The Case for Distance-Bounded Spatial Approximations

Eleni Tzirita Zacharatou Andreas Kipf[★] Ibrahim Sabek[★]
Varun Pandey[◇] Harish Doraiswamy[†] Volker Markl
Technische Universität Berlin MIT CSAIL[★] Technische Universität München[◇] NYU[†]
{eleni.tziritazacharatou, volker.markl}@tu-berlin.de {kipf, sabek}@mit.edu pandey@in.tum.de harishd@nyu.edu

ABSTRACT

Spatial approximations have been traditionally used in spatial databases to accelerate the processing of complex geometric operations. However, approximations are typically only used in a first filtering step to determine a set of candidate spatial objects that may fulfill the query condition. To provide accurate results, the exact geometries of the candidate objects are tested against the query condition, which is typically an expensive operation. Nevertheless, many emerging applications (e.g., visualization tools) require interactive responses, while only needing approximate results. Besides, real-world geospatial data is inherently imprecise, which makes exact data processing unnecessary. Given the uncertainty associated with spatial data and the relaxed precision requirements of many applications, this vision paper advocates for approximate spatial data processing techniques that omit exact geometric tests and provide final answers solely on the basis of (fine-grained) approximations. Thanks to recent hardware advances, this vision can be realized today. Furthermore, our approximate techniques employ a *distance-based error bound*, i.e., a bound on the maximum spatial distance between false (or missing) and exact results which is crucial for meaningful analyses. This bound allows to control the precision of the approximation and trade accuracy for performance.

1 INTRODUCTION

There is an explosion in the amount of spatial data being generated and collected today. Billions of GPS-enabled mobile devices, cars, social networks, satellites, sensors, and many other sources produce spatial data constantly. As a result of the ever-increasing data sizes and the computationally-intensive nature of spatial queries, it is hard to provide fast response times, which opposes the interactivity requirements of exploratory applications.

On the bright side, users often do not need exact results. They are instead satisfied with approximate answers, especially if these answers are accompanied by precision guarantees. However, approximate spatial data processing has attracted limited attention [3, 16, 20–22]. There are two different notions of approximation in spatial databases. Synopsis-based techniques aim to accelerate spatial queries by evaluating them on small samples of the data [16, 20, 21]. Existing techniques in this category are limited to certain types of queries (i.e., range queries, selectivity estimation, k-means clustering and spatial partitioning). On the other hand, most spatial querying techniques approximate individual spatial objects with simpler geometries such as rectangles or convex polygons to accelerate queries [5, 22]. Unlike synopsis-based techniques, geometric approximations support arbitrary spatial predicates. Our work is related to the latter category, i.e., spatial query processing based on approximations of individual objects, which is orthogonal to sampling techniques that reduce the number of objects to be processed.

Notably, prior work does not give guarantees on the *spatial distance* between false (or missing) and exact results. Consequently, it is hard to interpret the provided approximate results, as the user has no information about how closely these results correspond to the particular region she is interested in. Guaranteeing *distance-based error bounds* is thus crucial. These bounds should be controlled by the user, essentially allowing to trade off between query results accuracy and query execution time.

Motivating Application: Visual Exploration of Mobility Data. In an effort to enable urban planners to make data-driven decisions, in early 2017 Uber introduced Uber Movement, a visualization platform for the exploration of Uber rides[1]. The platform allows users to visualize data of interest at different resolutions over varying time periods. Such visual analyses require interactivity, since high latency reduces the rate at which users make observations, draw generalizations, and generate hypotheses [12]. Furthermore, exact answers are not required, because visualizations are approximate in nature. Moreover, users typically perform "level-of-detail" exploration. They first look at a high level overview, and then zoom into regions of interest for further details [15]. Finally, there is usually uncertainty with respect to spatial coordinates, as GPS positions are typically accurate to within a 4.9 m radius [19]. Similarly, geographical region boundaries are often fuzzy, in the sense that adjacent regions are separated by extended zones (e.g., a street surface) rather than one-dimensional lines. As a result, these zones can be considered to be part of any of the adjacent regions. Overall, the interactivity expected from exploratory applications (visual or not), coupled with the inherent properties of spatial data, necessitate a paradigm shift towards spatial data processing techniques that have approximation at their core.

Hardware Trends. Spatial approximations have been widely used in spatial databases. Recent hardware trends, however, indicate that the time has come to rethink their design and utility. Existing techniques typically use a two-step "filter and refine" strategy [5] where approximations are only employed in a first filtering step that yields a candidate result set. The subsequent refinement step eliminates false matches by performing exact geometric tests. The underlying assumption behind this approach is that main memory is scarce and secondary storage accesses are slow. Consequently, approximations are assumed to be stored in secondary storage and have to be compact to minimize the amount of accesses. To achieve a compact representation, approximation precision is sacrificed.

Today's machines, however, have large main memory sizes that can go up to multiple terabytes and are often equipped with Non-Volatile Memory (NVM) that offers fast access combined with large storage capacity. As a result of the faster access, the filtering step is no longer in the critical path, and the CPU-intensive refinement step

[1] https://www.uber.com/newsroom/introducing-uber-movement-2/

becomes the bottleneck. As recent work shows [7], the CPU-based filtering step takes only a few milliseconds even for billions of points. To improve performance, we need to increase the precision (and thus size) of spatial approximations in exchange for better filtering efficacy, thereby reducing the number of costly refinements or even completely eliminating them.

With increasing data sizes, the computation of precise approximations becomes more expensive. However, to support exploratory applications where the workload changes dynamically, we need to compute spatial approximations fast and on-the-fly. GPUs, and in particular their native support for *rasterization* make that possible today. The rasterization operation takes as input a geometric primitive (e.g., a polygon) and converts it into a collection of pixels which essentially form a fine-grained uniform grid approximation of the primitive. GPUs perform rasterization at interactive speeds, as they employ highly optimized hardware implementations. This enables us to design techniques that leverage GPUs to compute spatial approximations and evaluate spatial queries in real time.

This vision paper argues that fine-grained grid approximations can form the basis of spatial data processing. We show that these approximations allow to provide distance-based error bounds, enable us to exploit modern hardware, and facilitate further optimizations such as the use of learned indexes. The remainder of this paper outlines our vision of incorporating distance-bounded spatial approximations in different components of a spatial system, highlights individual challenges, and presents promising initial results.

2 APPROXIMATE PROCESSING

In this section, we first present geometric approximations commonly used in spatial data processing. We then describe how we can quantify the error that these approximations introduce. Finally, we discuss the benefits of integrating distance-bounded spatial approximations in different components of a spatial system.

2.1 Geometric Approximations

Spatial objects can have an arbitrarily complex structure. Even worse, different spatial objects can have very different structures (e.g., a point is different from a polygon). To address this challenge, spatial query processing algorithms perform geometric tests (e.g., intersection, containment) on approximations of the geometries [5]. The employed approximations can represent objects with different geometries and retain the objects' main features. In addition, they have a significantly simpler structure than the actual objects, which reduces computation and storage costs.

The most widely used spatial object approximation is the Minimum Bounding Rectangle (MBR), which is the smallest axis-aligned rectangle that encloses the complete geometry of an object (Figure 1(a)). MBRs are rather rough and inaccurate approximations. Being more accurate than MBRs, raster approximations have recently attracted attention. They represent geometric primitives using a set of cells that can be either equi-sized [18, 22] (Uniform Raster, Figure 1(b)) or variable-sized [9] (Hierarchical Raster, Figure 1(c)). For a detailed study of spatial approximations, see [5].

Executing spatial queries on geometric approximations leads to approximate results that are typically further processed to obtain exact answers. However, when the geometric approximation is sufficiently precise and exact answers are not required, approximate query processing techniques can provide final answers solely on the basis of the approximate geometries. In this paper, we advocate for approximate techniques with application-driven accuracy and discuss next how to bound the approximation error.

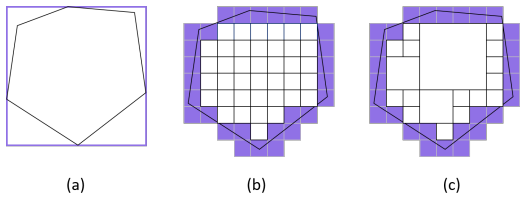

Figure 1: Three example geometric approximations of a polygon: (a) Minimum Bounding Rectangle (MBR), (b) Uniform Raster (UR), (c) Hierarchical Raster (HR).

2.2 Distance Bound

Spatial queries involve predicates that evaluate relations among objects in space (e.g., intersection, containment). Therefore, we argue that it is only natural for *approximate* techniques to provide distance-based error bounds, i.e., *guarantees* on the spatial distance between false (or missing) and exact results. Approximate results without this notion of spatial distance can be misleading and hard to interpret. To illustrate this, consider the example in Figure 2. It shows a set of points corresponding to the pickup location (latitude/longitude) of taxi rides. To optimize its operational planning, the taxi service provider needs to compute the count of trips that originate from within a given region % depicted in the figure. The exact count of taxis is 18. Consider now two approximate results. The first one is computed over the set of black and red points and equals to 22, while the second one is computed over the set of black and violet points and equals to 28. Although the first aggregate result is closer to the exact value, it contains points which are quite far away from the region % that the user is interested in, while it does not include the violet points that are closer to %. We argue that for such exploratory analyses, the second result is more meaningful as it matches more closely the user's region of interest. We further argue that in order to interpret the obtained approximate result, the user needs information about the spatial distance between the data points from which the approximate result was derived and the query geometry. In other words, it is often admissible for the user to compute the result over a region that closely approximates %, as long as she knows how close in space the approximation is.

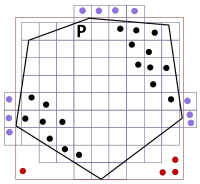

Figure 2: Example polygon and points and two approximations of the polygon, MBR (red), and Uniform Raster (violet).

Formally, a geometry $6'$ n-approximates a geometry 6 if the Hausdorff distance $3\ \ (6, 6')$ between the two geometries is at most n, where

$$3\ \ (6, 6') = \max\left\{\max_{?' \in 6'} \min_{? \in 6} 3(?, ?'), \max_{? \in 6} \min_{?' \in 6'} 3(?', ?)\right\}$$

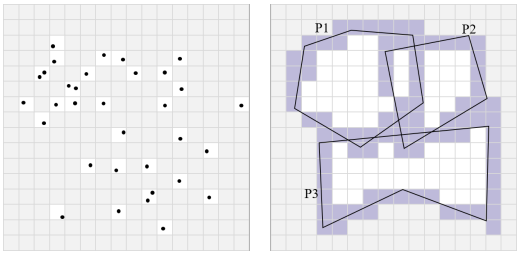

Figure 3: Uniform Raster approximation of points (left) and polygons (right). Figure from [18].

and $\delta(?', ?)$ denotes the Euclidean distance between two points. Intuitively, this ensures that any false positive (false negative) results that are present (absent) when answering queries using the approximate geometry G' are within a distance ε from the boundaries of the original geometry G.

Interestingly enough, not all geometric approximations can be distance-bounded. The spatial extent of MBRs is data dependent: MBR corners are the convergence points of the dimension-wise maxima/minima of the object they bound. Consequently, the distance between a corner and the closest point in the object boundary can be very large.

Raster approximations, on the other side, can be distance-bounded. Given ε, raster approximations such as the ones shown in Figure 1, can guarantee that $\delta(G, G') \leq \varepsilon$ by using a cell side length equal to $\varepsilon' = \frac{\varepsilon}{\sqrt{2}}$ (i.e., the length of the diagonal of the cell is ε) for the cells that are at the boundary of the geometry (shown with violet color). The interior cells that are fully contained in the geometry can have a cell side length larger than ε' as they do not contribute to the approximation error. At the boundary, there can be two types of errors, depending on the implementation. If all the cells that overlap even the slightest with the boundary are part of the approximation, then there can only be *false positive* results as the whole cells are considered to be part of the object. We call such a raster approximation *conservative*. In non-conservative raster approximations, the cells that have a small overlap with the boundary can be omitted, which can introduce *false negative* results. Overall, the precision of raster approximations is independent of the geometry they approximate and *tunable*. This property makes them particularly suitable to form the basis of approximate spatial query processing techniques.

2.3 The Power of Distance-Bounded Raster Approximations

To illustrate the power of distance-bounded raster approximations, consider the example in Figure 3 showing two input data sets, a set of points (left) and a set of polygons (right) approximated with UR.
Indexing. Figure 3 essentially shows how the data is represented *logically*: geometric objects are approximated by a set of cells, potentially along with additional information that denotes the cells that intersect with the geometry boundaries. Given this representation, a database system needs efficient indexes to store the approximations and enable their fast retrieval. Since approximate query processing eliminates the expensive refinement step, the index lookup performance is crucial because it determines the query performance. Traditional R-tree-based indexes [2] are not applicable as they are designed to index MBRs and are not compatible with raster approximations. At the same time, raster approximations enable new opportunities for a new generation of indexes. Specifically, mapping the cells to an one-dimensional array by enumerating them with a space-filling curve, enables the use of a *learned index* [10]. As we show in Section 3, by learning the position of the cells in the 1D array, the learned index outperforms other spatial index structures.
Optimization. Section 4 discusses how, by abstracting away from the specific object geometries and providing a *unified* representation for different geometric data types, the raster approximation creates new opportunities in spatial query optimization. That is, the implementation of primitive operations (e.g., intersection tests) on the raster approximation can be *independent* of the geometries and thus re-usable, while it can also leverage modern GPUs.
Execution. Other than enabling efficient access to a single data set, the raster approximation also enables the efficient execution of queries that involve multiple data sets, such as joins. As we show in Section 5, by mapping geometries to sets of cells, we can observe the overlap at the cell level instead of performing geometry-to-geometry comparisons. Each cell can be processed independently, which makes the computation highly parallelizable. Furthermore, aggregations that are distributive or algebraic can be computed very efficiently. The final aggregate can be obtained by combining partial aggregates calculated (in parallel) for each cell.

In the following, we describe how to use distance-bounded raster approximations in various system components in more detail and present initial results.

3 DATA ACCESS

Storage layouts and index structures determine the efficiency of data access. This section shows the details of how we can build high-performing indexes for polygon and point geometries that leverage raster approximations.
Dimensionality Reduction. While raster cells could be indexed using spatial data structures such as a Quadtree, a *linearization* step can simplify the indexing problem significantly. A common approach is to map 2D cells into a 1D domain by enumerating them with a space-filling curve such as the Hilbert or Z curve. As we will show, we can achieve much higher lookup performance with linearized cells, even compared to well-tuned 2D spatial indexes [13].
Polygon Indexing. Adaptive Cell Trie (ACT) [9] is a recently proposed radix tree data structure designed for indexing linearized cells of hierarchical raster approximations. A radix tree has a clear advantage over a B+tree or a sorted array in this setting. That is, matching cells can be found in any level of the tree and larger cells are indexed closer to the root. Hence, larger cells are likely to be found sooner during the tree traversal. In addition, the radix tree offers implicit prefix compression as keys are not stored explicitly.

To index a set of polygons in ACT, we first perform a hierarchical raster approximation of the polygons that conforms to a user-defined distance bound (Section 2.2). ACT uses the IDs of the linearized cells to build the radix tree. To find a matching polygon for a query point, we first transform the query point to a cell on the most fine-grained grid level. Then, we traverse the radix tree with the query cell of this point and retrieve the ID of the matching polygon (if such a polygon exists).

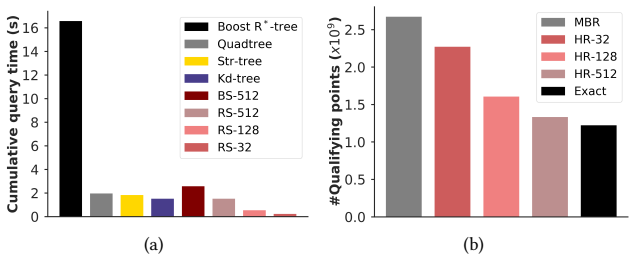

Figure 4: Data access efficiency. (a) Point-polygon containment query performance. (b) Impact of the precision of the raster approximation on the number of qualifying points.

Point Indexing. Like polygons, points are traditionally indexed with spatial data structures such as R-trees. Here, we propose to apply the same linearization for mapping 2D points to 1D cell identifiers. This again simplifies the indexing problem potentially leading to large speedups as we will demonstrate. We store the resulting 1D cell identifiers (corresponding to 2D points) in a data structure such as a B+-tree or simply in a *sorted* array.

To query the points with a polygon, we first approximate the query polygon using a hierarchical raster approximation, which yields a set of non-overlapping variable-sized cells that we call *query cells*. Then, for each cell, we perform a binary search on the sorted array to get the qualifying points. For aggregation queries (e.g., COUNT, SUM), one can pre-compute a prefix sum array and simply perform a lower and an upper bound lookup with the query cell's boundaries. By subtracting the lower bound from the upper bound, we can compute the aggregate value. In this setting, the time for computing both lower and upper bounds (essentially a binary search each) really matters. Therefore, we also explore using a learned index to speed up these searches.

We employ RadixSpline (RS) as a learned index [10]. RS consists of two main components: i) a set of spline points, and ii) a radix table to quickly determine the spline points to be examined for a lookup key (i.e., the query cell in our case). At lookup time, we first consult the radix table to determine an initial range of spline points. Next, this range is searched over to determine the spline points surrounding the lookup key. Finally, we use linear interpolation to predict the position of the lookup key in the sorted array. Building the RS requires only one pass over the data, and is thus efficient.

Performance. We experimentally compare the performance of our proposed RS-based index with binary search (BS) and other four spatial indexes, namely, R*-tree [2] from Boost Geometry [1], kd-tree [4], Quadtree [8], and STR-packed R-tree [11]. The spatial indexes act as baselines for filtering based on the MBR approximation. In our experiment, we use 39,200 polygons corresponding to the NYC Census regions (query polygons) and 1.2B points from the NYC taxi data set (years 2009 to 2016) [17]. We implemented the kd-tree, Quadtree, and STR-packed R-tree baselines based on recent research [13]. For the Boost R*-tree, we chose the bulk-loading mode and manually optimized the number of elements per node. This experiment was run single-threaded on a two-socket Arch Linux 5.7.4 machine with an Intel Xeon Gold 6230 Processor CPU (2.10 GHz, 10 cores, 3.90 GHz turbo) and 256 GB DDR3 RAM.

Figure 4(a) shows the cumulative query time to find the total number of points inside the query polygons, while varying the precision of the raster approximation (i.e., number of approximating cells per query polygon). We compared the results of three RS-based index variations, corresponding to three precision levels (32, 128, and 512 cells per polygon), with binary search at the highest precision level used (i.e., 512) and the other four spatial baselines. Note that the spatial baselines use MBR filtering, and hence they are agnostic to changing the precision level. Clearly, the three RS-based variations outperform both Boost R*-tree and BS baselines (at least 10× and 35% better than Boost R*-tree and BS, respectively). For the kd-tree, Quadtree, and STR-packed R-tree baselines, the RS-based variations are still either better or very close to them in terms of query time. However, as shown in Figure 4(b), RS-based variations are significantly better in terms of finding the tightest number of qualifying points compared to the exact number (precision level of 512 is almost similar to the exact case). Thus, in summary, our proposed RS-based index hits a sweet spot in the trade-off between precision and query time compared to all other baselines.

4 QUERY OPTIMIZATION

Existing approaches for spatial query processing are tied to specific geometric data representations and closely follow the relational model for query optimization [14]. They use operators that are tightly coupled to specific geometric types and query classes. Let us consider again the selection query from Figure 2. As mentioned earlier, this query is typically implemented as a *single* operator that uses two phases: filtering and refinement. While the filtering phase relies on MBRs and is thus generic, the refinement phase depends on the geometric type and operation. In this example, the refinement is specific to the input being points, and the performed operation is a point-in-polygon test. If the input changes from taxi pickup locations to restaurants represented by polygons, then a different implementation is required, since a polygon-intersect-polygon test must be performed instead. The use of such large monolithic operators limits the set of options over which optimization can be performed, as the operators cannot be reused across query classes.

To overcome these limitations, and to exploit modern GPUs, a GPU-friendly spatial data model and algebra was introduced in [6], that proposes a uniform data representation called *canvas* and a small set of simple parallelizable operators. These operators include common computer graphics operations: *blend*, *mask*, and *affine transformations*. More importantly, these operators are sufficient to realize common spatial query classes without being tied to specific geometries. For instance, both point-polygon and polygon-polygon intersection tests boil down to applying a combination of the above operations on the canvas. We propose to adapt the canvas model to support distance-bounded approximate queries: the canvas now simply becomes a rasterized image, where the pixel size depends on the required bound. The GPU-amenable operators work directly on such a *rasterized canvas*—in fact, the implementation of these operators now becomes straightforward since boundary conditions [6] need not to be taken care of. There are two ways to generate a rasterized canvas: by rendering the data directly on the GPU, or through the use of indexes (e.g., using ACT described in Section 3).

The rasterized canvas along with the proposed set of operators enable the creation of multiple alternative plans to realize any given ad-hoc query, thereby adding flexibility in the optimization process.

Furthermore, each operator can have multiple implementations and indexes can be reused across operators, which provides a wider set of options for the optimizer. Thus, optimizers can choose to use different query plans based on the query parameters, the distance bound (i.e., the resolution of the rasterized canvas), and the estimated selectivity. As an example of the potential gains that our proposed model provides, we show in Section 5.2 how the model allows for an alternate plan for an approximate spatial aggregation query that performs significantly faster than traditional approaches.

5 QUERY EXECUTION

This section highlights the benefits of distance-bounded raster approximations in query evaluation. As a representative example, we focus on the evaluation of spatial aggregation queries defined as follows in SQL-like notation:

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

Given a set of points of the form $P(p, O_1, O_2, \ldots)$, where p and O_8 are the location and attributes of the point, and a set of regions $R(83, 64, 4CA)$, this query performs an aggregation (AGG) over the result of the join between P and R. The geometry of a region can be any *arbitrary polygon*. Functions such as COUNT($*$) or AVG(O_8) are commonly used for AGG.

This query typically uses point-in-polygon (PIP) tests to identify polygons that contain each of the points. Note that each PIP test requires time linear with respect to the size of the polygon. Since real-world polygonal regions often consist of hundreds of vertices, these tests are computationally intensive. This challenge is compounded due to the fact that data sets can have hundreds of millions, or even billions of points, requiring a large number of PIP tests to be performed.

Existing systems typically evaluate spatial aggregation queries by performing a spatial join of the points and the polygons, followed by the aggregation of the join results. To reduce the number of PIP tests, the join is first solved using MBR approximations. As we show next, our evaluation strategies that are based on raster approximations, outperform the above approach significantly.

5.1 Main-Memory Join

Using our ACT index (Section 3), we can evaluate the query with an index-nested loop join: we simply index the polygons with ACT, and query the radix tree for every point. We combine the join with the aggregation to avoid materializing the join result. Given that ACT employs a fine-grained distance-bounded HR approximation, we omit the PIP tests and provide approximate results.

Performance. We experimentally compare the performance of our approximate join with the Boost [1] R*-tree [2] and Google's S2ShapeIndex (SI)[2], all implemented in C++. ACT uses HR polygonal approximations satisfying a 4m distance bound. The R-tree indexes the polygons' MBRs, while, similarly to ACT, SI uses HR approximations. However, SI's approximation is not distance-bounded and SI does not support approximate evaluation. We use 1.2B points from the NYC taxi data set [17], and three NYC polygon data sets: Boroughs (5), Neighborhoods (289), and Census (39,200).

[2]https://s2geometry.io/devguide/s2shapeindex

This experiment was run single-threaded on a machine with 14-core Intel Xeon E5-2680 v4 CPUs and 256 GB DDR4 RAM. Figure 5 shows that the ACT-based approximate join significantly outperforms other approaches. It is over one order of magnitude faster than SI in all cases. Compared to the R*-tree, it brings over two orders of magnitude improvement for Boroughs, and over one order otherwise. The low performance of the R*-tree for Boroughs is due to the fact that Boroughs are complex polygons and thus PIP tests are expensive. Therefore, reducing the number of those tests by approximating the polygons more closely (as SI does) or completely eliminating them by using distance-bounded fine-grained approximations (like ACT) has a significant impact on performance.

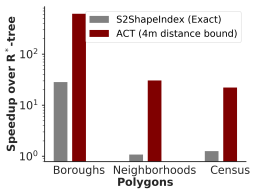

Figure 5: Main-memory join.

5.2 GPU Join

Section 4 outlined the use of a rasterized canvas model for executing spatial queries on GPUs. Here we show the gains that the proposed model brings in the evaluation of spatial aggregation queries. In fact, the query can be realized by simply combining a small set of operators from our query algebra on top of the rasterized canvas model. This is exactly what our recently proposed algorithm, Bounded Raster Join [18] (BRJ), does. Intuitively, BRJ takes as input a uniform representation of the points and polygons on rasterized canvases. It then merges (using the *blend* operator) all the points into a single canvas that maintains partial aggregates, i.e., each canvas pixel keeps the aggregate of all points falling in that pixel. Then, it joins this canvas with the set of polygon canvases (by composing the *blend* and *mask* operators) to identify points that intersect with the polygons, and finally merges the results (using a combination of *transformations* and *blending*) to compute the final aggregate. That is, the aggregates from the individual pixels that fall within a polygon are combined to generate the aggregation for that polygon. The precise query plan can be found in [6]. The above operations are natively supported by the graphics pipeline, leading to orders of magnitude speedup over typical evaluation strategies on CPUs without requiring any pre-computation as we showed in [18].

Performance. We implemented BRJ using C++ and OpenGL. We create the canvases on-the-fly by simply rendering the geometries onto an off-screen buffer and store the aggregates in the buffer's color channels (r,g,b,a). We experimentally compare BRJ with an accurate GPU Baseline that follows the traditional index-based evaluation strategy of first filtering the polygons with a grid index (with 1024^2 cells) and then performing PIP tests. This experiment was run on a machine with an Intel Core i7 Quad-Core CPU, 16GB RAM, and an NVIDIA GTX 1060 mobile GPU with 6GB of memory, out of which we use only 3GB. We join 600M points of the NYC taxi data set [17] (transferred in batches to the GPU) with 260 NYC neighborhood polygonal regions (some of the regions are multi-polygons) and count the number of points in each region. Figure 6 shows that there is a trade-off between the accuracy and the query time.