Geographic support for Vega-lite

Youying Lin
University of Washington
Computer Science & Engineering
liny33@cs.washington.edu

Vivek Paramasivam
University of Washington
Computer Science & Engineering
paramv@cs.washington.edu

ABSTRACT
In this paper we describe how we add Geographic support to Vega-Lite, a high-level grammar that enables rapid specification of interactive data visualizations. This project involves adding geo projection to Vega-Lite and providing a tile layer to Vega/vega-embed. Ultimately, this allows for the use of the succinct Vega-Lite syntax to project latitude and longitude coordinates onto an Euclidean plane, as well as the ability to display more detailed geographic information about any given geographic coordinates.

INTRODUCTION
Vega-lite specifications are compiled into Vega specifications, which is a visualization grammar. Currently Vega-Lite does not support cartographic projections on geographic coordinates. Meanwhile, Vega has geo and geopath transforms which can perform these projections but without a tile layer. To make Vega-Lite support projections with a tile layer, we want to add the tile layer at the Vega and Vega-Embed level. Then, we adapt the Vega-Lite to allow users to visualize geographic coordinates.

RELATED WORK
Vega, Vega-Lite, and Vega-Embed are developed by the UW Interactive Data Lab. [2][3][4]

METHODS
In this section, we will discuss the choices we made in implementing our changes, and the reasoning behind those choices.

Leaflet Plugin for Vega and Vega-Embed
Instead of implementing our own tile layers at the Vega level, we opted to use Leaflet, an "open-source javascript library for mobile-friendly interactive maps." Leaflet's rich variety of interaction features made it an optimal choice for this task.

We opted to perform this implementation as a plugin because we wanted to make use of the Vega runtime API instead of modifying the actual Vega codebase.

Synchronization between the Vega layer and the Leaflet layer was the biggest hurdle we needed to overcome. We solved this by outputting the center, translate, and scale from the geo and and geopath transforms as signals which the Leaflet plugin can read and and adapt to accordingly.

Adding Geo Projections to Vega-Lite
The high-level goal for adding geo projections was simple: we modify Vega-Lite to allow users to note that a data element should be projected in a certain way. The actual implementation of this is actually fairly complex.

In order to support this feature, the first step was to determine the Vega-Lite syntax which would produce a Vega specification that projects latitude/longitude data onto a euclidean plane. We explored a number of approaches to this, and decided on the syntax descrisbed in Figure 1.

```
{
  "layers": [{
    "data":{
     "url":"data/us-10m.json",
     "format": {"type": "topojson", "feature": "counties"}
    },
    "mark":"path",
    "projection":{
        "type":"albersUsa",
        "zoom":1000,
        "translate":[400, 250],
        "center":[0, 0]
    },
    "encoding":{
        "geopath":{
            "field":"geodata",
            "type":"geojson"
        },
        "color": { "value": "#dedede"}
    },
    "config": {
      "mark": {"stroke": "white"}
    }
}, {
    "data": {"url": "data/demo-airports.csv"},
    "mark": "point",
    "projection": {
      "type": "albersUsa",
      "zoom": 1000,
      "translate": [400, 250],
      "center": [0, 0]
    },
    "encoding": {
      "x": {"field": "longitude","type": "longitude"},
      "y": {"field": "latitude","type": "latitude"},
      "size": {"field": "total", "type": "quantitative"}
    },
    "config": {
      "mark": {"filled": true}
    }
  }],
  "config": {
    "cell": {"width": 800, "height": 500}
  }
}
```

Figure 1. Vega-Lite syntax map projection

Here we will discuss the added syntax in a more detailed way:

For geopath transform, we added a new mark type and a new encoding transform. We added a new mark named path for the geopath transform which corresponds to the Vega path mark. The geopath encoding channel correponds to the extra properties geopath transform in Vega needed compared to geo transform.

The projection part, as shown below in Figure 2, is controlling the scale of the projected map. This figure is only showing a subset of properties users can set in this section. All the shared properties between geo transform and geopath transform in Vega can be specified in this section, if not specified, the properties will just be their default values which are what d3[1] has for them.

```
"projection":{
    "type":"albersUsa",
    "scale":1000,
    "translate":[400, 250],
    "center":[0, 0]
},
```

Figure 2. Projection section in Vega-lite

To detect if a map projection is requested, we checked the encoding channel. If $geopath$ is specified and of type $geojson$, or if any of the x and y channels are of type $latitude$ or $longitude$, it implies that a map projection is requested. We use the mark type to distinguish which transform is requested: a $path$ mark implies this is a geopath transform and a $point$ mark implies a geo transform. This requires users to use the correct mark type for each transform in order to generate a correct geo or geopath transform.

The next step was to identify and modify the parts of the Vega-Lite codebase which needed to be modified in order to implement the changes we needed.

A hurdle we ran into while performing these changes was that we had to perform some refactoring work on the current Vega-lite codebase to adapt the changes for the map projections part.

RESULTS

In this section we will overview the new visualizations which are possible in Vega and Vega-Lite as a result of our changes.

Projection in Vega-Lite

Users can now use Vega-Lite to project latitude/longitude data onto a variety of map projections, including mercator, hammer, and albers. Figure 3 shows the resulting figure produced from the Vega-Lite specification in Figure 1. It comprises of two datasets: a set of points representing airport locations, and a set of paths representing the outline of states, both of which have been projected on to an albersUsa projection.

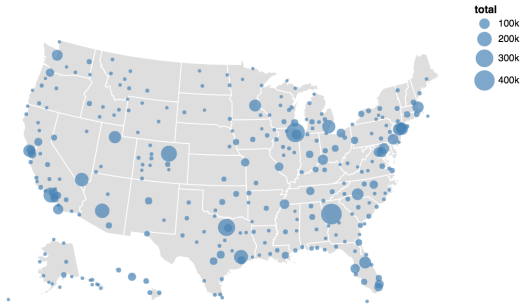

Figure 3. Map of the U.S. main airports in Mercator projection

In Figure 4, we can see the usefulness of the projection feature. By simply changing the projection type to stereographic, we have a very different looking projection. This can be helpful to users in many ways. In fact, Vega and Vega-Lite now support all of the projections provided by D3, from conicEqualArea and conicEquidistant to orthographic and even gnomonic.

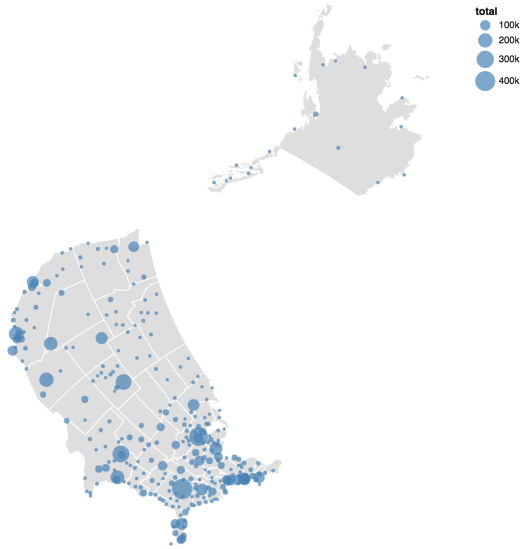

Figure 4. Map of the U.S. main airports in Stereographic projection

Vega Tile Layer Plugin

The Leaflet Plugin for Vega allows us to provide relevant geographic details to users without the need for the user to provide any extra data. Now, instead of providing a dataset with paths to create a map upon which to project geographic data, users can simply display the tile layer, and project their data upon the Leaflet-provided tiles. This greatly simplifies the process of visualizing geographic data.

We found that our implementation of the Leaflet plugin was very useful for visualizing geographic data in a rapid fashion due to the fact that the user does not need to specify their own map data. However, we found that if the user does provide a significant amount of data, such as in the case of geopaths, panning and zooming are very slow.

In addition, we found that due to the way that the plugin interacts with Vega, we cannot zoom Vega in the same way that the Leaflet-provided tiles zoom. If we tried this, instead what the user would see is the tiles zooming, while the Vega visualization remained static until Leaflet completed its zoom, at which point the Vega visualization would re-draw itself at the new zoom level. In order to circumvent this problem, we decided that during the zoom animation, we will hide the Vega visualization until leaflet finishes the zoom, and then we re-draw the vega. This hides this non-intuitive interaction from the user. Ideally, the Vega would zoom at the same rate, but that was outside the scope of our project.

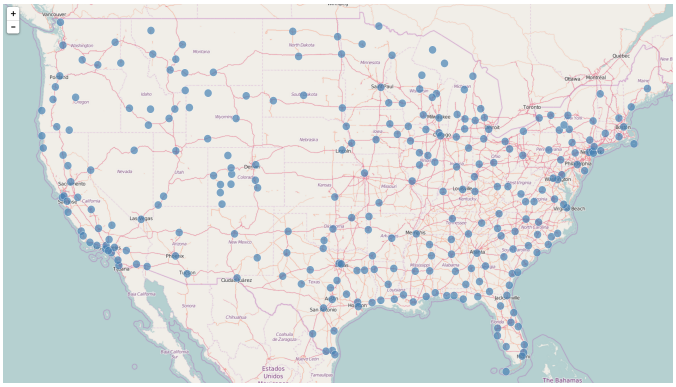

Figure 5. Map of the U.S. main airports in the Leaflet plugin

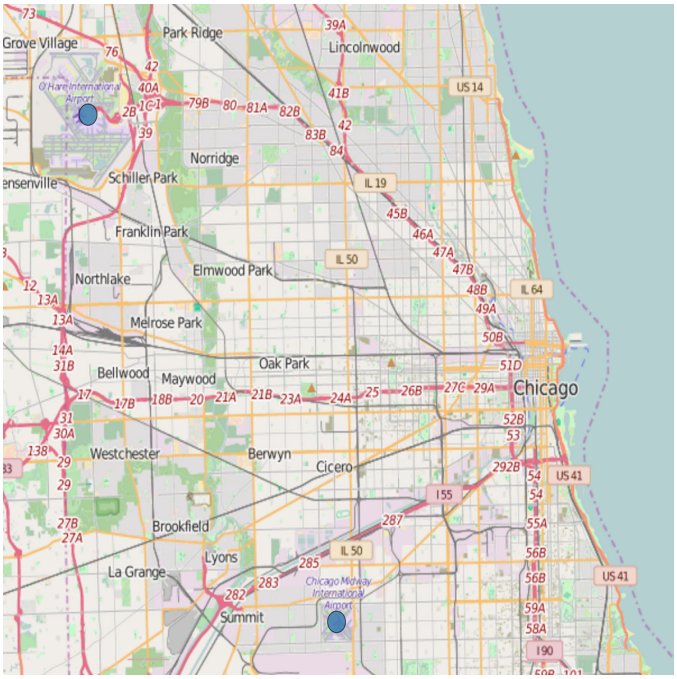

Figure 6. Map of the U.S. main airports in the Leaflet plugin, zooming into the Chicago area

DISCUSSION

In our interactions with people at a poster session, we had a number of interesting comments from visitors about our work. An individual from health sciences asked if she could use Vega to visualize rates of disease in areas around the world. She pointed out that she did not have experience writing code, and neither did anyone on her team; was it still possible to create a good visualization in a rapid manner? She was glad to hear that with the features we contributed to Vega-Lite, she could generate these kinds of visualizations with relatively little learning curve, as we showed her how a simple Vega-Lite spec could be written.

Another individual had an interesting proposition–he wanted to generate maps, but instead of a map on the scale of the world, using latitude and longitude, he wanted to map out a floorpan in Vega. Specifically, he posed the situation in which he has cameras around a building, and is using these cameras to detect when two people meet and interact, which he defined as two people facing each other for a certain amount of time. He wanted to visualize the locations in a room where people were meeting. He asked if this were possible in Vega or Vega-lite, to which we were able to confirm that Vega-Lite, with our contributions, would allow him to do this so long as he provides a set of paths to represent the floorpan.

However, he also asked about interacting with Vega-Lite visualizations with features such as hovering or click handlers. Unfortunately, we had to point out to him that Vega-Lite does not currently support the kinds of interaction features he was asking for, although regular Vega does.

This individual was not the only one to remark about the lack of interaction features. Another individual pointed out that it would be useful if our tile layer plugin could also display relevant information about places you click on or interact with, the way that Google Maps does. It would be interesting to find a dataset which maps coordinates to points of interest, and displays information about the nearest point of interest to which the user clicked on. This of course could be added as an optional feature which the user could enable from the specification.

Ultimately, we learned from these interactions that people see the value of our contributions both the geographic projections in Vega-Lite as well as the benefit of the tile layer plugin. In fact, a number of people eagerly asked, “When is this getting merged into Vega-Lite?”. However, many users were hoping for further interaction capabilities, which were of course outside the scope of our project.

Future Work

There are a number of directions in which this work could move forward from here. One extension would be to add on-hover handlers for marks in Vega-Lite, which do not exist yet. Another natural extension is to implement auto-zoom, which would automatically adjust the translation, center, and zoom to fit given data. In addition, our work with Leaflet is not yet integrated with our work from Vega-Lite. Performing this merge is a necessary extension.

As was mentioned in the Discussion, many users were upset with the lack of interaction features. Naturally, hooking

into Vega's interaction features, as well as integrating with Leaflet's interactions, would be a beneficial extension to our project. However, at the tile layer level there are some problems with how this would need to be implemented. Currently, interactions are passed through Vega to the tile layer, allowing Leaflet to drive panning and zooming. If we wanted to use Vega's interactions in this case, we would need to have Vega catch all events, then filter them, with some being passed through to Leaflet, while others are consumed by Vega.

An important point about our Leaflet plugin is that the tile provider can be replaced. That is, we currently use MapBox as the tile provider, but an extension to our project would be to replace it with other tile providers such as OpenStreetMap, and perhaps perform a comparison of performance.

A final project extension could be a modification to the current Vega-Lite spec for projections. Currently, a user must define the projection for each data source separately, as can be seen in Figure 1. Ideally, the user would only need to provide this information once, to keep the specification succinct. However, investigation into the best way to perform this syntax change is needed before more work can be done in this direction.

REFERENCES

1. M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-Driven Documents. *IEEE Trans. Visualization & Comp. Graphics*, 17(12):2301–2309, 2011.

2. A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer. Reactive Vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2016.

3. A. Satyanarayan, K. Wongsuphasawat, and J. Heer. Declarative interaction design for data visualization. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*, pages 669–678. ACM, 2014.

4. K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager: Exploratory Analysis via Faceted Browsing of Visualization Recommendations. *IEEE Trans. Visualization & Comp. Graphics*, 2015.