Large Scale Interactive Visualization of Urban Accessibility Data

Manaswi Saha and Coco Moke Mao

Problem

We are working on making visualizing large scale data faster on interactive maps. In order to do that, the fundamental challenge that needs addressing is of thinning. Authors in [1] describe thinning to be "determining appropriate samples of data to be shown on specific geographical regions and zoom levels." Our project is based on Project Sidewalk[1] data. Its an online crowdsourcing tool that allows the volunteers to contribute data on physical accessibility. The collected data are geographic locations of different types of accessibility features and issues such as curb ramps, missing curb ramps, surface problems, obstacles in path and missing sidewalks. Current number of labels in the database are 181,349. The base interactive map prototype for visualizing this data is very slow ~10s. In this project, we aim to reduce the time for visualizing this data by the help of sampling labels, different at each zoom level.

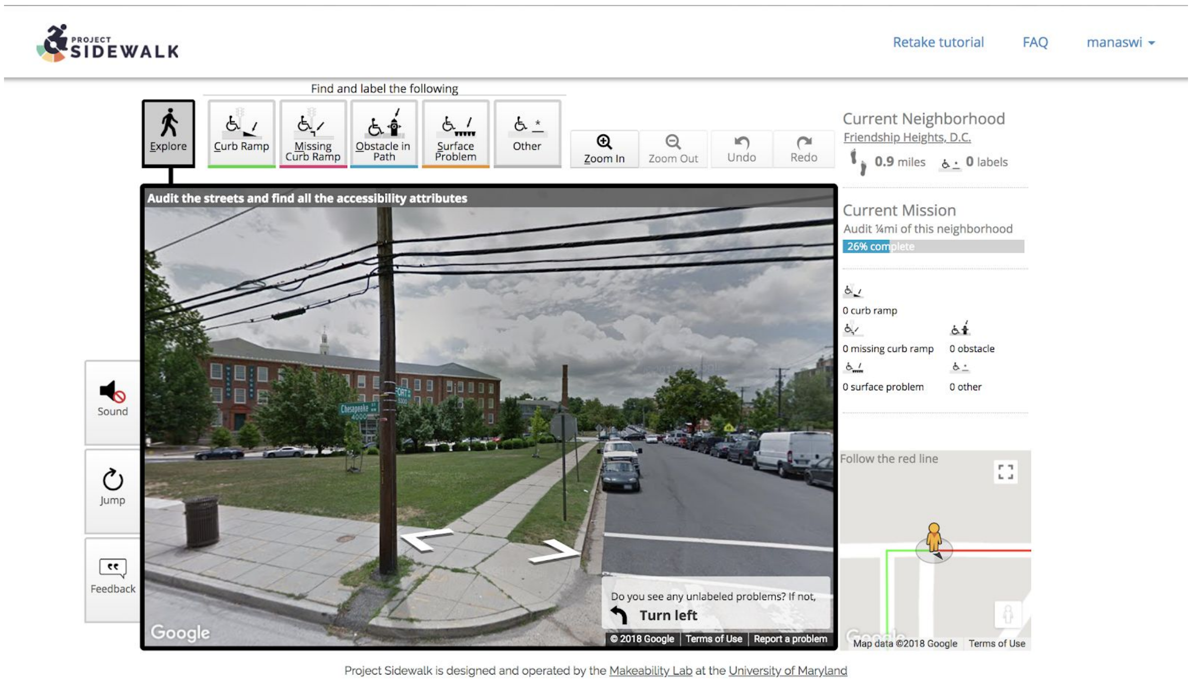

Figure 1: Project Sidewalk tool to collect the point label data capturing different label types

[1] http://projectsidewalk.io

Background and Related Work

R-tree: R-tree [2] was proposed by Antonin Guttman in 1984 and then widely used theoretically and in application. Similar to B tree, R-tree is a balanced search tree, but designed for indexing multi-dimensional data. The key idea is to group nearby objects and represent them with their minimum bounding rectangle (MBR) in the next higher level of the tree. At the leaf level, each rectangle describes a single object, and at the higher level, the aggregation of objects is bounded by the MBR. Since our database is static, the R tree will be pre-built in our case. The time complexity of search is O(log m N) where M is the maximum number of objects in a node and N is total number of objects in the dataset.

Spatial data thinning: Google [1, 3] has proposed and implemented a sampling algorithm for large geographical tables to address the very similar problem of determining appropriate samples of data to be shown on specific zoom level. In this work, several concepts are proposed which we have adopted in this project: 1) Visibility of a point when zooming; 2) Constraints of thinning on visibility and consistency. In this work, they used a balanced 4-ary rooted spatial tree and a DFS thinning algorithm to do sampling for complex geographical shape while we have a point only dataset where a random sampling should be sufficient.

Distinctive entries selection: Another work from University of Maryland [4] proposed an approach to efficiently sample more favourable data points (e.g. more popular photos compared to less popular ones) in a limited display window. They built a voting system on an ensemble of interrelated indexes which allows fast determination of distinctiveness of all entries with a query window.

Dynamic Client-Server Optimization: Moritz et al. from University of Washington [5] have explored the optimization of visualization plan in the form of dataflow graph by partitioning work across server and client. They have proposed a cost model for determining a partitioning of visualization plan that minimizes latency based on data statistics, network performance, computing and memory resources.

Finally, people have implemented similar map visualization using Quadtrees (https://www.domoritz.de/vbb-coverage/)

Approach

As mentioned above, we would be implementing a sampling approach that will allow us to request for less data based on the zoom level the user is at. The intuition is that requesting all of the data takes too long to be received at the client end. Hence, we want to restrict the number of data points sent to the client for rendering, in turn reducing the time for visualization.

We are following the SIGMOD'12 paper [1] on Efficient Spatial Sampling of Large Geographic Tables. The idea is to show only a few representative samples at the outermost zoom level and request for more data when the user zooms in. While performing these requests, we want to ensure that we satisfy the following constraints:

1. Zoom Consistency: As we zoom into finer granularities, the points shown on the coarser granularity should show up in the finer zoom level.
2. Data Distribution: The data to be visualized is of different types. We want to ensure at each zoom level, the distribution is maintained.

Ideally, the sampling would be done dynamically i.e. based on the view of the map, the data is requested for that certain view. Similarly, when the user pans, the map incrementally builds. However, in the interest of time and for the purposes of this project, we will be implementing a static sampling approach i.e. the samples for each zoom level would be pre-computed and stored in the backend Postgres database. When the user requests a certain zoom level, the data for the corresponding zoom-level will be requested. The pre-computation is done via the sampling algorithm we implemented in Python. Also, the panning portion will not be covered in this course project and is left for future work. We now describe the algorithm we used to do sampling.

The map visualization has 8 zoom levels, $Z \in [0,7]$. Let the coarsest granularity zoom level be 0, and let the finest level be zoom level 7. Any point in the map may be seen at a specific visibility $z \in [0,7]$. For example, if a point has a visibility of z=1, the point can be seen when the map zoom level $Z \geq 1$. The visibility z of each data point will be pre-computed and implemented in two ways:

1) 1 table that stores point index and visibility
2) 8 tables that stores point index that are visible at certain zoom level.

The efficiency of these two storage schemes are compared in the prelim work.
The point labels are selected using the following algorithm:

1) In the beginning, let z = 8 and add all the points into set $\boldsymbol{V}_z$.
2) Assign a zoom level of z to all points in $\boldsymbol{V}_z$.

3) Sample 60% of the points in $\boldsymbol{V}_z$ and add them into $\boldsymbol{V}_{z-1}$. z = z-1
4) Repeat step 2) and 3) until z = -1.

The sampling criteria in step 3 is to choose 60% label points with most severity in each label type. In the future work, we will also take geographic distribution into account.

Another technique we are going to use to accelerate visualization is to use R-tree to quickly retrieve labels in certain region when a user zooms in. The algorithm and application of R-tree will be further introduced in related work.

Implementation

We implemented an end-to-end system for an interactive map prototype. We prepopulated the database with the sampled labels using the offline sampling algorithm. Each time the user zooms in/out, requests for the zoom level data are made. The prototype is optimized such that we only make the request to the server once. All future zoom interactions relies on the local data for rendering.

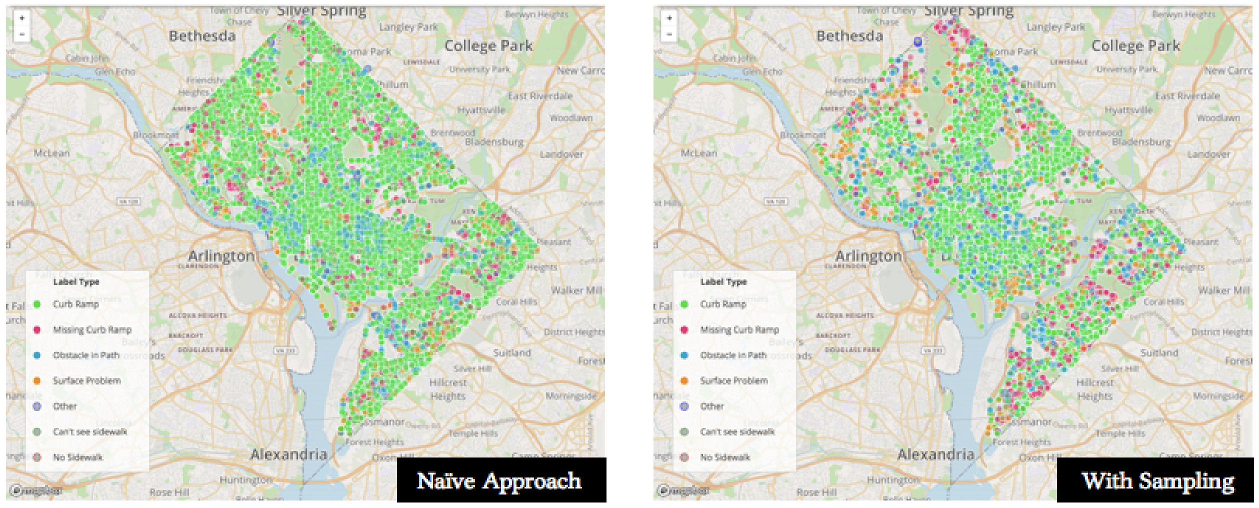

Figure 2: Illustration of the visualizations using the two approaches. This shows the map at the outermost zoom level, z=0. The difference is apparent in the number of point data seen in the figures. The naive approach had a download time of 7.02s, while the sampling approach had a download time of 1.86s.

We made two improvements once we had the basic setup done. First, we improved the sampling algorithm by sampling data per region. This was because sampling at the city level messed up the label distribution for each region. This is demonstrated in Figure 3. As we zoom-in, the label distribution is not maintained at each zoom level. From Figure 3, it is clear how this approach maintains the label distribution of the underlying data at all zoom levels.

Second, we used an hybrid approach to improve the interactive prototype. We found that even after decreasing the data load, the rendering itself took time. The sampling approach

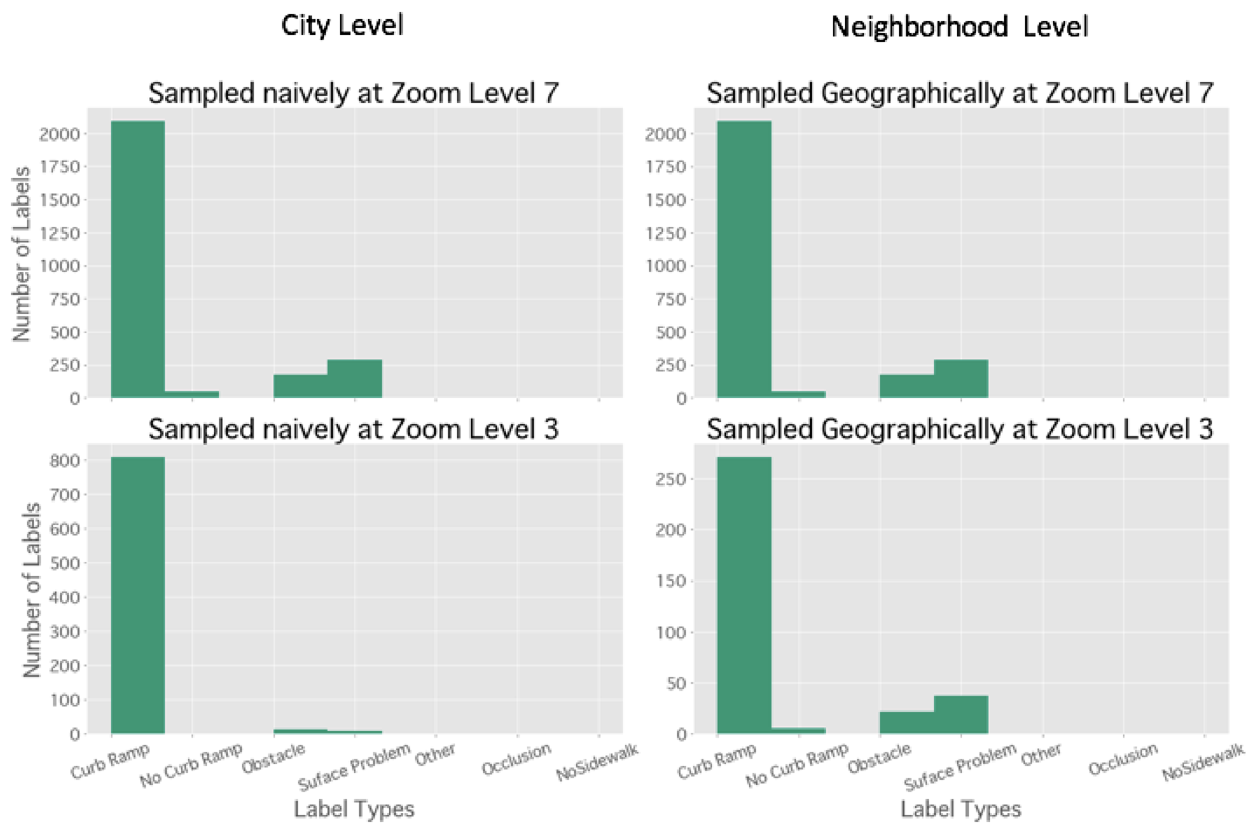

Figure 3: Illustration of the differences in sampling at the city level and at the neighborhood level.

starts to behave similar to the naive approach when you start zooming-in as you get more data for the whole city. See Figure 5 in the evaluation section. To remedy this problem, we using viewport based querying i.e. we only requested data at the zoo level for the region that is currently visible. We used a boundary box over the geographic region and send it to the server. The geographic coordinates of the box changes if the user pans or zooms. We found significant improvement in the response time. More details in the next section.

Evaluation

For evaluating, we compared three prototypes:

1. **Naive approach**: querying all the data together during the initial load of the page,
2. **Sampling Only approach**: Pre-sampling data for different zoom levels
3. **Hybrid Approach**: Requesting the pre-sampled data at the different level based on the current view of the map.

We compared all the three approaches based on two metrics: Download time and Rendering time. For a fair comparison, we kept the implementation for rendering the map and the data points same for all three prototypes. Using the browser's network console, we isolated the rendering time from network latency and total download time.

As it is visible from Figure 4, the download time and rendering time of the sampling approaches is significantly better than the naive approach: at zoom level of 5, the rendering time decreased from 18s (naive approach) to 100 ms (hybrid approach). Between the sampling approaches, the hybrid approach outperforms on both metrics as illustrated in Figure 5.

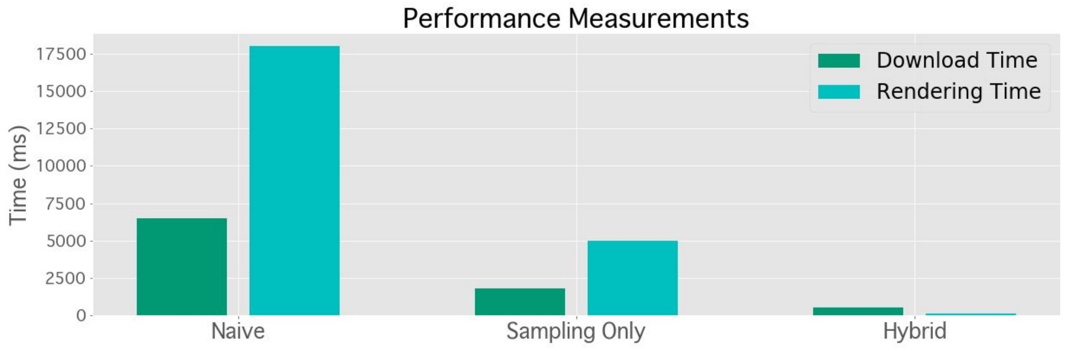

Figure 4: Data download time and rendering time of 3 approaches: Naive, sampling only and hybrid approach of sampling and viewport based querying.

As demonstrated in Figure 5, the downloaded data increases as we zoom into the lower levels. The simple trick of viewport based querying in the hybrid approach works really well to keep the downloaded data size to be consistent across different levels and as a consequence, reduces the download time. The reason is that at the lower levels, as we zoom in, the geographic region becomes smaller and smaller resulting in less data being requested. For the hybrid approach, the data across zoom levels is maintained at few KBs. The point to remember is that data size would depend on the labels contributed in that specific region. For e.g. regions with more problems will have more data than the others.

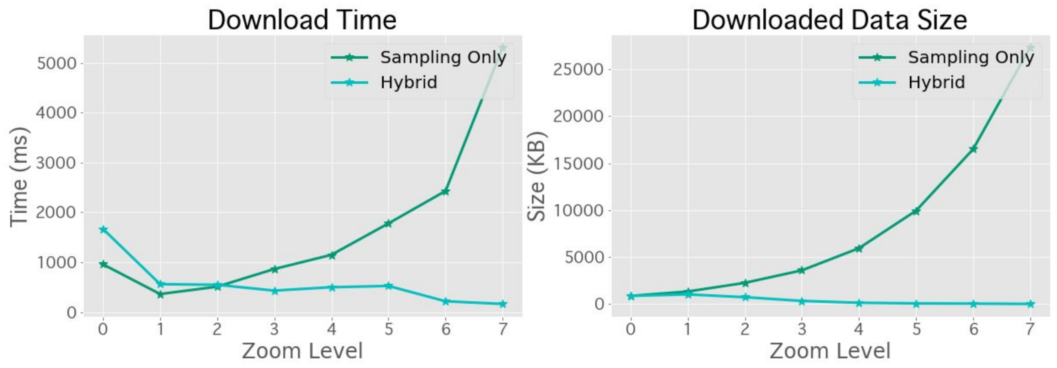

Figure 5: Performance comparison between Sampling Only and the Hybrid approach. Significant differences are seen across the metrics: data download time and downloaded data size across different zoom levels.

Conclusion and Future Work

We studied the problem of reducing visualization time for interactive map visualizations. This is particularly important when visualizing a large dataset. We used sampling and viewport based querying to reduce the data size and increase the response time. The project was done as a case study on Project Sidewalk's accessibility dataset. We improved the rendering significantly using the hybrid approach from 18s to 100ms.

The current work led us to think about an important aspect of visualization: sense making. Even though sampling works well to reduce the data size, we might lose important data points that will allow us to understand and interpret the data, in this case, accessibility of a region. Liu et. al mention this problem in [6]. As future work, we would like to explore this problem: what is the best way to represent accessibility and how can we build interactive prototypes that visualize accessibility and that are fast. There is a lot of related work in the area of information visualization that will inform us in this future direction.

Work Distribution

Coco - Worked on implementing the sampling algorithm - that involved reading the paper closely and writing the pre-sampling scripts.

Manaswi - Worked on implementing the interactive prototypes - that involved modifying the existing approach (used in Project Sidewalk), and implementing the client side code for rendering the interactive map prototypes and server side code to interact with the database.

Note: This is part of Manaswi's research project and hence, she is familiar with the existing codebase of the tool and the database implementation. Hence, she took lead on the interactive prototype implementation and Coco worked on the sampling techniques.

References

[1] Anish Das Sarma, Hongrae Lee, Hector Gonzalez, Jayant Madhavan, and Alon Halevy. 2012. *Efficient spatial sampling of large geographical tables*. In Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (SIGMOD'12). ACM, New York, NY, USA, 193-204. DOI: https://doi.org/10.1145/2213836.221385 .
[2] Guttman, Antonin. R-trees: A dynamic index structure for spatial searching. Vol. 14. No. 2. ACM, 1984.
[3] Anish Das Sarma, Hongrae Lee, Hector Gonzalez, Jayant Madhavan, and Alon Halevy. 2013. Consistent thinning of large geographical data for map visualization. ACM Trans. Database Syst. 38, 4, Article 22 (December 2013), 35 pages. DOI= http://dx.doi.org/10.1145/2539032.253903 4.

[4] Sarana Nutanong, Marco D. Adelfio, and Hanan Samet. 2013. An efficient layout method for a large collection of geographic data entries. In Proceedings of the 16th International Conference on Extending Database Technology (EDBT '13). ACM, New York, NY, USA, 717-720. DOI: https://doi.org/10.1145/2452376.2452462.
[5] Moritz, D., Heer, J., & Howe, B. (2015). Dynamic Client-Server Optimization for Scalable Interactive Visualization on the Web. In Workshop on Data Systems for Interactive Analysis (DSIA'15) (Vol. 9).
[6] Liu, Zhicheng, Biye Jiang, and Jeffrey Heer. "imMens: Real-time Visual Querying of Big Data." In Computer Graphics Forum, vol. 32, no. 3pt4, pp. 421-430. Blackwell Publishing Ltd, 2013.