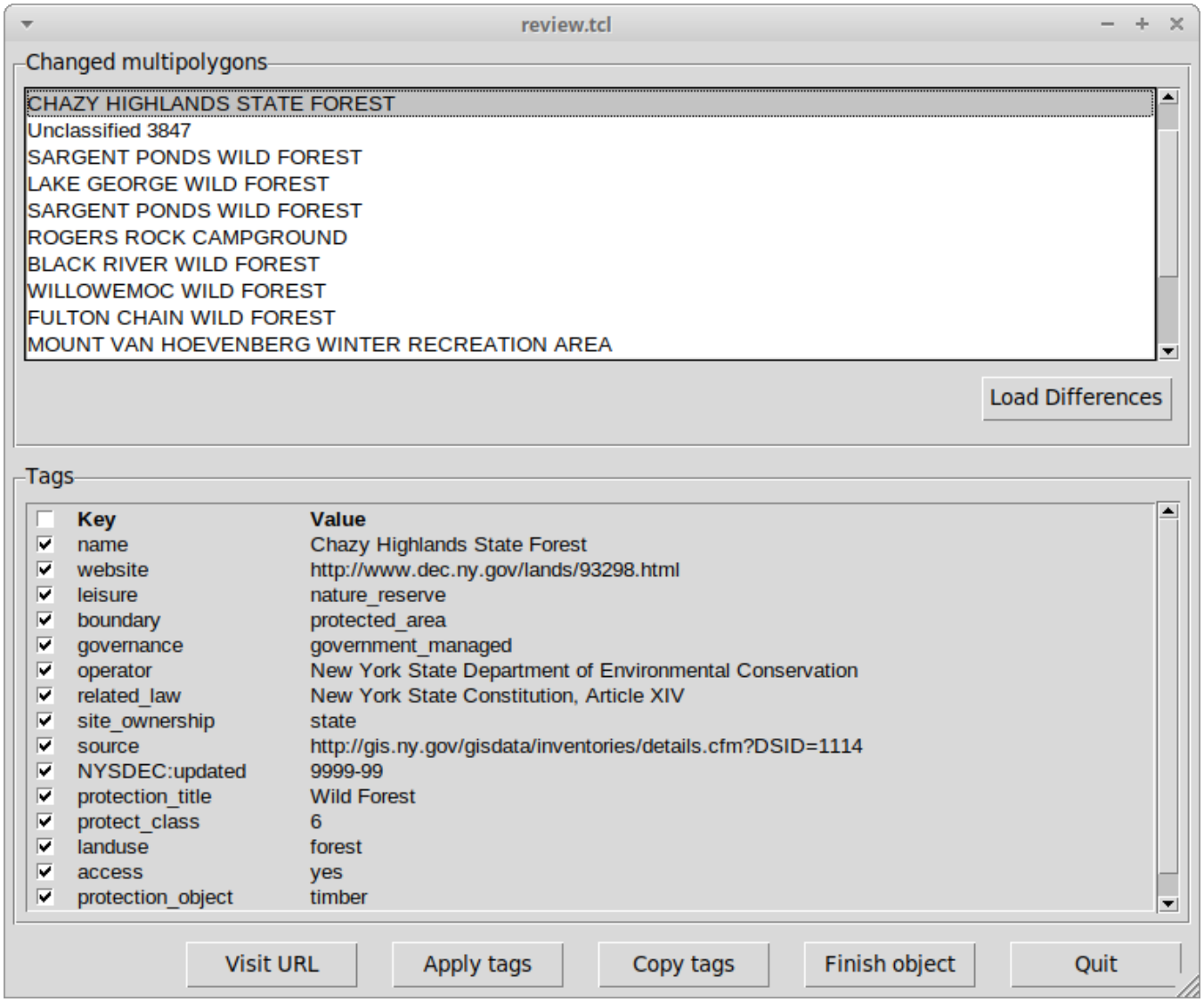


Figure 2. GUI for choosing and editing changed features

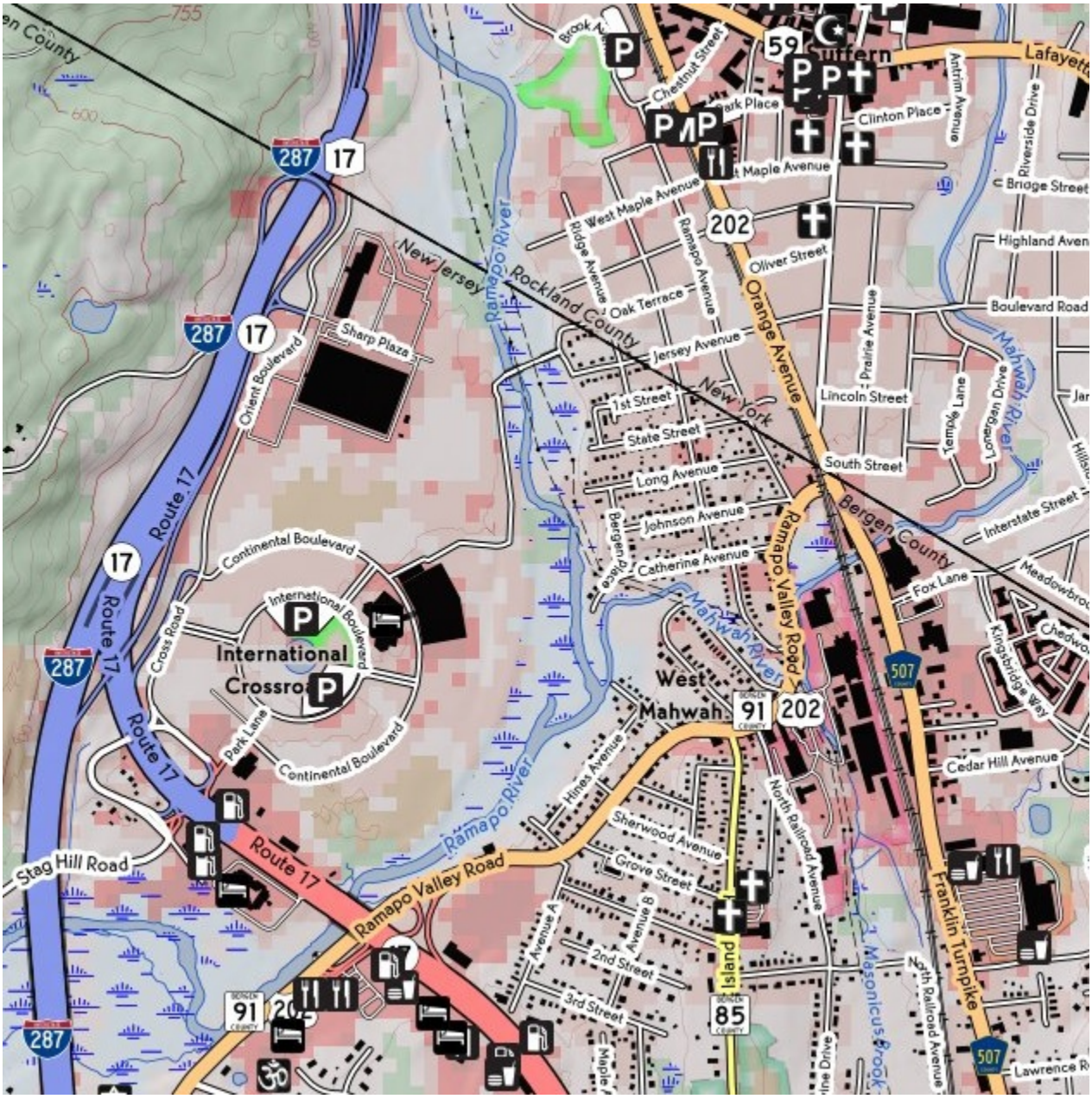


Figure 3. Fragment of a map showing numbered routes from multiple networks and route concurrences

Tcl in map rendering

As well as using Tcl in curating the map database, the author also makes use of it in drawing maps. The largest single application is code for drawing highway markers in North American style. This is needed to produce an American-style map that shows overlaid routes in a usable fashion. Figure 3 shows a highly information-dense map with that style of rendering. There are numbered routes from the Interstate and US highway systems, from two states and several counties; many of these numbered routes run concurrently for portions of their length.

Producing this rendering requires that the local OpenStreetMap data be augmented with some additional information that distills out the route concurrencies and presents them in a form convenient to render. (This is all done in a complicated set of stored procedures in the PostGIS database) It also requires producing the graphics for the markers. This last part is done in Tcl, with a script of a few thousand lines parsing out the route numbers and

deciding what shapes are required, and a procedure that then launches Inkscape to render the markers. Details are not presented here; the interesting reader can visit the project's Github, where the source code and documentation can be found.

Conclusion

Tcl is still well in touch with its roots as a 'glue language'. With a handful of popular extensions to perform common data exchange issues like HTTP transactions, SQL access, and HTML/XML/JSON parsing and fabrication, and with use of [exec] to run external programs for the 'heavy lifting', it is possible to build some quite complex systems with a relatively tiny amount of code.

References

[KUPR10] Kupries, Andreas. "Tcl/GSoC 2010", in Proc. 17th Annual Tcl/Tk Conference. Chicago, Ill.: Tcl Association, 11-15 October 2010, pp. 111-115. http://www.tclcommunityassociation.org/wub/proceedings/Proceedings-2010.html