A GPU-friendly Geometric Data Model and Algebra for Spatial Queries

Harish Doraiswamy
New York University
harishd@nyu.edu

Juliana Freire
New York University
juliana.freire@nyu.edu

ABSTRACT

The availability of low cost sensors has led to an unprecedented growth in the volume of spatial data. Unfortunately, the time required to evaluate even simple spatial queries over large data sets greatly hampers our ability to interactively explore these data sets and extract actionable insights. While Graphics Processing Units (GPUs) are increasingly being used to speed up spatial queries, existing solutions have two important drawbacks: they are often tightly coupled to the specific query types they target, making it hard to adapt them for other queries; and since their design is based on CPU-based approaches, it can be difficult to effectively utilize all the benefits provided by the GPU. As a first step towards making GPU spatial query processing mainstream, we propose a new model that represents spatial data as geometric objects and define an algebra consisting of GPU-friendly composable operators that operate over these objects. We demonstrate the expressiveness of the proposed algebra and present a proof-of-concept prototype that supports a subset of the operators, which shows that it is orders of magnitude faster than a CPU-based implementation and outperforms custom GPU-based approaches.

ACM Reference Format:
Harish Doraiswamy and Juliana Freire. 2020. A GPU-friendly Geometric Data Model and Algebra for Spatial Queries. In *Proc. 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3318464.3389774

SIGMOD'20, June 14–19, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6735-6/20/06…$15.00
https://doi.org/10.1145/3318464.3389774

1 INTRODUCTION

The availability of low cost sensors such as GPS in vehicles, mobile and IoT devices have led to an unprecedented growth in the volume of spatial data. Extracting insights from these data sets requires the ability to effectively and efficiently handle a variety of queries.

The most common approach to support spatial queries is through the use of spatial extensions that are available in existing relational database systems (e.g., the PostGIS extension for PostgreSQL [24], Oracle Spatial [22], DB2 Spatial Extender [1], SQL Server Spatial [28]). Popular geographic information system (GIS) software typically use these systems to process spatial queries [2, 14, 25] and some also provide their own database backend. Using these state-of-the-art systems, the response times to even simple spatial queries over large data sets can runs into several minutes (or more), hampering the ability to perform interactive analytics over these data [11, 19]. While faster response times can be attained by powerful clusters [10, 23], such an option, due to its costs and complexity, is often out of reach for many analysts.

Modern Graphics Processing Units (GPUs) provide a cost-effective alternative to support high-performance computing. Since GPUs are widely available, even in commodity laptops, effective GPU-based solutions have the potential to democratize large-scale spatial analytics. Not surprisingly, several approaches have been proposed that use GPUs to speed up spatial queries (e.g., [5, 7, 32–34]). However, these typically follow the traditional approaches that were designed primarily for the CPU, and simply porting the algorithms may lead to an ineffective use of GPU capabilities. For example, a spatial aggregation query that combines input points across different polygonal regions would typically be implemented as a spatial join of the points and polygons followed by the aggregation of the join results. On the other hand, the recently proposed RasterJoin [30] represented a departure from traditional strategies: by modeling spatial aggregation queries using GPU-specific operations, it attained significant speedups even over the traditional query plan executed on a GPU, suggesting that GPU-specific strategies can lead to substantial performance gains.

Another drawback of traditional GPU-based approaches is that they require different implementations for each query class. Consequently, it is hard to re-use and/or extend them

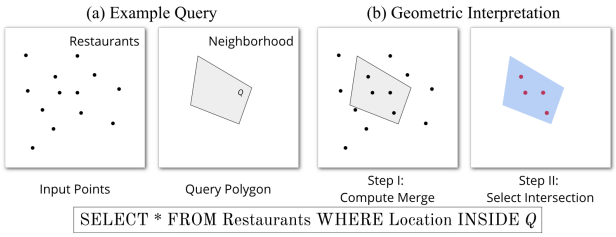


Figure 1: Reformulation of the spatial selection query.

for other similar queries. For example, a GPU-based implementation for the selection query in Figure 1, such as STIG [7], cannot be used if the input representation of the data (restaurants in this example) is changed from points to polygons. This shortcoming, coupled with the complexities involved in implementing GPU-based solutions, has impeded a wider use of GPUs in spatial databases.

Our Approach. With the goal of enabling GPUs to be exploited without the need to build custom solutions, we revisit the problem of designing a spatial data model and operators. We adapt common computer graphics operations for which GPUs are specifically designed and optimized, to propose a *new geometric data model that provides a uniform representation for different geometric objects*, and an *algebra consisting of a small set of composable operators capable of handling a wide variety of spatial queries*.

While several data models and algebras have been described in the literature [8, 13, 15, 17, 26], they were all designed before the advent of modern GPUs and suffer from at least one of the above shortcomings, as we discuss in Section 8. Furthermore, these models are typically *user facing*: users express the queries of interest by making use of the data types and the operators provided in the model; the implementation of the operators is left to the developer. In contrast, we aim for a *developer-facing* model that can be incorporated into existing systems unbeknownst to the users, while at the same time providing significant benefits to the database engine and query performance.

To give an intuition behind the proposed model, consider the example in Figure 1(a) from a geometric point of view. The query can be translated into two operations performed one after the other as shown in Figure 1(b). Graphically, the input points and the query polygon are uniformly represented as *drawings on a canvas*. The first operation merges the input points and the query polygon into a single canvas. The second operation computes the intersection between the points and the polygon to eliminate points outside the polygon. Unlike the traditional execution strategy for spatial aggreagtion (i.e., join followed by aggregation), the two operations used here are applicable to any kind of geometry. Therefore, even if the data (restaurants) were represented as polygons instead of points, the same set of operations could be applied.

Informally, we represent a spatial object as an embedding of its geometry onto a plane, called a *canvas*, and define GPU-friendly operators, similar to the ones in Figure 1(b), that act on one or more canvases. As we show in Section 4, these operators can be re-used and composed to support a diverse set of spatial queries.

Given a small set of basic operators, our model makes it possible for implementations to focus on the efficiency of these operators, the gains from which become applicable to a variety of queries. While our focus in this paper is on the conceptual representation and modeling of spatial data and the design of the algebra, this work opens new avenues for research in spatial query optimization, both for theory (e.g., developing plan generation strategies, designing cost models) and systems (e.g., designing indexes leveraging GPUs).

Contributions. To the best of our knowledge, this is the first approach to propose a query algebra designed with a focus on enabling an efficient GPU realization of spatial queries. Our main contributions are as follows:

- We propose a new geometric data model that provides a uniform representation for spatial data on GPUs (Section 2).
- We design a spatial algebra consisting of five fundamental operators designed based on common computer graphics operations (Section 3). We show that the algebra is: expressive and able to represent all standard spatial queries; and closed, allowing the operators to be composed to construct complex queries (Section 4).
- We present a proof-of-concept implementation of a subset of the proposed operators that shows how: 1) the proposed model and operators are naturally suited for GPUs; and 2) operators can be re-used in different queries. Our implementation achieves over two orders of magnitude speedup over a custom CPU-based implementation, and outperforms custom GPU-based approaches (Sections 5 & 6).
- We discuss the compatibility of the proposed algebra with the relational model and its utility for query optimization (Section 7).

An extended version of this paper can be found in [6].

2 DATA REPRESENTATION

In this section, we first formalize the notion of a spatial data set, and then define the concept of a *canvas*, the spatial analogue of a relational tuple.

2.1 Spatial Data

As discussed in Section 1, an important limitation of current spatial operations is that they are tied to specific representations for geometric data types. To design flexible operators, we need a uniform representation for the geometry that is independent of underlying types. We propose to schematically

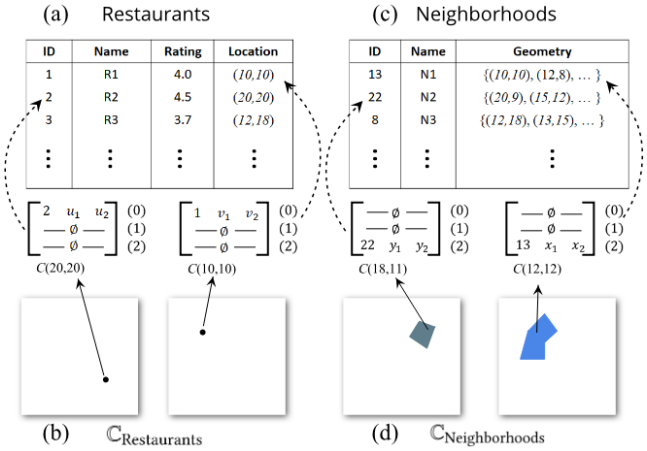


Figure 2: *Canvas* representation of spatial data. (a) Example point data. (b) Two canvases corresponding to the first two records of the table. The [0,0] element of the matrix (corresponding to the 0-primitive) stores the unique ID corresponding to the record. (c) Example polygon data. (d) Two canvases corresponding to the first two records – all points inside a polygon map to the same value, with the element [2,0] storing the unique ID. The white parts of the canvas (not part of the geometry) maps to a null value.

represent the geometry using a single type called *geometric object*, which can conceptually represent any complex geometric structure.

Definition 1 (Geometric Object). *A geometric object is defined as a collection of geometric primitives.*

Definition 2 (Geometric Primitive). *A d-dimensional geometric primitive (d-primitive) is defined as a d-manifold (with or without a boundary).*

Informally, a d-manifold is geometric space in which the local neighborhood of every point represents $\mathbb{R}^d$. In the context of spatial data that is of interest in this work, we focus on 2-dimensional (2D) space and d-primitives, where $0 \leq d \leq 2$. Intuitively, a 0-primitive is a point while a 1-primitive is a line (not necessarily a straight line or with a finite length). 2-primitives include any subset of $\mathbb{R}^2$ that is neither a line nor a point, such as polygons and half spaces.

A spatial data set can now be defined in terms of geometric objects as follows:

Definition 3 (Spatial Data). *A spatial data set consists of one or more attributes of type geometric object.*

Note that the above definition allows geometric objects of arbitrarily complex shapes composed using a heterogeneous set that contains points, lines, as well as polygons. However, geometric objects common in real world data sets are primarily only points (e.g., locations of restaurants, hospitals, bus stops, etc.), only lines (e.g., road networks), or only polygons (e.g., state or city boundaries).

2.2 Canvas

We define the notion of a canvas to explicitly capture the geometric structure of a spatial data set. As mentioned above, we assume that the dimensions of the geometric primitives composing a geometric object in a spatial data set is either 0, 1 or 2. Let S be a set of k-tuples, where $k \geq 1$, such that $\emptyset \in S$. A *canvas* is formally defined as follows:

Definition 4. *(Canvas) A canvas is a function $C : \mathbb{R}^2 \rightarrow S^3$ that maps each point in $\mathbb{R}^2$ to a triple $(s[0], s[1], s[2]) \in (S \times S \times S)$, where the i^{th} element of the triple, $s[i]$, stores information (as a k-tuple) corresponding to i-dimensional geometric primitives.*

Definition 5. *(Empty Canvas) A canvas C is empty iff C maps all points in $\mathbb{R}^2$ to $(\emptyset, \emptyset, \emptyset)$.*

A canvas is analogous to a tuple in the relational model. It is defined to capture geometric objects in the world coordinate space of the graphics pipeline, thus making it straightforward to apply computer graphics operations on them. Intuitively, a canvas stores for each point in $\mathbb{R}^2$ information corresponding to the geometric primitives that intersect with that point. This information is captured by the elements of the set S (we discuss S in more detail below).

Given a spatial data set, each record in the data is represented using one or more canvases – one canvas per geometric attribute in the record. For ease of exposition, consider a data set having a single geometric attribute and a geometric object o corresponding to one of the records in this data. Let $o = \{g_1, g_2, g_3, \ldots, g_n\}$, where g_i is a geometric primitive having dimension $dim(g_i)$, $0 \leq dim(g_i) \leq 2$, $\forall i$. A canvas representation of the geometric object o is defined as follows.

Definition 6. *(Canvas representation of a geometric object) A canvas corresponding to a geometric object is a function $C_o : \mathbb{R}^2 \rightarrow S^3$ such that $\forall d \in [0, 2]$*

$$C_o(x, y)[d] = \begin{cases} s_d \neq \emptyset \in S, & \text{if } \exists i \mid dim(g_i) = d \text{ and} \\ & \quad g_i \text{ intersects } (x, y) \\ \emptyset & \text{otherwise} \end{cases}$$

The set S used in the above definition is called the *object information set*, and is defined as follows.

Definition 7. *(Object Information Set S) The object information set S is defined as a set of triples (v_0, v_1, v_2) where v_0 stores a unique identifier (or a pointer) for the record corresponding to the geometric object. v_1 and v_2 are real numbers storing meta data related to the canvas.*

The range of the canvas function C can thus be represented as a 3×3 matrix, where each row corresponds to the Object

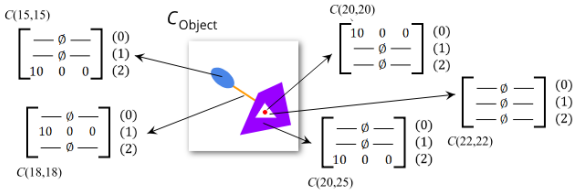


Figure 3: A canvas representing a complex object. Since all the primitives (colored differently) are part of the same object, they have the same ID.

Information Set for the associated primitive dimension. We abuse notation to represent the triple $(\emptyset, \emptyset, \emptyset)$ simply as $\emptyset$.

Example 1: Consider the two example data sets in Figure 2. The first data set corresponds to the set of restaurants in a city (represented as points) (a), while the second data set corresponds to the neighborhood boundaries of this city (represented as polygons) (c). Figure 2 also illustrates the canvas representations corresponding to two records from each of these two data sets. Note that in this example, we use only the identifier element of the object information set. The values of the other elements are initialized depending on the query scenario (Section 4).

Example 2: The complex geometric object shown in Figure 3 consists of two polygons (an ellipse and a polygon with a hole) connected by a line, with the hole also containing a point. This is represented in the canvas by mapping the regions corresponding to the different primitives using the appropriate rows in the matrix (for point, line, polygon).

3 OPERATORS

Below, we define the operators we designed for the canvas representation of a spatial data set. We give concrete examples of how these operators are used in Section 4. We categorize the set of operators into three classes: fundamental, derived, and utility operators. We use the following notation to represent operators that take as input zero or more canvases:

$$Op[P_1, P_2, \ldots](C_1, C_2, \ldots, C_n)$$

where Op is the operator name, $P_i, \forall i$, the parameters of the operator, and $C_j, \forall j$, the canvases input to the operator. The output of all the operators is always one or more canvases. Thus, *the proposed algebra is closed by design.*

3.1 Fundamental Operators

Fundamental operators form the core of the proposed algebra. Their design is based on common computer graphics operations that are already supported in GPUs. Figure 4 illustrates the five fundamental operators.

Geometric Transform $C' = \mathscr{G}[\gamma](C)$**:** This operator takes as input a single canvas C and outputs a canvas C' in which

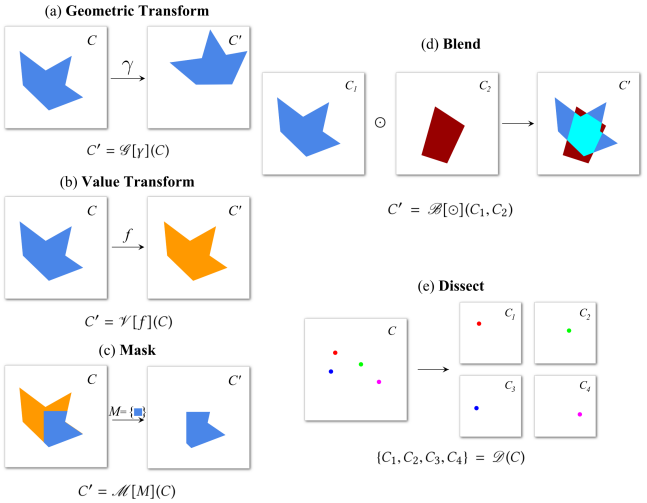


Figure 4: The five fundamental operators. Color is used to illustrate the information stored in each point of the canvas.

all the geometric objects of C are transformed to a new position in C' defined by the parameter function γ. Here, the parameter function γ can be defined in two ways: (1) $\gamma : \mathbb{R}^2 \rightarrow \mathbb{R}^2$; and (2) $\gamma : S^3 \rightarrow \mathbb{R}^2$. In the first case, the new position (x', y') of a geometry is dependent on its current position (x, y), i.e., $C'(\gamma(x, y)) = C(x, y)$. Examples of such functions include operations such as rotation, translation, etc. The example in Figure 4(a) rotates and translates (moves) the polygon object to a different position.

A scenario where this operator is useful is when spatial data sets in a database use different coordinate systems. Thus, when performing binary or n-ary operations on canvases from these data sets, the geometry has to be converted into a common coordinate system first. The parameter function γ can be defined appropriately for this purpose.

In the second case, the new position (x', y') of the geometry is dependent on the information stored at the current position $C(x, y)$, i.e., $C'(\gamma(C(x, y))) = C(x, y)$. Such a transformation is useful, for example, to accumulate values (e.g., for aggregation queries) corresponding to a geometric object—in this case, the function γ can be defined to move all points having the same object identifier to a unique location.

Value Transform $C' = \mathscr{V}[f](C)$**:** This unary operator outputs a canvas C' in which the information corresponding to the geometries is modified based on the parameter function f. That is, $C'(x, y) = f(x, y, C(x, y))$, where, $f : \mathbb{R}^2 \times S^3 \rightarrow S^3$ is a function that changes the object information based on its location and/or value. Figure 4(b) illustrates an example of this operation where the color of the polygon in the canvas is changed from blue to orange.

Mask $C' = \mathscr{M}[M](C)$**:** The mask operator is used to filter regions of canvas so that only regions satisfying the condition

specified by $M \subset S^3$ are retained. Formally, the application of this operator results in the canvas C' such that

$$C'(x,y) = \begin{cases} C(x,y), & \text{if } C(x,y) \in M \\ \emptyset & \text{otherwise} \end{cases}$$

For example, this can be used to accomplish select intersection operations shown in Figure 1(b) and Figure 4(c).

Blend $C' = \mathscr{B}[\odot](C_1, C_2)$**:** Blend is a binary operator used to merge two canvases into one. The blend function $\odot$:$S^3 \times S^3 \rightarrow S^3$ defines how the merge is performed:$C'(x,y) = C_1(x,y) \odot C_2(x,y)$. The merge operation used in Figure 1(b) is an instance of the blend function. Another example is shown in Figure 4(d).

Dissect $\{C_1, C_2, \ldots, C_n\} = \mathscr{D}(C)$**:** The dissect operation splits a given canvas into multiple non-empty canvases, each corresponding to a point $(x,y) \in \mathbb{R}^2$ having $C(x,y) \neq \emptyset$. That is, a new canvas C_i is generated corresponding to a non-null point (x,y) such that

$$C_i(x',y') = \begin{cases} C(x,y), & \text{if } (x',y') = (x,y) \\ \emptyset & \text{otherwise} \end{cases}$$

For example, in Figure 4(e), a canvas encoding 4 points is split into 4 canvases each corresponding to one of those points. As we show later, one of the uses of this operator is for queries involving aggregations over geometries with 1- and 2-primitives (such as polygons).

3.2 Derived Operators

It is common for certain combinations of fundamental operators to be repeatedly used for various queries. We represent these combinations as derived operators and describe a couple of such operators below.

Multiway Blend $C' = \mathscr{B}^*[\odot](C_1, C_2, \ldots, C_n)$**:** This n-ary operator takes as input n canvases and generates a single canvas after blending all these n canvases in the given order, i.e., $C' = \mathscr{B}[\odot](C_1, \mathscr{B}[\odot](C_2, \mathscr{B}[\odot](C_3, \ldots)))$. Note that if the blend function $\odot$ is associative, then it allows relaxing the grouping of the different blend operations, thus providing more flexibility while optimizing queries.

Map $\{C_1, C_2, \ldots, C_n\} = \mathscr{D}^*[\gamma](C)$**:** Map is a composition of a dissect followed by a geometric transform: $\{C_1, C_2, \ldots, C_n\} = \mathscr{G}[\gamma](\mathscr{D}(C))$. This operator is useful to align all the canvases resulting from the dissect. In such a case, γ is typically defined as a constant function: $\gamma(x,y) = (x_c, y_c)$, where x_c and y_c are constants.

Without loss of generality, the above notation of providing multiple canvases as input to a unary operator (in this case the geometric transform that takes as input canvases output from a dissect operation) is considered the same as applying the operator individually to each of the input canvases.

3.3 Utility Operators

Utility operators are primarily used to generate canvases based on a set of parameters. These are particularly useful for classes of spatial queries involving parametric constraints. In particular, we consider the following three types of utility operators:

Circle $C = Circ[(x,y), r]()$**:** generates a canvas corresponding to a circle with center is (x,y) and radius r.

Rectangle $C = Rect[l_1, l_2]()$**:** generates a canvas corresponding to a rectangle having diagonal end points l_1 and l_2.

Half Space $C = HS[a,b,c]()$**:** generates a canvas representing the half space defined by the equation: $ax + by + c < 0$.

4 EXPRESSIVENESS

To demonstrate the expressiveness of the proposed model, in what follows we describe how common spatial queries can be represented as algebraic expressions. We build upon the classification of spatial queries used by Eldawy et al. [10] and categorize spatial queries into the following classes: *selection*, *join*, *aggregate*, *nearest neighbor*, and *geometric queries*. Note that this is a super set of the query types evaluated in a state-of-the-art experimental survey by Pandey et al. [23].

For ease of exposition, we consider only point and polygonal data sets. It is straightforward to express similar queries for other types of spatial data sets with lines, or more complex geometries. Without loss of generality, we assume that the different operators prune empty canvases from their output, and an empty canvas is generated when an input canvas does not satisfy a given constraint. This is similar to the relational model, where tuples that do not match the query constraints are excluded from the output table.

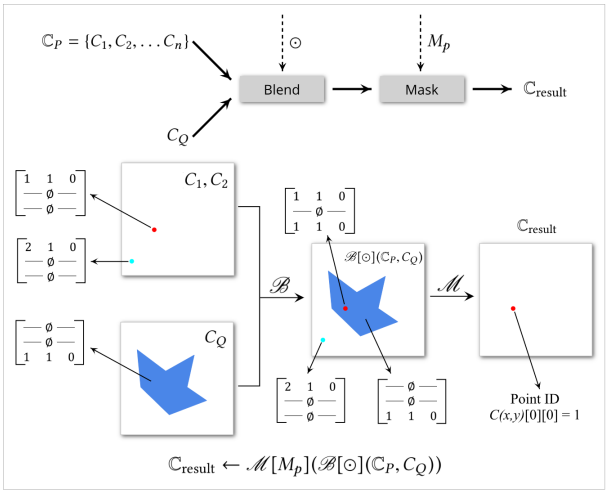
4.1 Selection Queries

We can classify spatial selection queries into three types: polygonal selection, range selection, and distance-based selection. We first consider selection queries that have a polygonal constraint, and then extend the algebraic expressions for other types of selection queries.

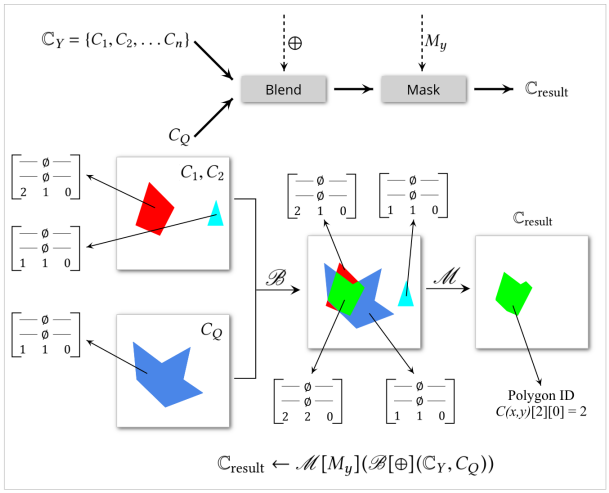
Polygonal Selection of Points. Let D_P be a data set consisting of a set of points. Let $\{(x_1,y_1), (x_2,y_2), \ldots, (x_n,y_n)\}$ be the coordinates corresponding to the location of these points. Let Q be any arbitrary-shaped polygon. Consider the following spatial query expressed in an SQL-like syntax:

```
SELECT * FROM D_P WHERE Location INSIDE Q
```

Note that this is the same query used for the example in Figure 1(a). Using the proposed data representation, let $\mathbb{C}_P = \{C_1, C_2, \ldots C_n\}$ be the set of canvases corresponding to each point (record) in D_P. Let the canvas C_i corresponding to the i^{th} record be defined as follows:



Polygonal selection of points



Polygonal selection of polygons

Figure 5: (Top) A schematic representation (plan diagram) of algebraic expressions used for polygonal selection queries. (Bottom) Illustration of the different steps in these plans on points (left) and polygons (right), with two objects colored red and cyan respectively. For simplicity, these objects are shown within a single canvas.

$$C_i(x,y)[0] = \begin{cases} (id, 1, 0) & \text{if } (x,y) = (x_i, y_i) \\ \emptyset & \text{otherwise} \end{cases}$$
$$C_i(x,y)[1] = C_i(x,y)[2] = \emptyset$$

Here, *id* corresponds to the unique identifier mapping the canvas to the corresponding record in D_p. We use the second element of the tuple $C_i(x,y)[0]$ to keep count of the points incident on the location (x,y), which in this case is 1. The third element is ignored for this query. Let the canvas C_Q corresponding to the query polygon Q be defined as follows:

$$C_Q(x,y)[0] = C_Q(x,y)[1] = \emptyset$$
$$C_Q(x,y)[2] = \begin{cases} (1,1,0) & \text{if } (x,y) \text{ falls inside } Q \\ \emptyset & \text{otherwise} \end{cases}$$

Similar to the case of points above, the elements $C_Q(x,y)[2][0]$ and $C_Q(x,y)[2][1]$ stores the *id* of the query polygon (set to 1 since there is only one polygon) and count of 2-primitives incident on (x,y) respectively. Using the canvases defined above, the select query can be algebraically expressed as: $\mathbb{C}_{result} \leftarrow \mathscr{M}[M_p](\mathscr{B}[\odot](\mathbb{C}_P, C_Q))$, where, $\forall s_1, s_2 \in S^3$

$$s_1 \odot s_2 = \begin{bmatrix} s_1[0][0] & s_1[0][1] & s_1[0][2] \\ - & \emptyset & - \\ s_2[2][0] & s_2[2][1] & s_2[2][2] \end{bmatrix}$$

and $M_p = \{s \in S^3 \mid s[0] \neq \emptyset \text{ and } s[2][0] = 1\}$.

Similar to the example in Figure 1(b), the above expression first merges the input data with the query polygon using the blend operator $\mathscr{B}$, and then uses the mask operator $\mathscr{M}$ to select only the intersection (a location is part of the intersection if both, a 1-primitive and 2-primitive are incident on it). Figure 5(left) visualizes the above expression as a plan diagram, and illustrates the different steps for two examples: when a point is inside the query polygon (and hence part of the result), and when a point is outside respectively.

Polygonal Selection of Polygons. Let D_Y be a data set consisting of a set of polygons. Let $\{Y_1, Y_2, \ldots, Y_n\}$ be the set of polygons associated with each record of the data set. As before, the polygons can have any shape. Let Q be another arbitrary-shaped polygon. Let the set of canvases $\mathbb{C}_Y$ corresponding to polygons in D_Y be defined as follows:

$$C_i(x,y)[0] = C_i(x,y)[1] = \emptyset$$
$$C_i(x,y)[2] = \begin{cases} (id, 1, 0) & \text{if } (x,y) \text{ falls inside } Y_i \\ \emptyset & \text{otherwise} \end{cases}$$

Let the canvas corresponding to query polygon Q be defined as before. Now, consider the following selection query, similar to the one above, but over D_Y:

```
SELECT * FROM D_Y WHERE Geometry INTERSECTS Q
```

This query can be algebraically expressed as:
$\mathbb{C}_{result} \leftarrow \mathscr{M}[M_y](\mathscr{B}[\oplus](\mathbb{C}_Y, C_Q))$, where, $\forall s_1, s_2 \in S^3$

$$s_1 \oplus s_2 = \begin{bmatrix} - & \emptyset & - \\ - & \emptyset & - \\ s_1[2][0] & s_1[2][1] + s_2[2][1] & s_1[2][2] \end{bmatrix}$$

and $M_y = \{s \in S^3 \mid s[2][1] = 2\}$.

Note that unlike in the previous case of selecting points, since the data as well as the query consist of polygons, both the data canvas and the query canvas store information only for 2-primitives. Hence, the second element of the information tuple is made use of in this case to compute the

intersection (i.e., locations having two 2-primitives incident on them). Figure 5(right) shows the algebraic expression using a plan diagram, and illustrates two examples denoting selection and non-selection scenarios respectively.

Selection Using Other Spatial Constraints. Selection queries having other types of spatial constraints, such as a range query or distance-based query can be accomplished by replacing the query polygon with the utility operators [6].

4.2 Join Queries

Conceptually, the algebraic expression for joins is the same as the corresponding selection queries, with the exception that a single query polygon is instead replaced with a collection of polygons. Similar to the join operator in the relational model, spatial joins using our model can be implemented in several ways. A straightforward approach is using nested loops for the blend operation, which can be made more efficient if spatial indexes are available. We describe the types of join queries supported by the model in the extended paper [6].

4.3 Aggregate Queries

The third class of queries common for spatial data are spatial aggregation queries. We consider two types of such queries: aggregating the results from a selection, and the aggregation required for a group-by over a join. Consider first a simple count of the results from a selection query:

```
SELECT COUNT(*) FROM D_P WHERE Location INSIDE Q
```

This query can be realized using the expression: $C_{\text{count}} \leftarrow \mathscr{B}^*[+](\mathscr{G}[\gamma_c](\mathbb{C}_{\text{result}}))$, where $\gamma_c : S^3 \rightarrow \mathbb{R}^2$ is defined as $\gamma_c(s) = (s[2][0], 0), \forall s \in S^3$; $+ : S^3 \times S^3 \rightarrow S^3$ is defined as

$$s_1 + s_2 = \begin{bmatrix} 0 & s_1[0][1] + s_2[0][1] & 0 \\ - & \emptyset & - \\ s_2[2][0] & s_2[2][1] & s_2[2][2] \end{bmatrix}$$

and $\mathbb{C}_{\text{result}} \leftarrow \mathscr{M}[M_p](\mathscr{B}[\odot](\mathbb{C}_P, C_Q))$ is the set of canvases resulting from the select operation (same as in Section 4.1). See [6] for an in-depth analysis of this expression and its extension to other aggregation types, as well as to joins.

4.4 Nearest-Neighbor Queries

Consider a kNN query given a query point X. This is accomplished by first computing the distance r such that there are exactly k points within the circle centered at X with radius r, and then performing a distance-based selection query to obtain the kNN query result. See [6] for the algebraic expressions to accomplish such queries.

5 NOTES ON IMPLEMENTATION

We now briefly describe a proof-of-concept GPU-based implementation of our model to demonstrate its advantages with respect to enabling the reuse of operators. In particular, we discuss the blend and mask operators required to implement spatial selection queries. To further illustrate the expressive power of our model, in [6], we also examine the spatial aggregation operation proposed in [30], and show how it translates directly into an algebraic expression.

5.1 Proof-of-Concept Prototype

The prototype was implemented using C++ and OpenGL. It uses the traditional representation of point and polygon data sets, that is, they are stored as a set of tuples, The required canvases are then created on the fly when a query is executed.

Data Representation. Recall from Section 2 that geometric objects are modeled as a union of smooth manifolds, and a canvas representing these objects is defined as a scalar function over $\mathbb{R}^2$. Given such a continuous formal representation, it is therefore important to have a discrete representation to be used in the implementation. Our choice was to maintain a canvas as a *texture* [27], which corresponds to a collection of pixels. Here, each pixel stores the object information triple. The canvas functions are defined as discussed in Section 4.1. However, since the pixels discretize the space, it is also necessary to store additional data corresponding to the geometry boundaries. In the case of points, this additional information corresponds to the actual location of the points. For polygons, we store a flag that is set to true if the pixel is on the boundary of the polygon. To accurately identify all boundary pixels, we use an OpenGL extension that enables conservative rasterization. This identifies and draws all pixels that are touched by a triangle (or line), and is different from the default rasterization, wherein a pixel is only drawn when >50% of the pixel is covered by the primitive. This ensures that the border pixels are tracked in a conservative fashion, and hence there is no loss in accuracy. Additionally, a simple index is maintained that maps each boundary pixel to the actual vector representation of the polygon.

The canvases are generated by rendering the geometry onto a texture using the graphics pipeline. The color components (r,g,b,a) are used to store the canvas function. To handle polygons with holes, the outer polygon is first drawn onto the texture. The inner polygon (representing one or more holes) is then drawn such that the pixels corresponding to it are negated (i.e., the canvas function is set to null).

Operators. The *blend operator* is accomplished through a straightforward *alpha blending* [27] of two textures, which is supported as part of the graphics pipeline. The *mask operator* looks up each pixel of the texture in parallel and tests for the mask condition. Note that here, the boundary information is used to perform an accurate test if the point is part of a pixel that is on the boundary of the polygon.

Alternate Implementations. Another possible implementation is to represent geometric objects as a collection of simplicial complexes, thus avoiding any rasterization. The operators can then be implemented to make use of the native ray tracing support provided by the latest RTX-based Nvidia GPUs. We decided to use the rasterization pipeline instead so that our prototype could support any modern GPU from multiple vendors, and not just the RTX GPUs from Nvidia.

Queries. *Polygonal selection of points* is accomplished by first creating the canvases corresponding to the query polygon and query points, which are blended together and then filtered using the mask operator. The operator functions are as defined previously. Our implementation, without any modification, also works for *polygonal selection of polygons*, i.e., if the input is changed from a set of points to a set of polygons.

A straightforward variation of the selection query is to *support multiple polygons as part of the constraint.* In particular, consider the case when the constraint requires the input point to be inside at least one of the polygons (a disjunction). This query can be expressed using our model as follows using just the blend and mask operators: $\mathbb{C}_{\text{result}} \leftarrow \mathscr{M}[M_{p'}](\mathscr{B}[\odot](\mathbb{C}_P, \mathscr{B}^*[\oplus](\mathbb{C}_Q)))$. Here, $\mathbb{C}_Q$ is the collection of canvases corresponding to the query polygons, and $\odot$ and $\oplus$ are the blend functions defined in Section 4.1. The above expression first blends together all the query constraint polygons into a single canvas, which is then used to perform the select similar to the single polygon case. The mask function $M_{p'}$ is defined as: $M_{p'} = \{s \in S^3 \mid s[0] \neq \emptyset \text{ and } s[2][0] \geq 1\}$. Recall that the mask function M_p used for the single query polygon case tests the incidence of the polygon on a pixel by testing the *id* field of the function value corresponding to 2-primitives. Instead, this is accomplished using $M_{p'}$ by checking if the count of the polygons incident on the pixel is at least one. Thus, this mask function $M_{p'}$ is valid even when there is only a single query polygon.

6 EXPERIMENTAL EVALUATION

We now briefly discuss the performance of the spatial selection queries using the prototype described above. All experiments were run on a *laptop* having an Intel Core i7-8750H processor, 16 GB memory and 512 GB SSD. The laptop has a dual Nvidia GTX 1070 Max-Q GPU with 8 GB graphics memory, and an integrated Intel UHD Graphics 630 GPU.

Data and Queries. The main goal of our evaluation is to: 1) demonstrate the advantage of using GPU-friendly operators compared to previous GPU-based solutions; and 2) illustrate how the same operators can be used for variations of a given query. We do this using selection queries that select trips from the New York City's taxi data having their pickup location within a query polygon. To demonstrate point (2), we also use queries having a disjunction of multiple polygonal constraints. The size of the input is varied using the pickup time range of the taxi trips.

To mimic real-world use cases, all query polygons used in these queries were "hand-drawn" using a visual interface (e.g., [12]) and adjusted to have the same MBR. We then use as input only taxi trips that have their pickup location within this MBR. In other words, the evaluation assumes the existence of a filtering stage and focuses on the refinement step. We decided to do this for two reasons. First, the refinement stage, and not filtering, is now the primary bottleneck. Unlike previous decades, due to the existence of fast ssd-based storage and large CPU memory, filtering takes only a small fraction of the query time. For example, the filtering step used by the state-of-the-art GPU-based selection approach, even though it is CPU-based, takes only a few milliseconds even for data having over a billion points [7]. Second, when working with complex queries, depending on the query parameters, the optimizer need not always choose to use the spatial index corresponding to a spatial parameter, and the spatial operations could be further up in the plan (e.g., the optimizer might to choose first filter based on another attribute, say time, before performing a spatial operation). In such scenarios, the spatial operation will not have the benefit of an index-based filtering, and the bottleneck would be the refinement step. Additionally, the above setup also helps remove input bias when comparing the performance across polygonal constraints having different shapes and sizes.

Approaches. We compare our approach with a CPU baseline, a parallel CPU implementation using OpenMP, as well as a GPU baseline. Because of the above mentioned experimental setup that eliminates the effect of indexes used by current state-of-the-art, we only need to implement the PIP tests for the baselines. While our approach was executed on two different GPUs (denoted as Nvidia and Intel), the GPU baseline was executed only on the faster Nvidia GPU.

Performance. Figures 6(a) and 6(c) show the speedup achieved by the different approaches over a single threaded CPU implementation, for queries that have one and two polygonal constraints, respectively. Note that while all GPU-based approaches are over two orders of magnitude faster than the CPU-based approach, the speedup of our approach increases when the number of polygonal constraint increases. This is because, the only additional work done by our approach when there are additional polygons is to blend the constraint polygons. This is significantly less work when compared to existing approaches which have to perform more PIP tests in this case. This is corroborated by the query run times in Figures 6(b) and 6(d), wherein our approach (in red) requires only 4 seconds (using the Nvidia GPU) even when there are two polygons as constraints and the query

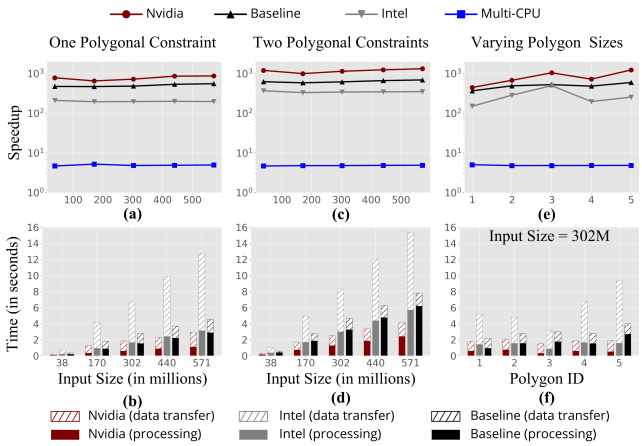


Figure 6: Performance evaluation.

MBR has as many as 571M points. For a given input and GPU, not only is the time to transfer data between the CPU and GPU similar, but is also a significant fraction of the query time. In this light, the speedup in the processing time achieved using our model over a traditional GPU-based approach (which is greater than the overall speedup depicted in Figures 6(a) & (c)) clearly demonstrates the advantages of using a GPU-friendly approach.

Figures 6(e) and 6(f) show the speedup and running times when the shapes (and sizes) of the polygonal constraint is varied. The query selectivity varies roughly between 3% to 83%. While there is some variation in the processing time depending on the complexity of the polygon constraint, this variation is higher for the baseline. This is because the number of PIP tests performed by the baseline is linearly proportional to the size of the polygon. Irrespective of this complexity, our approach using the discrete GPU requires at most 2 seconds even when the MBR has 302M points.

Also interesting to note is the performance of our approach on the integrated Intel GPU. While, as expected, it is slower than the GPU baseline using a Nvidia GPU, it is still over 2-orders of magnitude faster than the CPU implementation. Given that these GPUs are present in mid-range laptops, ultrabooks, and even tablets, our algebraic formulation can potentially allow fast spatial queries even on such systems.

7 DISCUSSION

Interoperability with Relational Model. The proposed model is compatible with the relational model and can be incorporated into existing relational systems. Recall that the minimalistic definition of S used in Section 2.2 reserves the first element of the triple to store the unique ID corresponding to the data record. Thus, given a set of canvases corresponding to existing data sets, it is possible to switch to the corresponding relational tuple using this ID. Analogously, the storage structure of a relational tuple can be changed to link to the corresponding canvas, thus allowing connection in the opposite direction. Alternatively, similar to our proof-of-concept implementation, the canvases could also be created on demand.

Thus, conceptually, one can consider the relational tuple and a canvas to be the *dual* of each other allowing a seamless use of the two representations by a query optimizer to appropriately generate query plans involving both spatial and non-spatial operators.

Query Optimization. The proposed model facilitates query optimization in the following ways.

1. Allowing different query execution plans. The model enables the creation of multiple plans to realize a complex query. We gave two such examples in the previous section—one for disjunction and the other for spatial aggregation. In all such scenarios, by appropriately modeling the cost functions of the operators together with metadata about the input, the optimizer can choose a plan that has a lower cost.

2. Supporting diverse implementations. It is also possible to have multiple implementations of the same operators, for example, using pre-built spatial indexes. Each of the indexes would result in a different cost based on the properties of the data and the query, thus providing a rich set of options over which to perform the optimization.

3. Enabling general query processing. Given the duality between the canvas and the relational tuple as discussed above, the proposed operators can also be easily plugged into existing query optimizers, thus allowing for complex queries involving both the spatial and relational attributes.

Limitations. While the proposed data representation can be directly extended to support 3D primitives, the operators over such 3D data do not have a straightforward implementation using the GPU. Given that native ray tracing support is now being introduced in GPUs, it would be interesting to explore extensions to our algebra that make use of such advances to support 3D spatial queries.

8 RELATED WORK

In this section, we review prior work that proposed models and algebras for spatial queries. Güting [15] introduced geo-relational algebra, which extends relational algebra to include geometric data types and operators. The geometric data types included points, lines, and polygons (without holes) and the geometric operators included operations that are now common in most spatial database solutions (containment, intersection, perimeter, area, etc.). Aref and Samet [3, 4, 26] generalized the above model and provided one of the first high-level discussions on integrating spatial and non-spatial data to build a spatial database system, and the challenges involved in designing a query optimizer for such a system. Note that current spatial extensions follow

approaches very similar to the ideas proposed in these works. The model they use is *user facing*, i.e., the queries of interest are expressed making use of the data types and the operators provided in the model. The implementation of the operators, however, is left to the developer, and often devolves into having separate implementations for each data type/query combination. Our approach, on the other hand, which uses a single representation for all spatial data types and set of GPU-friendly operators different from the traditional operators, is *developer facing* and *primarily meant to help database developers implement an efficient GPU-based spatial query engine*: it can be incorporated into existing systems unbeknownst to the user while at the same time providing significant benefits to the database engine and query performance.

Different from the extended relational models, Egenhofer and Franzosa [8] proposed a model that uses concepts from point set topology for spatial queries. They model spatial data objects (of a single type) as closed sets, and uses the topological relationship between pairs of closed sets to answer spatial queries. Egenhofer and Sharma [9] showed the equivalence of the above model to a raster space, thus making it suitable for GIS queries involving raster data. Kainz et al. [18] model the same topological relations as above, but using partially ordered sets. While theoretically elegant, there are three main shortcomings of this topological approach: (1) the topological relationships are tied to a particular data type and cannot be used on complex spatial objects; (2) computing the relationships requires costly intersection tests between all pairs of spatial objects, making the approach untenable for large data sets; and more importantly, (3) distance-based queries, such as distance-based selections/joins and nearest neighbors, cannot be expressed using this model.

Gargano et al. [13] proposed an alternate model that represents spatial objects as a set of rectangular regions. The spatial queries are then realized as operations over these sets. To avoid loss of accuracy, very small rectangles must be used, resulting in high memory overheads and expensive set operations, thus limiting its practical applicability.

Güting and Hartmut proposed the Realms model [16] and the corresponding ROSE algebra [17]. A Realm models spatial data as a planar graph, where the nodes correspond to points on an integer grid. Given the hardware limitations during that era, the goal of this approach was to avoid costly floating point operations and any imprecision in the query computation. As data is inserted into the database, the spatial objects are "redrawn" to ensure topological consistency. This framework has important limitations. First, due to the distortion caused by the redrawings, distance-based queries now become imprecise. Second, it is necessary for all query parameters to be a part of the Realm. Thus, when generating dynamic queries (common in several data analysis tasks), the query parameters have to first be *temporarily* inserted into the Realm resulting in numerous redrawrings. Not only is this expensive, but it also results in further distortions since these redrawings are not undone when the query parameter is removed from the Realm. Third, queries involving spatial objects outside the Realm boundaries are not possible. Finally, it is difficult to support complex spatial objects consisting of multiple primitive types.

All of the above models/algebras were designed before GPUs became mainstream, and thus an implementation of these models using GPUs is non-trivial (they are difficult to parallelize, involve iterative algorithms like intersection computations, etc.). Our approach, on the other hand, was designed with GPUs in mind, and uses computer graphics operations for which GPUs are optimized.

Models have also been proposed that focus on moving objects [21] which are orthogonal to our work. For GIS applications, Tomlin [29] proposed the Map algebra which was then extended to support time by Jeremy et al. [20]. The map algebra was designed to enable cartographers to easily specify common cartographic functions. Voisard and David [31] propose a layered model specific to geographic maps to help users build new maps. From an implementation point of view, all of the above operations can be translated into spatial queries for execution, and thus an efficient spatial model will be useful in such scenarios as well.

9 CONCLUSION

In this paper, we introduced a GPU-friendly data model and algebra to support queries over spatial data sets. Key novel contributions in this work include a representation that captures the geometric properties inherent in spatial data and the design of an algebra of GPU-friendly operators that can be applied directly on the geometry. We have shown that the proposed algebra is expressive and able to realize common spatial queries. In addition, since the algebra is closed, it can also be used to construct complex queries by composing the operators. The potential ease of implementation afforded by our approach, coupled with its performance (even on commodity hardware), can greatly influence the design as well as adoption of GPU-based spatial database solutions, thus democratizing real-time spatial analyses over large data. This is corroborated by the results of the experimental evaluation carried out with our proof-of-concept prototype.

This work opens opportunities for new research in the area of spatial query optimization, both for the development of new theory, including algorithms for plan generation and cost estimation, and systems that use the algebra to efficiently evaluate queries over the growing volumes of spatial data.

Acknowledgements. This work was partially supported by the DARPA D3M program, the NYU Moore Sloan Data Science Environment, and NSF award CCF-1533564.

REFERENCES

[1] David W. Adler. 2001. DB2 Spatial Extender - Spatial Data Within the RDBMS. In *Proc. VLDB*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 687–690.

[2] ArcGIS 2018. ArcGIS. https://www.arcgis.com/.

[3] Walid G. Aref and Hanan Samet. 1991. Extending a DBMS with Spatial Operations. In *Proc. SSD*. Springer-Verlag, London, UK, UK, 299–318.

[4] Walid G. Aref and Hanan Samet. 1991. Optimization for Spatial Query Processing. In *Proc. VLDB*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 81–90.

[5] B. Bustos, O. Deussen, S. Hiller, and D. Keim. 2006. A Graphics Hardware Accelerated Algorithm for Nearest Neighbor Search. In *Proc. ICCS*, V. N. Alexandrov, G. D. van Albada, P. M. A. Sloot, and J. Dongarra (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 196–199.

[6] Harish Doraiswamy and Juliana Freire. 2020. A GPU-friendly Geometric Data Model and Algebra for Spatial Queries: Extended Version. arXiv:2004.03630 [cs.DB]

[7] H. Doraiswamy, H. T. Vo, C. T. Silva, and J. Freire. 2016. A GPU-based index to support interactive spatio-temporal queries over historical data. In *Proc. ICDE*. IEEE, 1086–1097.

[8] Max J. Egenhofer and Robert D. Franzosa. 1991. Point-set topological spatial relations. *Int. J. Geogr. Inf. Syst.* 5, 2 (1991), 161–174.

[9] Max J. Egenhofer and Jayant Sharma. 1993. Topological relations between regions in R^2 and Z^2. In *Adv. Spatial Databases*, David Abel and Beng Chin Ooi (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 316–336.

[10] A. Eldawy and M. F. Mokbel. 2016. The Era of Big Spatial Data: A Survey. *Found. Trends databases* 6, 3-4 (Dec. 2016), 163–273.

[11] Jean-Daniel Fekete and Claudio Silva. 2012. Managing Data for Visual Analytics: Opportunities and Challenges. *IEEE Data Eng. Bull.* 35, 3 (2012), 27–36.

[12] Nivan Ferreira, Jorge Poco, Huy T Vo, Juliana Freire, and Cláudio T Silva. 2013. Visual exploration of big spatio-temporal urban data: A study of new york city taxi trips. *IEEE TVCG* 19, 12 (2013), 2149–2158.

[13] M Gargano, E Nardelli, and M Talamo. 1991. Abstract data types for the logical modeling of complex data. *Information Systems* 16, 6 (1991), 565 – 583.

[14] GRASS 2018. GRASS GIS. https://grass.osgeo.org/.

[15] R. H. Güting. 1988. Geo-Relational Algebra: A Model and Query Language for Geometric Database Systems. In *Proc. EDBT*. Springer-Verlag, London, UK, UK, 506–527.

[16] Ralf Hartmut Güting and Markus Schneider. 1993. Realms: A foundation for spatial data types in database systems. In *Adv. Spatial Databases*, David Abel and Beng Chin Ooi (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 14–35.

[17] Ralf Hartmut Güting and Markus Schneider. 1995. Realm-based spatial data types: The ROSE algebra. *VLDBJ* 4, 2 (01 Apr 1995), 243–286.

[18] Wolfgang Kainz, Max J. Egenhofer, and Ian Greasley. 1993. Modelling spatial relations and operations with partially ordered sets. *Int. J. Geogr. Inf. Syst.* 7, 3 (1993), 215–229.

[19] Zhicheng Liu and J. Heer. 2014. The Effects of Interactive Latency on Exploratory Visual Analysis. *IEEE TVCG* 20, 12 (2014), 2122–2131.

[20] Jeremy M., Roland V., and C. D. Tomlin. 2005. Cubic Map Algebra Functions for Spatio-Temporal Analysis. *CaGIS* 32, 1 (2005), 17–32.

[21] Jan Kristof Nidzwetzki and Ralf Hartmut Güting. 2017. Distributed secondo: an extensible and scalable database management system. *Distributed and Parallel Databases* 35, 3 (01 Dec 2017), 197–248.

[22] Oracle-Spatial 2018. Oracle Spatial and Graph. https://www.oracle.com/technetwork/database-options/spatialandgraph/documentation/spatial-doc-idx-161760.html.

[23] Varun Pandey, Andreas Kipf, Thomas Neumann, and Alfons Kemper. 2018. How Good Are Modern Spatial Analytics Systems? *PVLDB* 11, 11 (July 2018), 1661–1673.

[24] PostGIS 2018. PostGIS: Spatial and Geographic objects for PostgreSQL. http://postgis.net.

[25] QGIS 2018. QGIS. https://www.qgis.org/en/site/.

[26] H. Samet and W. G. Aref. 1995. Spatial Data Models and Query Processing. In *Modern Database Systems*, Won Kim (Ed.). ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 338–360.

[27] Dave Shreiner, Graham Sellers, John M. Kessenich, and Bill M. Licea-Kane. 2013. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.3* (8th ed.). Addison-Wesley Professional.

[28] SQL-Spatial 2018. SQL Server Spatial. https://docs.microsoft.com/en-us/sql/relational-databases/spatial/spatial-data-sql-server?view=sql-server-2017.

[29] C D. Tomlin. 1994. Map algebra: one perspective. *Landscape and Urban Planning* 30, 1-2 (1994), 3–12.

[30] 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.

[31] A. Voisard and B. David. 2002. A database perspective on geospatial data modeling. *IEEE TKDE* 14, 2 (March 2002), 226–243.

[32] Jianting Zhang and Simin You. 2012. Speeding up large-scale point-in-polygon test based spatial join on GPUs. In *Proc. BigSpatial*. 23–32.

[33] Jianting Zhang, Simin You, and Le Gruenwald. 2012. High-performance online spatial and temporal aggregations on multi-core CPUs and many-core GPUs. In *Proc. DOLAP*. 89–96.

[34] Jianting Zhang, Simin You, and Le Gruenwald. 2012. *High-Performance Spatial Join Processing on GPGPUs with Applications to Large-Scale Taxi Trip Data.* Technical Report. The City College of New York.