Scout: A GPU-Aware System for Interactive Spatio-temporal Data Visualization

Harshada Chavan, Mohamed F. Mokbel

Department of Computer Science and Engineering
University of Minnesota, Minneapolis, MN 55455, USA
{chava057, mokbel}@cs.umn.edu

ABSTRACT

This demo presents Scout; a full-fledged interactive data visualization system with native support for spatio-temporal data. Scout utilizes computing power of GPUs to achieve real-time query performance. The key idea behind Scout is a GPU-aware multi-version spatio-temporal index. The indexing and query processing modules of Scout are designed to complement the GPU hardware characteristics. Front end of Scout provides a user interface to submit queries and view results. Scout supports a variety of spatio-temporal queries-range, k-NN, and join. We use real data sets to demonstrate scalability and important features of Scout.

1. INTRODUCTION

Interactive spatio-temporal data visualization has emerged as one of the most important tools to explore data and derive valuable insights leading to strategic decisions [11]. Examples of such visualization include analyzing Twitter data for a certain event, analyzing taxi trips for traffic monitoring, and analyzing flight data for air traffic. Interactive spatio-temporal data visualization boils down to a set of consecutive spatio-temporal operations that correspond to users' actions in exploring the spatial space by panning and zooming. Therefore, to be effective, interactive data visualization needs to adhere to stringent query latency requirements, where queries need to be answered within 160ms [6], otherwise, it may severely impact the quality of the generated hypothesis [4]. As a result, several generic data visualization systems were introduced in academia (e.g., SeeDB [13], [2], imMens [5]) and industry (e.g., Tableau [10]). However, such systems suffer from one or more of the following two main drawbacks: (1) They are designed to deal with data visualization in general, with no special support for a variety of spatial and spatio-temporal operations, and (2) They do not scale up to the massive amounts of big spatial data, commonly generated by various applications, e.g., multi-billion cell phone records [9], over a billion records generated by taxi trips in New York city [8], a million trip records generated by Uber in China alone [12].

This research is supported by NSF grants IIS-0952977, IIS-1218168, IIS-1525953, and CNS-1512877.

SIGMOD'17, May 14-19, 2017, Chicago, IL, USA

DOI: http://dx.doi.org/10.1145/3035918.3056444

In this demo, we present Scout; a scalable interactive visualization system for exploring big spatio-temporal data. Scout is designed with built-in spatio-temporal data structures and access methods. Hence, Scout gives much better performance for spatial and spatio-temporal operations than prior generic visualization systems. In particular, Scout supports spatial and spatio-temporal range queries, aggregate queries, k-Nearest Neighbor queries, and join queries. To scale up with the massive amounts of big spatial and spatio-temporal data, Scout exploits the enormous compute capability of Graphics Processing Units (GPUs) to parallelize computations, which results in real-time query performance. It is important to note that Scout does not use GPUs just as more efficient processor than regular CPUs. Instead, *Scout designs its spatio-temporal index structures and operations specifically for GPU processing.*

Scout is a full-fledged system that supports all basic spatio-temporal operations that can be combined together to support a myriad of spatio-temporal functionality, e.g., heat maps, spatial analysis, and event detection. This distinguishes it from prior work that used GPUs to support one particular type of spatio-temporal interactive queries under a certain environment, namely, range queries [1] or aggregate queries [3] for historical data. Another distinguishing characteristic over prior work is that Scout supports data inserts without rebuilding its index structure.

The main idea behind Scout is a multi-version spatio-temporal index structure, where time is sliced into equal intervals. Each temporal interval includes a quadtree-based spatial index structure that has been adapted to match the architecture of GPUs. In particular, instead of representing quadtrees in a traditional hierarchical manner, we store the final partition boundaries in contiguous memory locations, which matches GPU's access pattern. Similarly, index lookup has been modified to take advantage of the parallel processing on GPUs. Instead of traversing the quadtree to reach the required leaf nodes, we simply perform a parallel test on the partition boundaries array to identify the required leaf nodes of the quadtree. Such GPU-aware adaptations allow Scout to make more coalesced memory accesses and reduce expensive thread synchronizations; properties desired by an ideal GPU computation. While processing the query, Scout avoids unnecessary data transfers between CPU and GPU by pruning the quadtree partitions that are guaranteed not to contribute to the query answer.

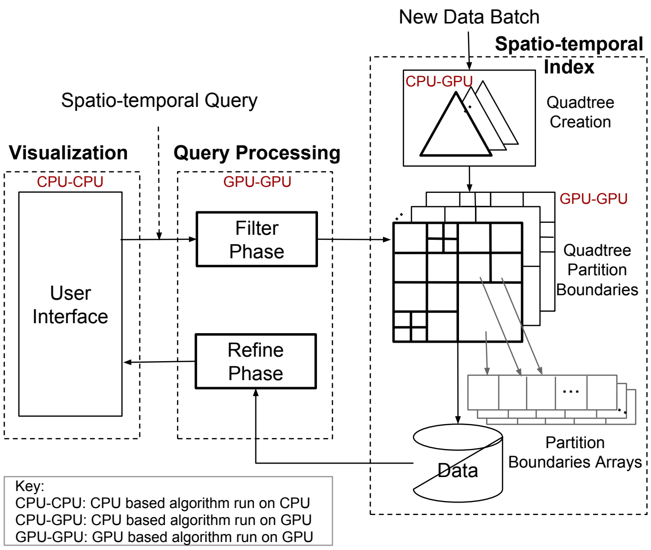

Figure 1: System Overview

2. SYSTEM OVERVIEW

Figure 1 gives an overview of Scout, composed of three main modules: *visualization*, *indexing*, and *query processing*.

Visualization. The visualization module serves three purposes. First, it acts as a user interface that allows system users to explore the spatio-temporal data on a map by panning and zooming, while specifying certain analysis functions, e.g., heat maps, data plots. Second, it transforms user actions and requests into corresponding spatio-temporal queries and sends to the query processing module. Third, it displays query result received from query processing module. Details are in Section 5.

Query processing. The query processing module receives its queries from the visualization module, and executes them through two phases, namely, *filter* and *refine* phases. The *filter* phase uses the underlying spatio-temporal index structure to early prune index partitions that are not going to contribute to the final answer. The *refine* retrieves candidate partitions from the underlying spatio-temporal index structure and performs actual expensive computational geometry operations to come up with the final query result. Both the *filter* and *refine* phases are designed to take advantage of GPU parallel processing. Details are in Section 4.

Spatio-temporal index. Spatio-temporal index is used in *filter* phase of query processing. The index receives lookup keys from the query processing module in the form of temporal and spatial ranges mentioned in the query. The result of the lookup operation is a list of candidate partitions. The structure and access methods of quadtree are designed to adapt to GPU architecture. Scout uses array based quadtrees, instead of traditional hierarchical quadtree structure. The leaf partitions of quadtree are stored in contiguous memory locations to eliminate the need of tree traversal during index lookup. These two changes help in reducing the expensive conditional execution and thread synchronization operations, and employ more coalesced memory accesses. Time slicing allows easy updates to the existing index when new data batches are added. Details are in Section 3.

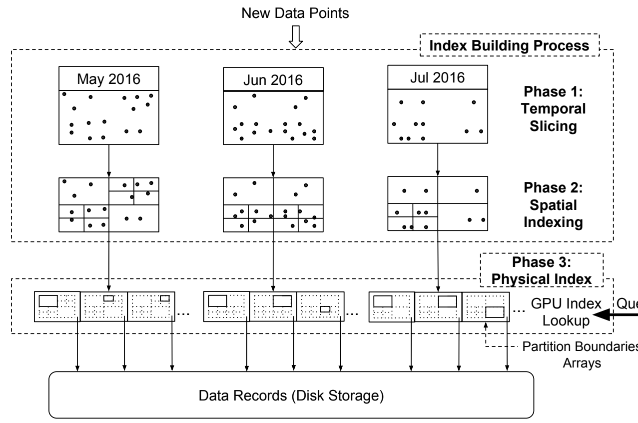

Figure 2: Spatio-temporal Index Structure

3. SPATIO-TEMPORAL INDEX

Figure 2 gives the spatio-temporal index structure employed by Scout, which is first created through a bulk loading process, then updated with a batch of new incoming data. This section focuses on how to build and update the index structure, while Section 4 focuses on how to query this index structure. Scout builds and updates its spatio-temporal index structure through three phases, namely, *temporal slicing*, *spatial indexing*, and *physical indexing*, described below. It is important to note that each of these three phases is designed specifically for the GPU environment. All design decisions were taken based on how GPU friendly they are.

Phase 1: Temporal slicing. This phase partitions new or incoming data into one or more equal time slices, in a temporally increasing order, e.g., one slice for May 2016, another slice for June 2016, and so on. Each time slice contains data that has arrived within the temporal interval of the slice, regardless of where the data is located in the space. We strongly prefer to use time slicing independent from the spatial domain, over prior spatio-temporal index structures that combine both space and time together (see [7], for a comprehensive survey), for two main reasons: (1) Queries posed to big spatio-temporal data mostly need to be bounded temporally, hence the temporal filter has to be in most posed queries. To clarify, a query about finding tweets that mention Trump or a query about Taxi drop-off locations in New York does not have much meaning without a temporal filter. For example, if the query is to find tweets about Trump in New York, would a tweet that was posted few years ago be relevant? If so, the size of the answer may not be practical. So, such a query needs to have a temporal filter, e.g., last week, or during a certain month. Same can be said about taxi drop-off locations. Time-slicing index structures are the most appropriate for such kind of queries. (2) Updating the data is much easier with temporal slicing. Whenever a new data batch is added, it only affects the most recent time slice, and may add one more time slices. This is in contrast to the need to rebuild the whole index in case we are not using time slicing.

Phase 2: Spatial indexing. This phase is applied separately on the data of each temporal slice, generated from Phase 1. Data belonging to each time slice is further partitioned using a quadtree spatial index that recursively divides space into four equal quadrants until the data contained in each quadrant is less than a specified size. For faster processing, quadtree building is performed on GPU, where multiple threads work on dividing multiple quadrants at the same time. Also, multiple threads are working on each time slice concurrently; another main advantage of using time slicing index structure in Scout. The main reason for selecting a quadtree index structure over other spatial index structures is that it handles data skewness in a way that data workload is distributed equally to different partitions; a property that is needed for GPU processing.

Phase 3: Physical indexing. As depicted in Figure 2, first two phases belong to *index building process*, where the main objective is to come up with a set of boundary partitions (obtained from each quadtree). Once we have the partition boundaries for each time slice, we do not need the tree hierarchy anymore and it is not even stored anywhere. Instead, this phase physically stores the partition information in contiguous memory locations, as needed by GPU deployment, for two main reasons: (1) It helps in reducing global memory access time by making use of coalesced memory accesses. (2) It eliminates the need of tree traversal. Instead, a parallel search is executed directly on the partition boundaries arrays. Performing tree traversal on GPUs involves executing expensive conditional and thread synchronization operations. These operations frequently lead to thread waiting, thus, under-utilizing the hardware resources.

4. QUERY PROCESSING

Scout supports basic spatio-temporal operators, described briefly in this section, namely, range, k-nearest-neighbor, and join, along with aggregation. Together, these basic operators can support a myriad of spatial analysis functions. For example, heatmaps are composed of aggregates with range queries. As it is the case with traditional spatial systems, spatial operations are executed in two phases, *filter* and *refine*. The *filter* phase is responsible for early pruning of spatial data that will not make it to the query result. The *refine* phase performs actual expensive computational geometry operations to decide on the final answer. Unlike traditional spatial systems, the query processor of Scout is specifically designed for GPUs. In particular, the GPU index lookup used in the *filter* phase exploits the contiguous storage of partition boundaries by avoiding expensive tree traversals. The compute intensive *refine* phase operations are designed to be compatible with GPU memory access and parallel processing.

Spatio-temporal range query. Such a query consists of a spatial and temporal range as query parameters, e.g., *find all Taxi drop off locations in Manhattan in June 2016. Filter* phase invokes the GPU index lookup that performs intersection operations, in parallel, between the rectangular index partition boundaries and the minimum bounding rectangle of query region. Overlapped partitions are considered as candidates that include the final answer. Data records belonging to candidate partitions are transferred to GPU for the *refine* phase. For each input data point, the *refine* phase makes use of multiple GPU threads to check if its time and location falls within the spatio-temporal query ranges.

Spatio-temporal k-NN query. An example of k-NN queries is: *find k nearest drop off locations to a certain restaurant on Jan., 1, 2017.* The *filter* phase invokes the GPU index lookup to perform parallel checks on the partition boundaries arrays to determine the partition which contain the query point i.e., query partition. The data belonging to query partition is transferred to GPU memory. The *refine* phase processes query partition to select the k nearest points using multiple threads of GPU. Once the k points are found, a circle is drawn using the distance between the query location and the k^{th} point as radius. If this circle is completely contained in the query partition, the process is deemed complete. If the circle touches other partitions, these partitions are processed to get the final answer.

Spatio-temporal join query. An example of join queries is: *find all drop-off locations within five miles of each voting site on election day.* In the *filter* phase, multiple GPU index lookups are performed by multiple thread blocks to find candidate partitions that intersect with each of the spatial query ranges. The data belonging to candidate partitions is transferred to GPU memory for *refine* phase. Each thread block performs the *refine* phase in parallel to select points that satisfy the temporal and spatial range of the query.

The query processing module also handles the case when the spatio-temporal index does not fit into GPU's global memory. The index is divided into multiple blocks such that each block fits into GPU global memory and the *filter* phase is performed in a *block-by-block* fashion.

5. VISUALIZATION

The visualization module in Scout serves three purposes: (1) It provides a map-based user interface for users to explore the data, (2) It converts user actions into spatio-temporal queries submitted to query module, and (3) It displays the results on the map in the format asked by the query (data points or heatmap). Figure 3 shows the web-based user interface of Scout for a spatio-temporal range query.

Visualization module converts user actions into four query parameters- query type, spatial attribute, spatial geometry, and temporal range. Spatial k-NN queries consist of k as the fifth parameter. As the user explores the map, every user action, that changes any one of these parameters, is converted into a new query and results are updated. For example, as user pans the map, every move produces a different spatial range (equivalent to map-space on screen) for the query and therefore, a new set of query parameters is forwarded to query processing module for producing the result. After the background processing, visualization module receives the results from query processing module. If the query required result to be in the form of data points, the result points are rendered on the screen, in parallel, with the help of OpenGL. If the results are expected to be in the form of a heatmap (e.g., Figure 4), the visualization module computes the heatmap from the result points and renders it onto the map.

6. DEMO SCENARIOS

Scout will be demonstrated to the conference audience using two main datasets, New York taxi trip data for the last six years [8] and Twitter data collected between January 2015 and May 2015. Both data sets have over a billion records. Conference attendees will be able to experience three main scenarios with Scout, namely, query execution, interactive data exploration, and background processing.

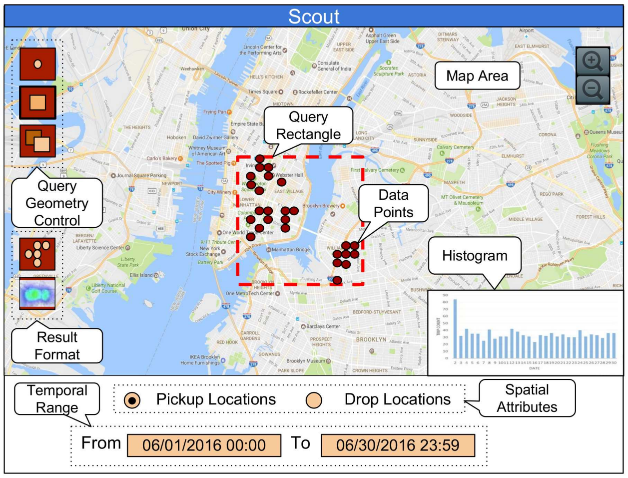

Figure 3: Query Result as Data Points

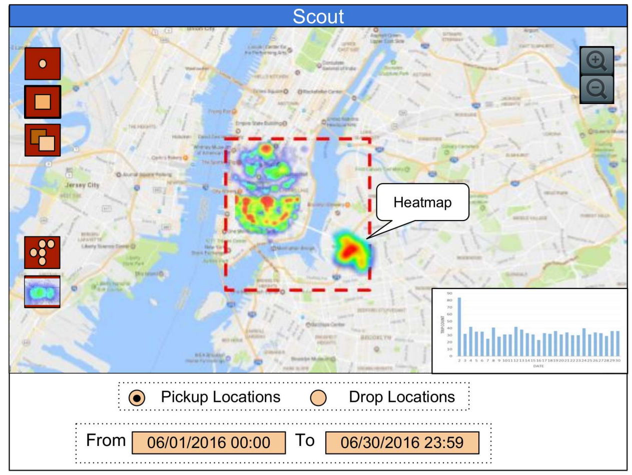

Figure 4: Query Result as Heatmap

Scenario 1: Query execution. Attendees can submit various spatio-temporal queries through Scout GUI by selecting the query type (range,k-N, and join) and output format (data points or heatmap). Results can be seen in real-time. Figure 3 shows the result of a sample spatio-temporal range query as data points and Figure 4 shows the result of the same query as a heatmap.

Scenario 2: Interactive data exploration. Interactive visualization is the main highlight of Scout. Attendees do not have to explicitly submit the queries by clicking a button. Instead, as they explore the map area or change query parameters on the user interface, the query answer is changed immediately. This will show the power of Scout system and it is ability to exploit GPU to provide smooth interactive experience.

Scenario 3: Background processing. Attendees will also be able to get a peek into the background query execution process in Scout. As the queries are submitted from the user interface, on another terminal, attendees can see the background process visualizer continuously displaying the status of the query (shown in Figure 5), e.g., the query parameters, execution time of each of the steps in query processing, result size, and total query time. It proves handy when debugging the query execution process. This will also allow demo attendees to understand the intellectual merits and technical details of Scout.

```
*********** Query Submitted ***********

--------- Visualization UI ---------
Query Submitted: 01/15/2017 12:04:33
        Query Type:             Spatio-temporal Range Query
        Spatial Range:          (-74.009634, 40.734324) to
                                (-73.977018, 40.716241)
        Temporal Range:         06/01/2016 00:00 to
                                06/30/2016 23:59
        Spatial Attribute:      Pick-up Points
        Result Format:          Point data

--------- Query Execution ---------
Identifying time slices ... (Time taken: 1msec)
Temporal Slices: Slice 8

Filter Phase initiated

Identifying spatial partitions ... (Time Taken: 5msec)
Spatial Partitions: 8:3, 8:4, 8:5

Transferring data from CPU to GPU ... (Time taken: 30msec)

Executing Refine phase ... (Time taken: 50msec)
Result size: 10035

Transferring data from GPU to CPU ... (Time taken: 20msec)

--------- Visualizing Results ---------
Rendering result ... (Time taken: 10msec)

Total Query Time: 117msec

*********** Query Completed ***********
```

Figure 5: Background Processing

7. REFERENCES

[1] H. Doraiswamy, H. Vo, C. Silva, and J. Freire. A GPU-based index to support interactive spatio-temporal queries over historical data. In *ICDE*, May 2016.

[2] N. Ferreira, J. Poco, H. Vo, J. Freire, and C. Silva. Visual exploration of big spatio-temporal urban data: A study of new york city taxi trips. *IEEE TVCG*, 19(12), 2013.

[3] L. Lins, J. Klosowski, and C. Scheidegger. Nanocubes for Real-Time Exploration of Spatiotemporal Datasets. *IEEE TVCG*, 19(12), 2013.

[4] Z. Liu and J. Heer. The effects of interactive latency on exploratory visual analysis. *IEEE TVCG*, 20(12), 2014.

[5] Z. Liu, B. Jiang, and J. Heer. immens: Real-time visual querying of big data. In *EuroVis*, 2013.

[6] A. Michotte. *The Perception of Causality*. Basic Books, 1963.

[7] M. Mokbel, T. Ghanem, and W. Aref. Spatio-temporal access methods. *IEEE Data Engineering Bulletin*, 26(2), 2003.

[8] New york taxi data. http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml.

[9] Orange cell phone records. https://www.technologyreview.com/s/514476/a-motherlode-of-cell-phone-data/.

[10] C. Stolte and P. Hanrahan. Polaris: A system for query, analysis and visualization of multi-dimensional relational databases. In *Proceedings of the IEEE Symposium on Information Vizualization 2000*, 2000.

[11] Twitter and ibm partnership. https://blog.twitter.com/2015/twitter-and-ibm-a-year-of-changing-how-business-decisions-are-made.

[12] Uber ride statistics. https://uberexpansion.com/uber-statistics-infographic/.

[13] M. Vartak, S. Rahman, S. Madden, A. Parameswaran, and N. Polyzotis. SeeDB: Efficient Data-driven Visualization Recommendations to Support Visual Analytics. *PVLDB*, 8(13), 2015.