Interactive Visual Exploration of Spatio-Temporal Urban Data Sets using Urbane

Harish Doraiswamy
harishd@nyu.edu

Eleni Tzirita Zacharatou
eleni.tziritazacharatou@epfl.ch

Fabio Miranda
fmiranda@nyu.edu

Marcos Lage
mlage@ic.uff.br

Anastasia Ailamaki
anastasia.ailamaki@epfl.ch

Cláudio T. Silva
csilva@nyu.edu

Juliana Freire
juliana.freire@nyu.edu

ABSTRACT

The recent explosion in the number and size of spatio-temporal data sets from urban environments and social sensors creates new opportunities for data-driven approaches to understand and improve cities. Visual analytics systems like Urbane aim to empower domain experts to explore multiple data sets, at different time and space resolutions. Since these systems rely on computationally-intensive spatial aggregation queries that slice and summarize the data over different regions, an important challenge is how to attain interactivity. While traditional pre-aggregation approaches support interactive exploration, they are unsuitable in this setting because they do not support ad-hoc query constraints or polygons of arbitrary shapes. To address this limitation, we have recently proposed Raster Join, an approach that converts a spatial aggregation query into a set of drawing operations on a canvas and leverages the rendering pipeline of the graphics hardware (GPU). By doing so, Raster Join evaluates queries on the fly at interactive speeds on commodity laptops and desktops. In this demonstration, we showcase the efficiency of Raster Join by integrating it with Urbane and enabling interactivity. Demo visitors will interact with Urbane to filter and visualize several urban data sets over multiple resolutions.

ACM Reference Format:
Harish Doraiswamy, Eleni Tzirita Zacharatou, Fabio Miranda, Marcos Lage, Anastasia Ailamaki, Cláudio T. Silva, and Juliana Freire. 2018. Interactive Visual Exploration of Spatio-Temporal Urban Data Sets using Urbane. In *SIGMOD'18: 2018 International Conference on Management of Data, June 10–15, 2018, Houston, TX, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3183713.3193559

1 INTRODUCTION

Why do two regions in a city feel similar? Or different? What are the characteristics that determine the quality of a city? The stakeholders shaping the future of a city including architects, city planners, and other policy makers typically rely on experience, precedent and data analyzed in isolation to answer such questions when making decisions that are critical in enabling vibrant and sustainable environments. The recent explosion in the number and size of spatio-temporal data sets from urban environments (e.g., [1, 5, 7]) and social sensors (e.g., [8, 10]) creates new opportunities for data-driven approaches through which the stakeholders can better collaborate and make more informed choices.

Architects, for example, need to have a strong understanding of a neighborhood's characteristics to identify potential sites for development. By using the available open data sets and comparing the neighborhood of interest with other neighborhoods, they can understand its strengths and weaknesses and establish performance thresholds from other well-known and well performing neighborhoods. This will eventually facilitate the negotiation process with the city planner, who is concerned with maintaining the quality of the neighborhood. To satisfy such requirements, working in collaboration with architects, we designed *Urbane* [2], a 3D visual analytics framework that supports data-driven decision making in the design of new urban developments. Among its various features, Urbane allows users to visualize a data set of interest at different resolutions over varying time periods. For example, the map view in Figure 1 visualizes the number of pickups performed by New York City's (NYC) taxis in the month of January 2009 aggregated over the neighborhoods of NYC. At the same time, Urbane also enables the visual comparison of several data sets through the data exploration view (see Section 3.1). These visualizations are primarily accomplished through spatial aggregation queries that compute an aggregate function over the result of a spatial join between two data sets, typically a set of points and a set of polygons. This operation can be translated into the following SQL-like query:

```
SELECT AGG(a_i) FROM P, R
WHERE P.loc INSIDE R.geometry [AND filterCondition]*
GROUP BY R.id
```

Given a set of points of the form $P(loc, a_1, a_2, \dots)$, where loc and a_i are the location and attributes of the point, and a set of regions $R(id, geometry)$, the query performs an aggregation (`AGG`) over the result of the join between P and R. Functions commonly used for `AGG` include the count of points and average of the specified attribute a_i. The `geometry` of a region can be any *arbitrary polygon*. The query can also have zero or more `filterConditions` on the attributes. In the above example, P is the taxi data, R is the set of polygons representing the neighborhoods of NYC, `AGG` is the *count* function, and the data is filtered over the given time range.

SIGMOD'18, June 10–15, 2018, Houston, TX, USA

ACM ISBN 978-1-4503-4703-7/18/06…$15.00
https://doi.org/10.1145/3183713.3193559

One of the main challenges in visual analytics systems is supporting interactive responses to user queries and actions, since high latency reduces the rate at which users make observations, draw generalizations and generate hypotheses [4]. In a system like Urbane, multiple spatial aggregation queries can be generated based on the user interactions; thus providing efficient support for these queries is crucial. However, this is challenging for several reasons. First, the point-in-polygon (PIP) tests that associate data points to polygonal regions containing them require time linear with respect to the size of the polygons. Real-world polygonal regions have complex shapes, often consisting of hundreds of vertices. This problem is compounded by the fact that data sets can have hundreds of millions to several billion points. Second, since *the query rate is very high*, delays in processing a query have a snowballing effect over the response times. Furthermore, existing spatial join techniques, common in database systems, are costly and often suitable only for batch computations. Finally, while data cube-based structures (e.g., [3]) can be used to maintain aggregate values, they require costly pre-processing and can incur prohibitively high memory overhead. More importantly, these techniques *do not support queries over arbitrary polygonal regions*, and thus are unsuitable for our purposes.

To address these challenges in processing spatial aggregation queries over large spatio-temporal data sets, we introduced *Raster Join* [9], a rasterization-based approach that leverages current generation graphics hardware (GPUs). As we show in [9], Raster Join can execute queries involving over 868 million points in only 1.1 second even on a current generation laptop.

In this demo, we integrate the Raster Join approach into Urbane, thus allowing users to interactively explore, over space and time, several urban data sets at multiple resolutions.

2 RASTER JOIN

The design of Raster Join builds on the following key insights:

- *Insight 1:* A spatial join between two data sets is essentially the intersection observed when the two data sets are “drawn" on the same canvas;
- *Insight 2:* There is no need to materialize the result of the spatial join since the goal of the query is to compute aggregates; and
- *Insight 3:* When working with visualizations, small errors can be tolerated if they cannot be perceived by the user in the visual representation.

Insight 1 allows us to frame the problem of evaluating spatial aggregation as renderings, using operations that are highly optimized for the GPU. In particular, it allows us to exploit the *rasterization* operation to convert a polygon into a collection of pixels. As part of the driver provided by the hardware vendors, rasterization is optimized to make use of the underlying architecture and thus maximize occupancy of the GPU. Using Insight 2, Raster Join couples the aggregation operation with the actual join. The advantages of this are twofold: (i) no memory needs to be allocated for storing join results, allowing the GPU to process more input data, and thus compute the result in fewer passes; and (ii) since no materialization (and corresponding data transfer overhead) is required, query times are improved. By allowing approximate results, Insight 3 eliminates the need for costly PIP tests, leading to a significant performance improvement over traditional techniques. Moreover, it allows an

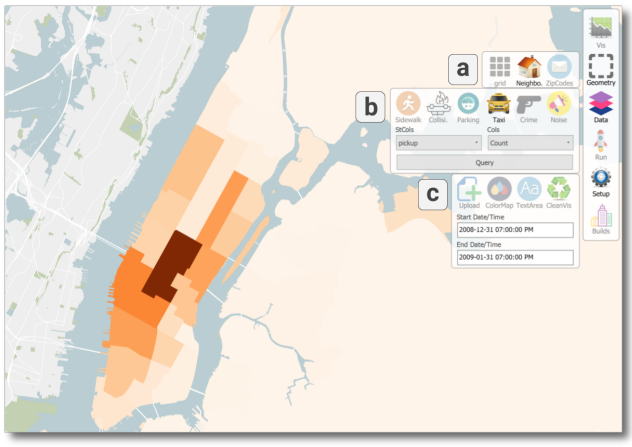

Figure 1: The map view of Urbane. The density of NYC taxi data (b) is visualized over the neighborhood regions (a) for a chosen time range (c). The menu highlights this selection.

algorithmic design in which the input data is transferred *only once* to the GPU, further reducing the memory transfer overhead.

We now briefly describe the graphics pipeline that forms the base of our approach, followed by describing the Raster Join approach. More details can be found in [9].

2.1 Rasterization-based Graphics Pipeline

The graphics pipeline, that is used to render a scene comprising of a set of triangles, is composed of a series of processing stages. First, the coordinates of all the vertices (of the triangles) are transformed into a common world coordinate system, and then projected onto the screen space. Next, triangles falling outside the screen (also called *viewport*) are discarded, while those partially outside are *clipped*. Parts of triangles within the viewport are then *rasterized*. *Rasterization* converts each triangle in the screen space into a collection of *fragments*. Here, a *fragment* stores the data corresponding to a pixel. The fragment size therefore depends on the *resolution* (the number of pixels in the screen space). In the final step, each fragment is appropriately colored and displayed onto the screen.

Instead of directly displaying the rendered scene onto a physical screen (monitor), it is also possible to output the result into a “virtual" screen. The virtual screen is represented by a *frame buffer object* (FBO) and has a user-defined resolution. Each pixel of the FBO has 4 32-bit values, corresponding to the red, blue, green, and alpha color channels. Since our goal is to compute the result of a spatial aggregation, we do not make use of any physical screen, but we make extensive use of FBOs to store intermediate results.

2.2 Bounded Raster Join Approach

The design of Raster Join builds on the aforementioned insights. Intuitively, our approach *draws* the points on a canvas and keeps track of the intersections by maintaining *partial aggregates* in the canvas cells. It then *draws* the polygons on the same canvas, and computes the aggregate result from the partial aggregates of the cells that intersect with each polygon. The above operations are accomplished in two steps as described next.

1. Render points: Each point is transformed into the screen space, and the fragment corresponding to it is rendered onto an FBO.

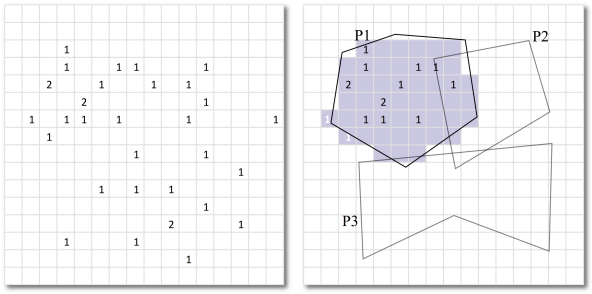

Figure 2: Raster Join first renders all points onto an FBO storing the count of points in each pixel (a). It then aggregates the pixel values corresponding to polygon fragments (b).

In this FBO, we use the color channels of a pixel for storing the partial aggregate (e.g., count, sum, etc.) of all the points falling in that pixel. For example, for the count aggregate, we *add* to the color of the pixel (e.g. the red channel of the pixel is incremented by 1). This step results in an FBO storing the aggregate over points that fall into each of its pixels. Figure 2(a) illustrates this step when *count* is used to aggregate on the example input shown in Figure 3.

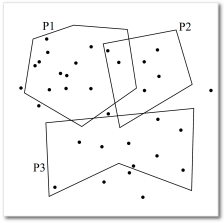

Figure 3: Input.

2. Render polygons: In this step (Figure 2(b)), we incrementally update the query result. To do so, we maintain a result array A of size equal to the number of polygons which is initially set to 0. The vertices of the polygons are first transformed into the screen space as before. These transformed polygons are converted into discrete fragments by the rasterization process. Each polygon fragment is then processed as follows: the partial aggregate of points falling in the pixel corresponding to this fragment is retrieved from the FBO from the previous step, and is used to update the aggregate in the result array corresponding to the polygon. After all polygons are rendered, the array A stores the result of the query.

Bounding errors. The above technique is approximate, potentially introducing errors for fragments that intersect the boundary of a polygon. In the example of Figure 2(b), the false positive counts are highlighted in white. Such errors can be controlled by increasing the resolution at which the above renderings are performed—as the pixel size decreases the approximate aggregates converge to the actual result. Formally, the accuracy is controlled by specifying a bound on the Hausdorff distance between the input polygons and the pixel-approximated polygons. Note that this is acceptable in real-world scenarios, where the data is inherently uncertain. For example, neighborhood boundaries typically coincide with street segments, and thus the entire street surface (and not just a thin line) is considered to be the boundary (so the width of the street can be used as the required bound). Furthermore, when working with visualizations, it is often impossible to perceive small approximations. We would however like to note that, in case users require exact results for further analysis, they can instead use an accurate variant of the Raster Join approach (see [9]).

To implement our approach, we customized parts of the rendering pipeline using OpenGL [6], a cross platform graphics API.

3 URBANE

In this section, we first briefly introduce the Urbane interface followed by describing the integration of Raster Join to speedup the visual exploration.

3.1 The Urbane Interface

The visual interface of Urbane is comprised of two components: the *Map View* (see Figure 1) and the *Data Exploration View*.

Map View. This view is composed of a map rendering component. The various menus and panels are overlaid on the map. Navigation and operations on map view such as panning, zooming, and rotating the view are accomplished through mouse interactions. The main menu (right side of map view in Figure 1) allows users to control all the functionalities of the system. This includes loading or deleting urban data sets as well as polygonal regions that define the different resolutions. Users can then choose the data set to be visualized along with the visualization resolution. For example, in Figure 1, the NYC taxi data is chosen to be visualized and the aggregation is performed over the neighborhoods of NYC (chosen polygonal regions). The menu also allows users to toggle the data exploration view that enables a comparative exploration of the different data sets.

Data Exploration View. The main goal of the data exploration view is to support the analyses of urban data at two different resolution levels—*region* and *building*. This view consists of two components—a parallel coordinates chart (PCC) and a data table (see Figure 4). At the region level, the PCC allows users to visually analyze and compare multiple data sets across different polygonal regions (selected via the menu), and the data table shows the values for each of the regions. In the PCC, each data set (or dimension) is represented as a vertical axis, and each region is mapped to a polyline that traverses across all of the axes, crossing each axis at a position proportional to its value for that dimension. This visual representation is effective for analyzing multivariate data, and can provide insights into the relationships between different indicators. Users can also filter regions by brushing the desired range of values on individual axes of the PCC. This updates the map by highlighting all regions that satisfy the filter constraints (Figure 4(a)).

Users can also drill down into the building level by selecting a region of interest from the data table, and choosing the building option. At the building level, users can perform a similar exploration as above, i.e., visualize and analyze the different data sets, but in the context of each of the buildings within the selected region. This operation is illustrated in Figure 4(b). Here, the value for each building is computed by aggregating the data within a fixed radius of the building. That is, the polygons used in the join correspond to circles centered around the different buildings.

3.2 Integrating with Raster Join

The Raster Join approach primarily takes as input a set of points and a collection of polygons, and computes the spatial aggregation as the output. Even though it also supports filtering the data over multiple attributes, it could still become inefficient to execute Raster Join over an entire data set due to the memory transfer overhead between the CPU and the GPU. Thus, to reduce this overhead, we store the different urban data sets in a 3D grid index of fixed size, where the dimensions correspond to the location (2 coordinates) and time. The extent of the grid in time is provided as a hint by the

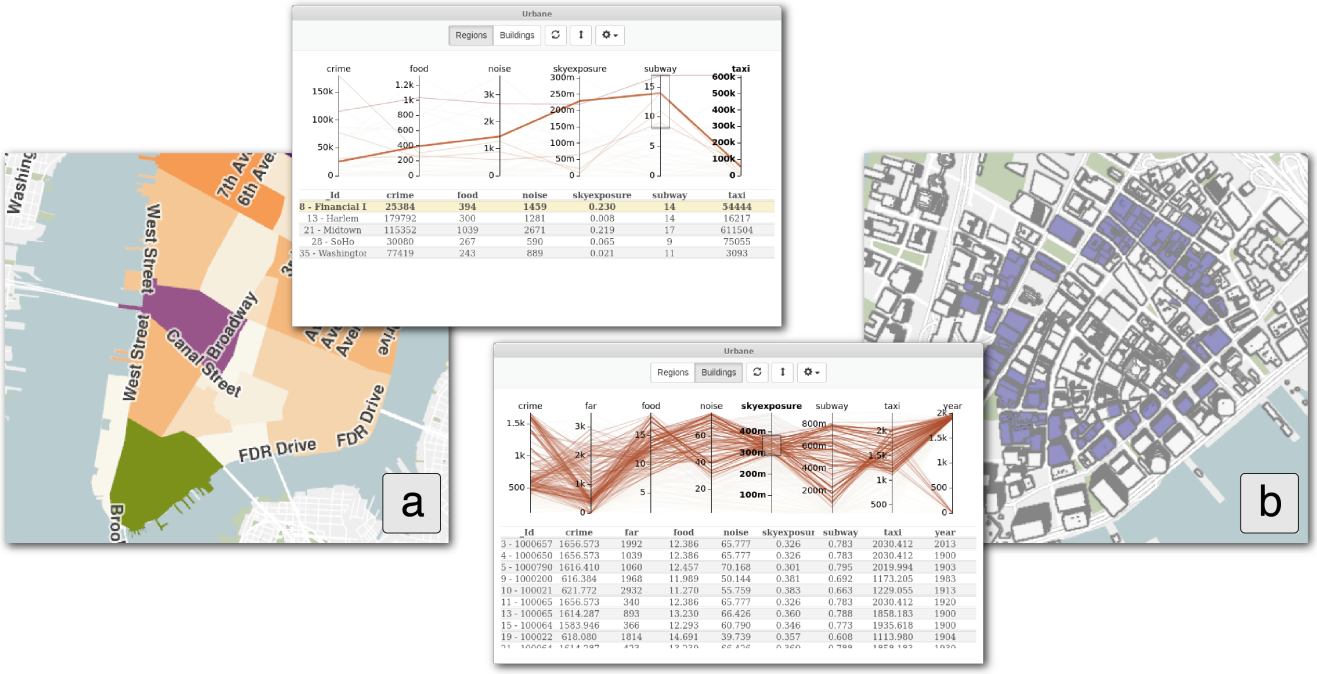

Figure 4: Multi-resolution exploration. Neighborhoods having a high density of subway stations are highlighted, and Financial District is selected for further analysis (a). Exploring buildings in the selected region (b).

user, while in space it depends on the city that is being explored. To handle outlier points that lie outside the defined extent, we simply associate them with the closest grid cell. Based on the query parameters—the time range and the extent of the polygonal data sets, only data from the appropriate grid cells are transferred to the GPU for further processing. In practice, this approach significantly reduces the amount of data that is being transferred to the GPU.

Urbane generates queries for two different operations—visualizing on the map, and visualizing on the PCC. For both these cases, we execute Raster Join using a pre-configured 20 meter bound. However, users can change this bound if they require higher accuracy.

4 DEMONSTRATION

In our demonstration, we will allow visitors to interact with the Urbane system including the following operations:

- Choose data sets to visualize along with the regions.
- Zoom and pan into regions of interest and control the accuracy of the obtained results by modifying the Hausdorff distance bound.
- Filter regions using the PCC.
- Select a region of interest to analyze the data at the building level resolution.
- Perform the same exploration as above (zoom, pan, filter on PCC) at the building level.

In addition to letting visitors do their own exploration, we will present two case studies that are part of an architect's workflow: 1) Understanding the Financial District neighborhood in the broader context of other neighborhoods in Manhattan; and 2) identifying sites with development potential. Figure 4 illustrates an example. As part of these case studies, we will use data sets from NYC. This includes point data sets such as the taxi data, 311 noise complaints, crime, sky exposure, subway and restaurant locations with sizes varying from a few thousand points to hundreds of millions of points; and polygonal regions such as neighborhoods and zip codes. The demo will be run live on a laptop. Finally, we encourage visitors to bring their own data sets that we can explore with Urbane.

Acknowledgements. This work was supported in part by: the Moore-Sloan Data Science Environment at NYU; NASA; DOE; NSF awards CNS-1229185, CCF-1533564, CNS-1544753, CNS-1730396, and OAC 1640864; EU Horizon 2020, GA No 720270 (HBP SGA1); EU FP7 (ERC-2013-CoG), GA No 617508 (ViDa); CNPq; and FAPERJ. J. Freire and C. T. Silva are partially supported by the DARPA MEMEX and D3M programs. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

REFERENCES

[1] Chicago Open Data 2018. https://data.cityofchicago.org/. (2018).
[2] N. Ferreira, M. Lage, H. Doraiswamy, H. Vo, L. Wilson, H. Werner, M. Park, and C. Silva. 2015. Urbane: A 3D framework to support data driven decision making in urban development. In *Proc. IEEE VAST 2015*. 97–104.
[3] L. Lins, J.T. Klosowski, and C. Scheidegger. 2013. Nanocubes for Real-Time Exploration of Spatiotemporal Datasets. *IEEE TVCG* 19, 12 (2013), 2456–2465.
[4] Z. Liu and J. Heer. 2014. The Effects of Interactive Latency on Exploratory Visual Analysis. *IEEE TVCG* 20, 12 (2014), 2122–2131.
[5] NYC Open Data 2018. http://data.ny.gov. (2018).
[6] D. Shreiner, G. Sellers, J. M. Kessenich, and B. M. Licea-Kane. 2013. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.3* (8th ed.). Addison-Wesley Professional.
[7] TLC Trip Record Data 2017. http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml. (2017).
[8] Twitter API 2018. https://dev.twitter.com/. (2018).
[9] E. Tzirita Zacharatou, H. Doraiswamy, A. Ailamaki, C. T. Silva, and J. Freire. 2017. GPU Rasterization for Real-Time Spatial Aggregation over Arbitrary Polygons. *PVLDB* 11, 3 (2017), 352–365.
[10] Yahoo Labs 2018. https://webscope.sandbox.yahoo.com/. (2018).