SnapToQuery: Providing Interactive Feedback during Exploratory Query Specification

Lilong Jiang Arnab Nandi
Department of Computer Science & Engineering
The Ohio State University
{jianglil, arnab}@cse.osu.edu

ABSTRACT

A critical challenge in the data exploration process is discovering and issuing the "right" query, especially when the space of possible queries is large. This problem of exploratory query specification is exacerbated by the use of interactive user interfaces driven by mouse, touch, or next-generation, three-dimensional, motion capture-based devices; which, are often imprecise due to jitter and sensitivity issues. In this paper, we propose *SnapToQuery*, a novel technique that guides users through the query space by providing interactive feedback during the query specification process by "snapping" to the user's likely intended queries. These intended queries can be derived from prior query logs, or from the data itself, using methods described in this paper. In order to provide interactive response times over large datasets, we propose two data reduction techniques when snapping to these queries. Performance experiments demonstrate that our algorithms help maintain an interactive experience while allowing for accurate guidance. User studies over three kinds of devices (mouse, touch, and motion capture) show that *SnapToQuery* can help users specify queries quicker and more accurately; resulting in a query specification time speedup of 1.4× for mouse and touch-based devices and 2.2× for motion capture-based devices.

1. INTRODUCTION

Ad-hoc data exploration is an important paradigm for users to analyze data and gain insight from it. Given the increasing availability and collection of data, there is a trend towards data-driven decision making in all fields from business to science. Businesses now hire data scientists to help them to sift through, explore, and analyze data to derive better business insights. In fields such as the sciences and medicine, practitioners are turning to "data-driven" methods in experiments, producing large amounts of data with the hope of deriving scientific insight through data exploration. The exploratory querying of data is typical even for non-expert users, with general-purpose use cases such as planning a trip and searching for a nearby satisfactory restaurant. Data exploration has also become inherently more interactive – as with modern web applications and direct manipulation interfaces, users have come to expect interactive interfaces that quickly respond to a user's actions and allow for a fluid experience.

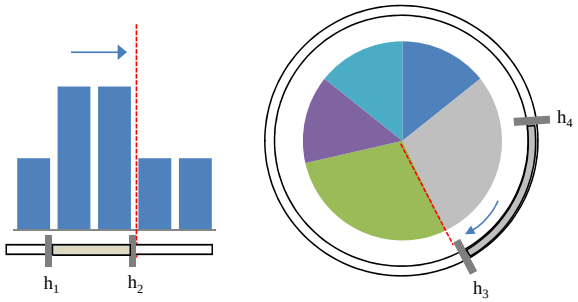


Figure 1: Providing visual and interactive feedback for exploratory `WHERE` Queries. When exploring the query space, the currently manipulated handles (h_2 and h_3) will snap to the red line.

In terms of user interfaces for exploratory querying, there has been a revolution in the availability of new devices that go beyond the traditional keyboard and mouse paradigm. Tablets and smart phones use capacitive touch as their only mode of interaction. In 2014, the sales of tablets alone were comparable to the sales of conventional PCs and laptops, and trends predict computing devices with touch and gesture-driven interaction will soon outnumber conventional devices [2]. Further, there has been a proliferation of motion capture-based devices in the recent past, including Google Glass, Kinect, HoloLens, and Leap Motion. Finger or skeletal motion capture-based gesture control is being incorporated into mainstream tablets and laptops [19, 20]. Motion capture-based interaction is also gaining traction in a wide variety of applications such as medical imaging exploration during surgery [30] and infotainment control in cars [1]. In the near future, intuitive and natural user interfaces may become the dominant setting for data exploration.

Given these contexts, several key questions arise. First, can we use the interactivity in modern query interfaces to guide the user to their intended queries during data exploration tasks? Second, are the new generation of computing devices ready for ad-hoc, exploratory data interaction? How can we improve them and effectively guide users to the intended query under different computing devices? In the following paragraphs, we summarize the challenges involved when answering these key questions.

Proceedings of the VLDB Endowment, Vol. 8, No. 11

1.1 Challenges in Data Exploration

Data Complexity: With the increasing size and complexity of the data, users are easily overwhelmed by the data. It is hard for users to understand and explore the dataset and then issue the intended query. Effective feedback must be provided during an exploratory querying step to guide users to the intended query.

Ambiguous Query Intent: Users often do not have a clear query intent when exploring datasets [27]. This could occur because the user is unfamiliar with the data, schema, or query language, or because the query specification requires information that can only be derived by prior data exploration. For example, if a user wants to find a cheap restaurant nearby, "cheap" here can be regarded as an ambiguous query predicate. Under this case, the system should be able to provide effective feedback on the recommended condition depending on the other predicates (e.g., *city*), and perhaps suggest that \$8 – \$12 be regarded as "cheap".

Continuous Actions: In the case of new devices that employ touch and motion capture, users articulate queries using gestures, treating query specification as continuous process [28]. Due to the fluid nature of this interaction, it becomes easier for users to miss the intended queries. This may be caused by jitter and sensitivity of the device, or by users performing an unintended gesture. Since such interfaces typically employ a direct manipulation paradigm, users rely entirely on the query result to orient themselves. In these cases, a "bad" query could dramatically change the result, confusing the user. For example, a bad query may return empty answers. In this case, it is the system's responsibility to provide effective feedback to guide and orient users towards their intended query.

Consideration of Feedback: In addition to the fact that feedback is necessary to help users, there are several considerations on how the feedback is rendered. First, the feedback should be obvious and noticeable enough to attract the user's attention. Second, if an action triggers feedback, the feedback should not interrupt the **fluid** data exploration. Feedback for unintended queries will interrupt the query interaction, resulting in an unpleasant user experience. This presents a challenging trade-off problem.

1.2 Challenges with Next-Generation Devices

Mapping from 3D to 2D Space: Unlike the 2D coordinate system in the traditional interaction modes, next-generation motion capture-based devices allow users to move their hands, fingers, and body around in 3D space. Although one simple solution is to map any two of the axes into the 2D space, this is not enough: should we map the observable 3D space onto the whole screen or the partial space of the screen? Should this mapping be linear? Considering that some parts on the user interface on the screen should not trigger effective actions, and that there is a varying amount of sensitivity in different parts of the 3D space (e.g., regions far away from the sensor), this mapping is nontrivial.

Gestures, Sensitivity, and Jitter: In the mouse-based or touch-based interaction, when users interact with the interface, since users are manipulating physical objects, the presence of friction and force makes the interaction process more accurate. Due to the absence of friction, although motion capture-based devices provide users a more free and natural way to perform gestures in 3D space, it also makes the interaction highly variable and sensitive. This is because users find it hard to hold the cursor at a specific point (jitter) and because sensors detect even the most minor hand movements (Fig. 10 provides a visualization of a Leap Motion trace). These effects can easily trigger unintended and noisy gestures. When it comes to data interaction, this makes it prone to users missing their intended query.

1.3 Motivating Example

To illustrate the concept of *SnapToQuery*, we go over an example. Let us assume a person wants to travel to a city for sightseeing in the next few weeks, and does not have a clear goal of what to do there. The only requirements are that he hopes to go a place with as many places of interest as possible and the date is also flexible within the next few weeks. So, he opens a trip-planning website on his iPad and finds out that he can choose different venue ticket prices, dates, ratings, and neighborhoods using a combination of range sliders and a zoom/pan-driven map. However, he finds it is very hard for him to find a satisfactory answer since there are so many combinations of different neighborhoods, prices, dates, and ratings. In some cases, the system even gives him an empty answer. After spending a lot of time exploring the query space, he gives up out of frustration. Such situations are common in a wide variety of exploratory query settings (as we will see in the two real-world spatiotemporal datasets in our experimental evaluation, Section 5). Beyond just touch, this challenge would exist even with a mouse-based interface. While 3D gestural interfaces such as Leap Motion would allow exploration of additional dimensions simultaneously (each degree of freedom can be used to explore one attribute in the dataset), our challenge becomes even worse when manipulating sliders and maps with such a device, as it is very easy for the user to miss intended UI positions because of sensitivity and jitter.

We look towards ameliorating this problem using *SnapToQuery*. First, the data is represented by a set of linked frequency histograms for date, rating, and area. It is easy for the user to see the number of places of interest under neighborhoods, date, and rating. Second, suppose our user has already chosen a certain area and star rating and when he moves the slider of the date, if there is a large difference on the number of places of interest in the adjacent dates, the user interface will provide some "snapping" feedback, i.e. UI-level resistance to changing the date, drawing attention to this query combination. Similar feedback can be applied to not just range sliders, but to zooming and panning actions on the map as well. With *SnapToQuery*, he finds it is quick and easy for him to find a satisfactory answer.

Contributions: We make the following key contributions:

- We introduce the concept of *SnapToQuery*, a feedback mechanism for guiding users to their intended queries during exploratory query specification.
- We introduce data reduction techniques that allow users to specify queries over large datasets while maintaining an accurate and interactive performance.
- We implement our system for three classes of devices (mouse, touch, and motion capture) and compare the performance of query specification with these different devices. Our

feedback techniques can also be used to power other user interfaces and devices.

- Through performance experiments and user studies over real-world data, we show that *SnapToQuery* helps users to specify queries faster and more accurately, significantly facilitating the exploratory query specification process.

2. RELATED WORK

We now introduce some terms in the context of prior work.

Snapping: Snapping [5, 6, 17] is an important user interface feedback technique to help align digital objects, and is widely used in computer-aided design and drawing programs. The idea behind this is that the shape will automatically jump to the aligned position when it is close enough to the "ideal" position, and users will feel a resistance when they try to move beyond the aligned position. In many cases, additional feedback is also provided to help the user, such as visual feedback. Snapping is also used in other applications, such as multiple monitor display systems [24], text selection [31], and the acquisition of small user interface targets [12]. Recently, a World Wide Web Consortium (W3C) draft discusses controlled panning and scrolling behavior with "snap points" [32]. However, these feedback techniques have rarely been applied to exploratory querying and have seldom been considered from a data standpoint. Further, while snap points restrict querying to one of n values, our method provides feedback that aids exploration while still allowing the user to express any value.

Next-Generation Devices: There are several user interaction devices beyond traditional keyboards and mice that are used for exploring data. There has already been preliminary work in performing traditional data interaction using next-generation devices, such as gestureDB [28], dbTouch [23], TouchViz [14] and Kinetica [34]. However, to our knowledge, *SnapToQuery* is the first to study the querying of data with all major classes of input devices: mouse, touch, and motion capture. Additionally, since the focus of *SnapToQuery* is to provide interactive feedback to aid users in the data exploration process, the contributions here are orthogonal to query language mappings and interface innovations, and can be used to enhance the usability of the aforementioned systems. For this paper, we focus on *Leap Motion*[1] as our motion capture device. Leap Motion is a consumer-grade stereoscopic infrared camera that connects to computers over USB and can track hand and finger skeleton-level gestures, providing programmatic access to recognized gesture, position, direction, and velocity information.

Guiding Exploratory Querying: One challenging task in data exploration and query steering is how to guide users through the process. One approach is to consider it from an interface standpoint, such as introducing visual clues [13, 37, 38] and adopting multi-resolution strategies in proportion to the degree of interest of data subintervals [18]. Another way is to organize the data into some kind of structure to make it easier for the user to explore, such as facets [4, 21] or clusters [11]. An alternative strategy is to incorporate the user's feedback into the iterative process [3, 9]. Finally, steering [8, 26] and recommendation of most likely queries [10] have also been considered. However, as discussed in Section 3 and Section 3.1, the current approaches differ from *SnapToQuery* in that we propose a general framework to provide interactive feedback during the data exploration process, which allows users define their own feedback and "snapping" conditions. Thus, all of these guidance mechanisms can leverage *SnapToQuery*.

Scaling Factor: When users move the cursor around with the mouse, fingers or Leap Motion, there is a mapping from the physical movement to the virtual movement of the cursor. For the mouse, the mapped virtual movement depends on two parameters [33]: one is the scaling factor, the other is the resolution in dots per inch (DPI). The moved dots will be translated to pixels on the screen according to the scaling factor. For example, if the mouse is moved 500 dots and the scaling factor is 1.0, then the cursor will move 500 pixels. For touch-based tablets, the scaling factor is always 1.0 since we need to maintain the relative position between the finger and manipulated object. For Leap Motion, we define our own scaling factor strategy.

Jitter: From a user interface perspective, jitter can be defined as *the vibratory motion of the hand or finger* [25, 39]. In this paper, we term the vibratory motion of the hand or finger around the user's targeted position as jitter. In the motion capture-based device, jitter is a challenging problem. Consider the manipulation of a range slider with Leap Motion, where the position of the user's index finger indicates the position of the handle of the slider. When users are asked to move the handle to a certain position, this task is very hard, since it is impossible for users to keep the index finger absolutely static in mid-air, especially when the scaling factor is large.

3. PROBLEM STATEMENT: SNAPPING TO INTENDED QUERIES

In light of the challenges and prior work, we formulate the *SnapToQuery* concept in a SQL-like syntax:

```
SELECT A from T_1, T_2, ..., T_i
WHERE C_1, C_2, ..., C_j
SNAPTO S_1, S_2, ..., S_k
FEEDBACK F_1(p_11, ...), ..., F_x(p_x1, ..., p_xy)
```

in which $T_{1...i}$ are the relations, $C_{1...j}$ are predicates, $F_{1...x}$ are the feedback functions with parameters $p_{1...x,1...y}$, and $S_{1...k}$ are **SNAPTO** conditions. Table 1 introduces additional terms and expressions used in the paper.

3.1 Implementation Considerations

Due to the tightly integrated nature of the system, there are several implementation considerations that determine how *SnapToQuery* can be adapted to the exact user interface and device combination:

Feedback: Feedback is used to guide users to the intended query. There are different kinds of feedback possible, such as visual feedback, haptic feedback, etc. Users are allowed to define their own feedback that best suits their use case as a UDF (user-defined function).

SNAPTO Condition: Users are allowed to define their own rule-based **SNAPTO** conditions to guide the user to the intended query. Alternatively, the system can automatically recommend a good snapping condition, which is a hard problem. As discussed in the challenges, we need to provide obvious and noticeable feedback, but at the same time we

[1] https://www.leapmotion.com/

need to ensure that the feedback will not interrupt the fluidity of the user interaction. A good snapping condition can be considered from different aspects. First, it can be derived from prior usage: snapping can be based on the most frequent queries from the query log. Second, we can consider snapping from a data-only perspective: we can define a metric to measure the "impact" of changing the query on the current result. The recommended condition can be the one with the maximum effect or the one that causes the maximum difference of effect in adjacent queries. The measures can be information entropy, `COUNT`, `AVG`, etc.

Visualization & User Interface: Another interesting consideration is that the feedback presented has a close relationship with the visualization and the user interface. For example, if the snapping feedback is provided on the boundary of a dataset, for two different visualizations (bar chart and pie chart), the UDF implementations of `FEEDBACK` functions can be quite different, as shown in Figure 1.

Devices: The implementation of *SnapToQuery* is also related to the specific device it is used with, based on the fact that different devices have different nuances in their user interaction. For example, mouse and touch-based interfaces are much more accurate than the motion capture-based device, but the motion capture-based is more expressive (it has more degrees of freedom and can be more sensitive to movement), making it possible to explore more `WHERE` conditions simultaneously. In order to cater to multiple device contexts, we discuss variations of the *SnapToQuery* implementation for each device in Section 4.2.

Symbol	Meaning
l_s, l_h	length of the slider, handle in pixels
r	ratio of length of bar over length of the bin
n	number of bins for each dimension
d	number of dimensions for the dataset
d_s	snapping distance
w/, w/o	with, without
R	ratio of d_s over l_h
R_l, R_{lr}	scaling factor of Leap Motion w/ snapping, w/o snapping
R_{ll}, R_{lh}	minimum, maximum value of R_l
Δ	difference of cardinalities in adjacent queries
Δ_l, Δ_h	minimum, maximum value of Δ when snapping occurs (we assume $\Delta \leq \Delta_h$)

Table 1: Expressions

3.2 Query Exploration

While we focus on aggregation queries on a single relation with the exploration of range predicates in this paper for simplicity, the concept can be generalized to a much wider range of queries, and is ideal future work.

For simplicity, we characterize our exploration model as follows. We assume here that only one relation is involved, that the dimensions involved in $C_{1...j}$ are numerical dimensions, and that each of $C_{1...j}$ is a range predicate (non-numerical values can be hashed to numerical values with some notion of locality). Thus, our query can be considered an aggregation over an n-dimensional subsetting of a composition of relations; e.g., if a relation had 3 attributes, each of which were numerical (from our motivating example: dates, rating, and price), then each tuple (i.e., venue) could be represented as a point in a 3-dimensional vector space, and the task of our exploratory query session is to come up with a cuboid that contains the intended points, by moving the handle on a slider for each dimension, as shown in Figure 1. An exploration is performed by issuing a succession of queries using a user interface, which are thus "adjacent" to each other (e.g., a slider being moved will cause the successive queries to be adjacent on a dimension). For successive queries, we assume that the change of predicate for a dimension is continuous and only one value is changed. For example, assume there are three ranges for a dimension, which are $[v_1, v_4]$, $[v_2, v_4]$ and $[v_3, v_4]$, and $v_1<v_2<v_3<v_4$. If the current range is $[v, v_4]$, $v = v_3$ and the user is changing v (called the **manipulated value**), there is no way to skip $[v_2, v_4]$ and jump to $[v_1, v_4]$ directly. Based on this model, we consider an example implementation of our SQL-like formulation:

```
SELECT COUNT(*) from T
WHERE v11 < d1 < v12, v21 < d2 < v22, ..., vj1 < dj < vj2
SNAPTO Δ ≥ Δl
FEEDBACK snapping(Δ, Δl, Δh)
```

where Δ represents the relative change in the `COUNT(*)`. This implementation uses the feedback of *snapping* using differences in query cardinality – users will feel a "stickiness" or "resistance" in the UI controls when they try to change one predicate of the current query and the change of a predicate results in a large change of result cardinality. This intuitively draws the user's attention to the fact that the current query is a representative of the current parameter space, and hence is a likely candidate to consider as an **intended query**. We call the position where the snapping happens as the **snapping position**.

Thus, our problem becomes: *given a current query, determine whether it is an intended query, and provide feedback if it is.* Note that it is hard to consider every query: first, under the direct manipulation, the number of possible follow-up queries issued can be superlinear [28], based on the number of columns (i.e., attributes / dimensions) that can be manipulated. If every query is considered, the computation cost can become very high, leading to a lack of responsiveness at the UI-level. Second, typically most adjacent queries have similar cardinalities, and hence the re-computation does not add any value. Thus, it is wise to group the adjacent queries together and use one query to represent the whole group, which is called the **representative query**: only representative queries are considered as candidate intended queries.

Given this scenario, a question arises: how do we group the queries and select which query to represent the query group? In the following section, we propose two methods: a baseline *naive* method, and a *data contour* method.

4. THE SnapToQuery SYSTEM

The overall architecture of our system is shown in Figure 2. The dataset is reduced in the backend and visualized as brushing and linking-based coordinated histograms [16] in the frontend, exposed as an application, such as our trip planning website from Section 1.3. Each dimension corresponds to one histogram and one range slider with two

handles. The dataset is divided into a set of bins on each dimension. The height of each bar in each bin represents the frequency of that bin. Users can manipulate the sliders on three devices (mouse, tablet, and Leap Motion). By moving handles on the sliders, users can issue range queries. We note that the filtering effect is cumulative, meaning, if users filter one dimension, the histograms in other dimensions should be changed dynamically. During the query specification process, the system will be "snap" to the intended query.

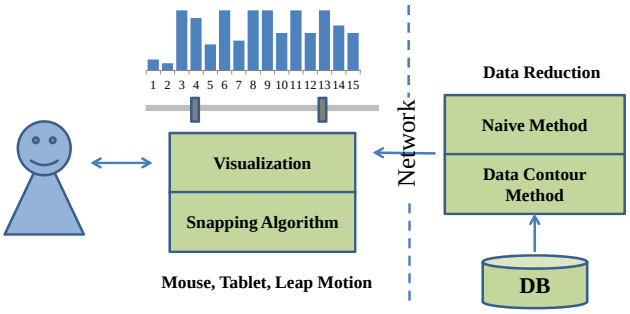


Figure 2: Overall Architecture of *SnapToQuery*. The dataset is preprocessed using a data reduction method at the backend, and then interacted with using a frontend that provides the interactive visualization and snapping feedback.

4.1 Data Reduction

Since we need to maintain interactive performance times for a fluid user experience, performing queries directly on the raw data is untenable if the dataset is large. Our goal is to find a way to reduce the dataset while still maintaining the characteristics of the dataset. At the same time, we group the queries together and select the representative query. Thus, two data reduction strategies are proposed: a *naive* method and a *data contour* method.

4.1.1 Naive Method

Backend: The main idea behind the naive method is aggregation. Specifically, instead of splitting the data when the dataset is large, we can split the space, map each data point into a cell, and then represent each cell by only one data point with an additional dimension **freq** representing how many data points reside in this cell and whose values are aligned with the left boundary of the cell over all of the dimensions. The algorithm for the naive method is shown in Algorithm 1. First, we divide each dimension into a certain number of bins. Currently, each dimension is evenly divided into the same number of bins. Second, we map the data point into the corresponding cell in the grid. Third, the empty cells will be removed. Finally, one cell is represented by one tuple, and, as we can see, the number of data points left will be at most $O(n^d)$. Thus, the remaining data points are independent of the data size and are impacted by the number of dimensions and bins.

Frontend: In the frontend, for each filter action, we aggregate over the **freq** dimension and update the histograms. We assume that the data is uniformly distributed in each cell, and if the predicates are not aligned with the bins, at the last step of the aggregation, we need to adjust the cardinality according to the overlapping percentage between the current filter range and selected cells.

Algorithm 1 Backend: Naive

1: Define the grid
2: Map the data points into the appropriate cell in the grid
3: Remove the empty cells
4: Each cell is represented a data point with an additional **freq** dimension

Algorithm 2 Determination of Snapping Positions: Naive

```
UpdateSnappingPoses (int curIndex)
1: for dim in dimensions do
2:     if dim.index != curIndex then
3:         snapPoses[dim.index] = [ ]
4:         for i = 1; i < dim.bins.length; i = i + 1 do
5:             curBin = dim.bins[i]
6:             preBin = dim.bins[i - 1]
7:             if curBin.val - preBin.val > Δ then
8:                 snapPoses[dim.index].push(curBin.key)
```

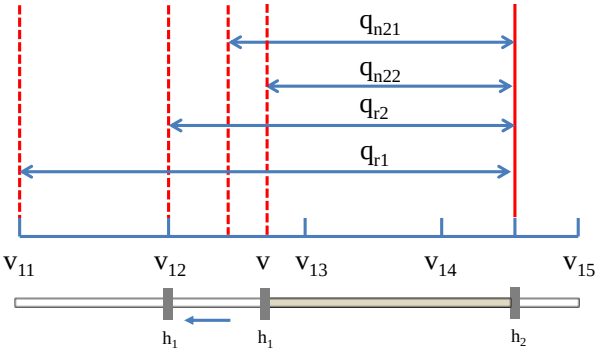


Figure 3: Determination of Intended Queries and Snapping Positions for Naive Method. v_{1i} is the boundary of the bin. The user drags the handle h_1 from v_{13} to v_{12}. q_{n21} and q_{n22} are the neighborhood queries of q_{r2} and represented by the query q_{r2}. If there exists a big cardinality difference between q_{r2} and q_{r1}, snapping will occur on v_{12}.

Determination of Intended Queries and Snapping Positions *(Naive)*: The representative queries are those whose manipulated value is aligned with the bin. The query between two adjacent representative queries is represented by its left representative query. Figure 3 gives an example to illustrate how the naive method groups the queries, selects the representative query, and determines the intended query. Assume $v_{11}, v_{12}, ..., v_{1i}, ...$ are the boundaries of bins in one dimension, $v_{12} \leq v < v_{13}$, and the user is dragging the handle h_1 from v_{13} to v_{12}. All the queries issued in the process are represented by the query q_{r2}. For example, q_{n21} and q_{n22} are represented by the query q_{r2}. Once the handle h_1 is moved onto v_{12}, it means q_{r2} is specified. Since q_{r2} is a representative query, the system will check whether there exist a big cardinality difference between q_{r2} and q_{r1}. If it is, the snapping will occur on the value v_{12}.

For each filter, we need to update the snapping positions in other dimensions, as shown in Algorithm 2. We iterate through each representative query and compare the cardinality difference between adjacent representative queries and save the snapping positions.

4.1.2 *Data Contour Method*

Although the naive method is very intuitive and works very well in most cases, there are two major problems with it: first, it cannot fit high-dimensional datasets because as the number of dimensions increases, the number of tuples left will increase exponentially. Second, it does not take advantage of the characteristics of the dataset. Actually, the queries aligned with the contour of the cluster is where the cardinality change is large, which is shown in Figure 4. Since the number of clusters is much less than the number of bins in each dimension, it will also vastly improve the interactive performance. There are several other options for detecting the contour of the clusters: one is the edge detection method in computer vision and the other is the grid-based clustering method, which suits this problem very well. However, both methods can only work on low-dimensional datasets. In order to handle high-dimensional datasets, we adopt a KMeans + histogram method.

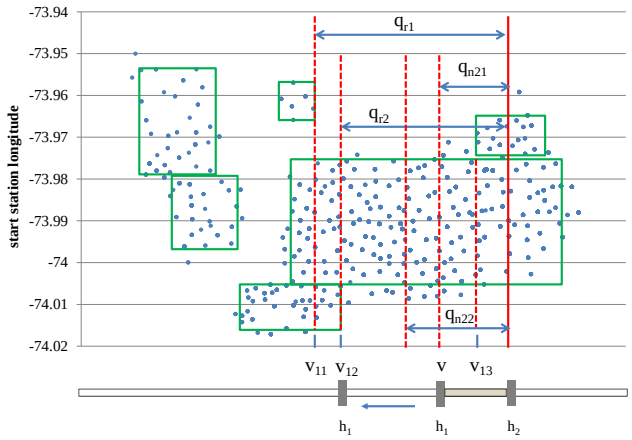


Figure 4: Determination of Intended Queries for Data Contour Method. There are six clusters in the dataset and each cluster is bounded by a box. The boundary of the box is the data contour of the cluster. v_{1i} is the contour of the cluster in one dimension. q_{n21} and q_{n22} are the neighborhood queries of q_{r2} and represented by the query q_{r2}. Snapping will occur on the contour of the cluster, such as v_{11}, v_{12} and v_{13}.

Backend: In the backend, the dataset is clustered first. In order to give a more accurate cardinality approximation, we divide each dimension into a set of bins for each cluster and assume in each bin the data points are uniformly distributed. The backend algorithm is shown in Algorithm 3. The number of clusters is determined by the gap statistic [15,36]. We use a cluster-based method [35] to remove outliers: whether a point is an outlier or not is based on the ratio of the distance between the point and the centroid to the distance between the median point and the centroid. After outliers are removed, we re-cluster the dataset.

Frontend: For a given filter range, Algorithm 4 shows how to update the frequency of the bin in the frontend. Specifically, for the filtered dimension, we attain the overlapping ratio between each cluster and the current filter for the current dimension, and then update the histograms in other dimensions. The cardinality is achieved by aggregating the frequencies of the selected bins.

Algorithm 3 Backend: Data Contour

```
1: Normalization
2: Select K points as initial centroids
3: Form K clusters by assigning each point to its closest centroid
4: Remove outliers
5: Form K clusters by assigning each point to its closest centroid
6: Divide each dimension into bins for each cluster
7: Remove empty bins of each dimension for each cluster
```

Algorithm 4 Update of Bins: Data Contour

```
UpdateBins (int curIndex)
 1: for cluster in clusters do
 2:     cluster.preRatios[curIndex] = cluster.ratios[curIndex]
 3:     cluster.ratios[curIndex] = getRatio(cluster, curIndex)
 4: for dim in dimensions do
 5:     if dim.index != curIndex then
 6:         for cluster in clusters do
 7:             for bin in cluster.dims[dim.index].bins do
 8:                 preRatio = cluster.preRatios[bin.index]
 9:                 curRatio = cluster.ratios[bin.index]
10:                 val = bin.val
11:                 if preRatio == 0 then
12:                     val *= curRatio
13:                 else
14:                     val *= curRatio / preRatio
15:                 dim.bins[bin.index].val += val
```

Algorithm 5 Determination of Snapping Positions: Data Contour

```
setSnappingPoses ()
 1: for cluster in clusters do
 2:     for dim in cluster.dims do
 3:         for bin in dim.bins do
 4:             if bin.val > thresh then
 5:                 snapPoses[dim.index].push(bin.key)
 6:                 break
 7:         for bin in reverse(dim.bins) do
 8:             if bin.val > thresh then
 9:                 snapPoses[dim.index].push(bin.key)
10:                 break
```

Determination of Intended Queries and Snapping Positions *(Data Contour)*: In contrast to the naive method, the representative queries are those whose manipulated value is aligned with the contour of the cluster. The query between two adjacent representative queries is represented by its left representative query. Snapping will occur if the manipulated value is aligned with the contour of the cluster. Figure 4 gives an example to illustrate how the data contour method groups the queries, selects the representative query, and where the snapping occurs. Assume $v_{11}, v_{12}, ..., v_{1i}, ...$ are the contour of clusters in one dimension, $v_{12} \leq v < v_{13}$, and the user is dragging the handle h_1 from v_{13} to v_{12}, all the queries issued in the process are represented by the query q_{r2}. For example, q_{n21} and q_{n22} are represented by the query q_{r2}. Once the handle h_1 is moved onto v_{12}, it means q_{r2} is specified and the manipulated value is aligned with the contour. Figure 4 shows there exists a large cardinality difference between q_{r1} and q_{r2} since another cluster will be involved if we move the handle h_1 slightly towards the left, the snapping will occur on the value v_{12} and always occur on the contour of the cluster.

The contour of the cluster is determined by histograms, detailed in Algorithm 5. Specifically, we iterate the bins of the dimension for each cluster from both ends, and stop when the frequency of the bin is greater than the threshold.

We use the value of the boundary bins as the contour of the cluster for this dimension. When users filter the dataset in the frontend, the representative queries are those that are aligned with the contours. Since we assume the data is uniformly distributed in each cluster, for each filter we do not need to re-cluster and re-compute the cluster contours.

4.1.3 Comparing Naive and Data Contour Methods

For the naive method, since we divide the data points over all dimensions, the remaining data points will be $O(n^d)$. For the data contour method, since we divide the data points on each dimension, the remaining data points will be $O(knd)$, where k is the number of clusters. As we can see, with the dimension increasing, the performance of the naive method will be unsatisfactory and for the data contour method it will maintain a linear scalability.

4.2 SnapToQuery Algorithm

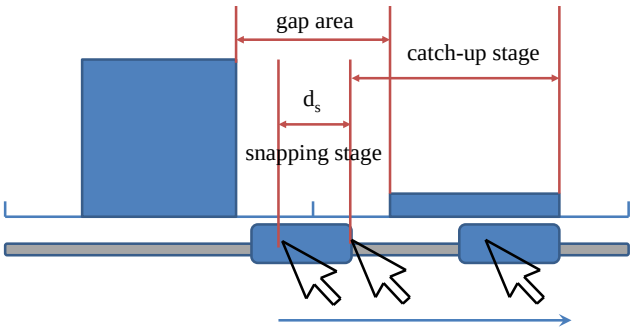


Figure 5: Mouse & tablet *SnapToQuery* Implementation. In the snapping stage, the handle will keep still while the cursor or finger is moving until the cursor or finger moves beyond the handle. In the catch-up stage, the handle will catch up with the cursor or finger before the next snapping position.

Snapping Function: We propose a linear snapping function depicted below. The larger the Δ, the larger the R, the larger the d_s, and then the more snapping users will feel.

$$R = \frac{\Delta - \Delta_l}{\Delta_h - \Delta_l} + \Delta_l \quad (1)$$

$$d_s = R \times l_h \quad (2)$$

where for the naive method, Δ refers to the cardinality difference, and for the data contour method, it refers to the data density over this dimension for the cluster.

There exist distinct differences between the mouse, tablet, and Leap Motion. For the mouse and tablet, we cannot control the position of the cursor, and it is necessary to maintain the relative position between the handle and the mouse cursor or finger. For Leap Motion, since we determine the scaling strategy from the finger position to the slider, we do not need to consider the relative position problem. We propose two different *SnapToQuery* implementations developed for different devices.

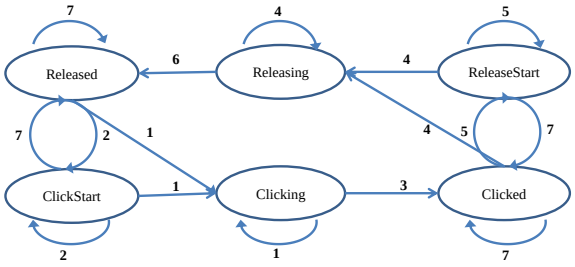
4.2.1 Mouse & Tablet SnapToQuery Implementation

The *SnapToQuery* algorithm for the mouse and touch-based tablets is shown in Figure 5. Once the handle is moved into the gap area, the system will determine whether this is an **intended query**. If it is, the handle will be static throughout the entire snapping stage while the cursor is moving. Until the cursor is moved beyond the handle, the handle will start to catch up with the cursor. And, d_s depends on the moving direction, if users move from left to right, it is offset between the left boundary of the cursor and the right boundary of the handle; otherwise it is the offset between the left boundary of the cursor and the left boundary of the handle. If users accidentally move across the snapping boundary, the handle will also be static and the value of the slider will be the value represented by the closest boundary of the gap to the left of the handle. It is obvious that the snapping greatly depends on d_s and the initial offset of the cursor and handle, in order to generate an obvious snapping feedback between the adjacent intended queries, we will adjust the relative position between the cursor and handle to make sure when the handle is close to the next intended query, d_s is in position.

4.2.2 Leap Motion SnapToQuery Implementation

There are several design considerations for Leap Motion

First, we allow the thumb and index finger of one hand to manipulate the sliders and perform queries. There are, in total, six gesture states. We collect the most recent 10 points and develop a rule-based classifier to recognize the gesture state according to the angle between the thumb and index finger. The idea behind this is that when the thumb and index finger are close enough, users are able to slide the handle. This state is called **clicked** state. When the thumb and index finger are separated enough, users can switch handles or sliders. We call this state **released** state. All of the others are intermediate states and will not trigger the effective action. This way, we can make sure that the ending gesture will not change the final state of the slider. The transition between gesture states and transition conditions are shown in Figure 6. *signLen* indicates the number of most recent points that are greater (Label 4, 5, 6) or less (Label 1, 2, 3) than the degree threshold in the sequence. *seqLen* indicates the number of the most recent points that are increasing (Label 5) or decreasing (Label 2) in the sequence. We allocate the vertical space evenly to different sliders and the horizontal space evenly to the handles of the same slider.



Label	Condition
1, 4	$signLen>0$
2, 5	$signLen = 0$ and $seqLen>thresh$
3, 6	$signLen = 10$
7	else

Figure 6: Gesture State Transition

Second, we use visual signals to indicate the state of change. We constrain the mapping to the handle only. The handle with a red border indicates the currently selected handle. When users drag the handle, a circle indicates the position of the simulated cursor.

Finally, we implement snapping effects from two perspectives, one from the scaling function, and the other from the snapping function. Additionally, the user position is smoothed over the 10 recent positions to reduce jitter.

Scaling Function: Between adjacent intended queries, a sigmoid function is proposed to adjust the scaling factor dynamically, which is shown in Figure 7. Specifically, the scaling factor is very small when the handle is close to the snapping position, once the users move out of the snapping position, the scaling factor will noticeably increase.

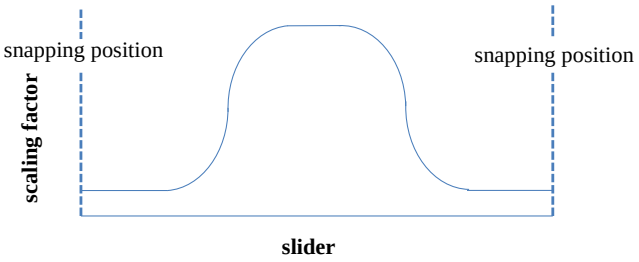


Figure 7: Dynamic Scaling Factor in Leap Motion. The scaling factor changes according to the handle position on the slider. When the handle is close to snapping positions, the scaling factor becomes small.

5. EXPERIMENTS AND EVALUATION

We evaluate *SnapToQuery* system from three aspects: performance, quality, and usability. Section 5.2.2 evaluates whether the data reduction strategies can maintain an interactive performance. Section 5.2.3 evaluates whether the data reduction strategies can give an accurate approximation of the cardinality of the query. Section 5.3 evaluates the performance of the query specification with three devices (mouse, iPad, and Leap Motion) with snapping and without snapping, and tries to show whether the snapping aids the query specification. Section 5.4 attempts to show whether users feel the snapping when they explore the dataset and thus pay attention to the intended query.

5.1 Experiment Setup

Configuration: Experiments are performed on a Windows PC, which has 4GB memory and one Intel i5-3320M quad-core processor. The computer has a 14-inch display with 1366×768 resolution. The iPad is the 3rd generation iPad with 16GB. We use Tornado[2] as a web server, D3 [7] to visualize, and Crossfilter[3] to aggregate the dataset.

Datasets & Workloads: We use two real-world datasets for the experiment, which are shown in Table 2. DATA$_{ROAD}$ is a 3D road network dataset from the UCI machine learning repository [22, 29]. DATA$_{BIKE}$ is a city bike trip dataset of NYC in January 2014[5]. For DATA$_{BIKE}$, we select only 5 dimensions: trip duration, start station latitude, start station longitude, end station latitude, and end station longitude. Further, we only keep the tuples whose trip duration is less than 2 hours. A random sample having the same number of tuples as the naive method is also generated from each dataset for scale experiments. We use the **clusterGap** function in **cluster** package of **R** to estimate the number of clusters[6] by measuring the goodness of clustering with a gap statistic [15] and selecting the best K from 1 to 50. To avoid the memory restrictions in the gap statistic, the number of clusters is estimated on the 1% of the dataset. For the performance experiment, we generate a random workload for each dataset. For each workload, we ensure that each dimension has been filtered multiple times. The session size of the workload for each dataset is presented in Table 2.

Name	DATA$_{ROAD}$	DATA$_{BIKE}$
number of tuples	434874	299648
number of dimensions	3	5
number of clusters	47[4]	44
session size	148	228

Table 2: Datasets and Workloads

5.2 Performance Experiments

In this section, we measure the performance of data reduction strategies and how accurate they approximate the cardinality of queries.

5.2.1 Data Reduction

In the backend, the dataset is preprocessed. Each dimension is divided into 20 and 50 bins separately. The size of the data file after the data reduction is shown in Table 3. Both the naive and the data contour methods reduce the data size immensely, and the data contour method occupies the least space.

Name	DATA$_{ROAD}$	DATA$_{BIKE}$
original size	15, 294 KB	14, 444 KB
naive w/ 20 bins	77 KB	3874 KB
data contour w/ 20 bins	32 KB	72 KB
naive w/ 50 bins	640 KB	10, 035 KB
data contour w/ 50 bins	52 KB	137 KB

Table 3: Data Reduction

5.2.2 Query Execution Time

The performance is determined by the average time that the system takes to run one query in one query session. We replay each query session for each dataset five times to get the average time for each query. Figure 8 shows the average **log time(μs)** for each query in each dataset. The data contour method has the fastest performance for all datasets. For naive and sampling methods, since the performance is closely related to the number of bins and dimensions, the performance will be pretty bad when the number of bins and dimensions increase. Usually, the sampling method takes less time than the naive method, since the naive method sums up the frequency instead of counting.

[2] http://www.tornadoweb.org

[3] http://square.github.io/crossfilter/

[4] Only 44 clusters are left when we removed empty clusters.

[5] http://www.citibikenyc.com/system-data

[6] http://cran.r-project.org/web/packages/cluster/index.html

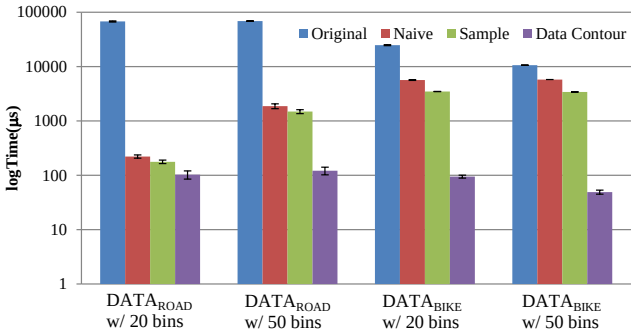


Figure 8: Average Query Execution Time for Each Dataset with Different Strategies. All of data reduction strategies can maintain an interactive performance and the data contour method takes the least amount of time.

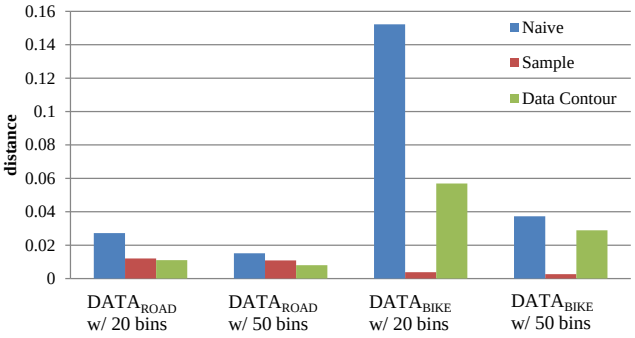


Figure 9: Distance for Data Reduction Strategies over Different Datasets. All data reduction strategies can maintain a low distance. The data contour method is better than the naive method in all cases.

5.2.3 *Distance*

We use a distance metric to measure the accuracy of the approximation of the cardinality for different data reduction strategies: $d(H,K) = \sum_i |h_i - k_i|$, where $H = \{h_i\}$ and $K = \{k_i\}$, h_i represents the accurate cardinality and k_i represents the approximate cardinality returned by one data reduction strategy. H and K are normalized so that $\sum h_i = 1$ and $\sum k_i = 1$.

Figure 9 shows the result for different data reduction strategies over different datasets. As we can see, all strategies are able to maintain a low distance, and the data contour method is always lower than naive method. One possible reason behind this is that for each filter, if the predicates are not aligned with the bins and the filter range overlaps with the cell, it means that in all the following filters of the other dimensions, the data points in the cell are not involved. It is even worse when the distribution of the data points in the cell is skewed, since it makes the cardinality adjustment less accurate. As more and more filters are performed over the dataset, the approximation becomes less accurate. However, for the data contour method, the dataset is organized in clusters first and then histograms. The clustering captures the characteristics of the dataset. On the other hand, each dimension is divided into bins separately, which means the filter over one dimension will not have a great impact on other dimensions. Thus, the data contour is more robust towards skewed datasets.

5.3 Query Specification Experiments

In this experiment, we ask users to specify the query task with three devices (Mouse, iPad, and Leap Motion) and compare the performance. 30 users are included in the experiment. Each user performs the task with one device with or without snapping. It means for each device under one condition, it has five users. For example, five users will perform the task with the mouse with snapping.

5.3.1 *User Study Setup*

Dataset: We perform the user study on DATA$_{ROAD}$ with 50 bins. The ranges for each of the three dimensions in the dataset are $[8.146, 11.262]$, $[56.582, 57.774]$, and $[-8.608, 137.361]$ respectively.

Parameters: The parameters for the experiment are shown in Table 4.

parameter	l_s	l_h	r	R_{lr}	R_{ll}	R_{lh}	R
value	300	60	0.8	0.2	0.1	0.6	0.9

Table 4: Parameters

Since the resolution of the computer is different from that of the iPad, we adjust the length of slider and handle in the iPad to make sure the sliders have the same absolute length. For Leap Motion, the unit of physical movement is in millimeters. Since $R = 0.9$, $d_s = l_h \times R = 54$. We choose $R_{lr} = 0.2$ since it allows users to move the handle from the start to the end without lifting their fingers again.

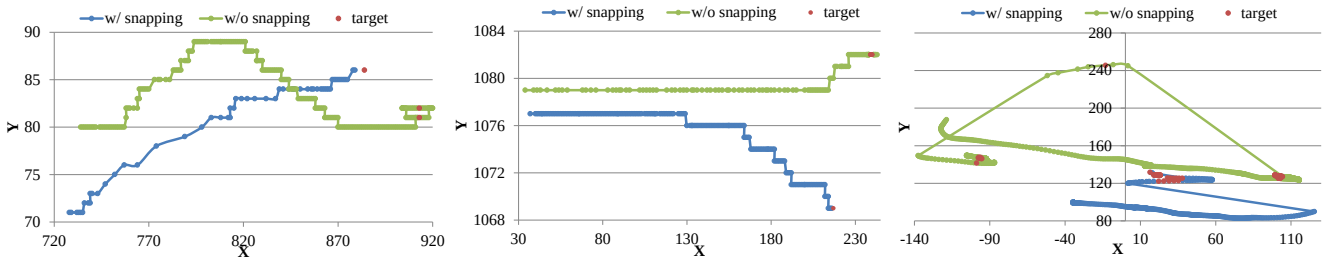
Task: The query task is shown below:

- Move the first handle of the first slider to [10.009,10.022]
- Move the second handle of the first slider to [10.57,10.582]
- Move the first handle of the second slider to [57.498, 57.502]

where [10.009,10.022] and [10.57,10.582] is the first snapping position while [57.498, 57.502] is the second snapping position. All the handles are positioned at the ends of the slider at first. The design of the query task considers the effect of the handle switch, slider switch, and the possible delay of the snapping.

Measures: The experiment is evaluated by two measures. The first one is the average specification time – how much time it took the user to specify the query over all users. The second one is to evaluate how *SnapToQuery* helps prevent users from making errors by measuring how many times users miss the intended query.

Considerations: In order to avoid carryover effects, we perform a between subjects experiment, which means each user only performs with one device under one condition. Another thing to keep in mind is that since users are used to the mouse and tablets, before the experiment, we will ask users to play a demo and manipulate the sliders with the Leap Motion for about 10 minutes. Since the purpose of the experiment is to show whether the snapping helps users to specify the query, we will fix the snapping positions to ensure intended queries will not change when users filter the dataset.



(a) Mouse Trace (b) iPad Trace (c) Leap Motion Trace

Figure 10: Traces of Different Devices with Snapping and without Snapping. Read points represent the targeted positions. Users can move the handle into the targeted positions on the first attempt for the mouse and iPad with snapping. Read points for Leap Motion without snapping are quite scattered while they concentrate on one area for Leap Motion with snapping. In summary, the devices without snapping present more jitter than devices with snapping and Leap Motion presents more jitter than the mouse and iPad.

5.3.2 *User Traces*

In order to get a visual sense of what happens during the query specification, we select one person's trace for the first sub query task. Figure 10 shows the trace for different device with and without snapping. The red points represent the targeted position. There are several observations we can gain from the trace. First, we can see users miss the targeted position several times and spend much time around the targeted position for devices without snapping. Second, Leap Motion presents more jitter than the mouse and the iPad. There are still misses for Leap Motion with snapping and the range of the data points are larger than the mouse and iPad. If we can reduce the jitter and miss times, it will help users to specify the intended query much faster. Third, snapping helps users specify the query. Users can move the handle into the targeted position without misses for mouse and iPad with snapping. For Leap Motion with snapping, the red points concentrate on one position. For Leap Motion without snapping, the read points are quite scattered, which means there are many misses and users start over the task several times.

5.3.3 *Time to Specify Range Queries*

Figure 11 shows the specification time over different devices with and without snapping. The time is divided into four parts: subtime1, subtime2, subtime3, and switchtime. subtime**i** indicates the specification time for i^{th} task, e.g. subtime1 is the time for the first task. Switchtime describes the time it takes to switch between different handles and sliders. For all devices, the *SnapToQuery* helps to reduce the specification time. And for Leap Motion, the gain is the most effective, the speed up is 2.2. For Leap Motion with snapping, it is comparable to the performance of the mouse and the iPad.

5.3.4 *Miss Times*

The miss times represent how many times users miss the intended query in the query specification process. Figure 12 shows the result. For all devices, *SnapToQuery* helps reduce the miss times, and the effects are pretty impressive. And for subtask1 and subtask3, the miss times are almost 0. Since the targeted position of subtask2 is close to the end of the slider, it is easier for users to miss it even with snapping.

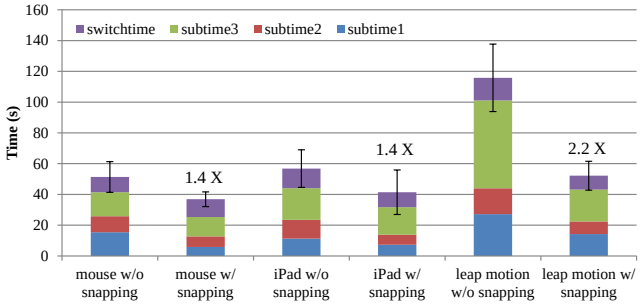


Figure 11: Specification Times of Devices with Snapping and without Snapping. Devices with snapping take less time than devices without snapping and the speedup is at least 1.4×. Leap Motion with snapping is comparable to the mouse and the iPad.

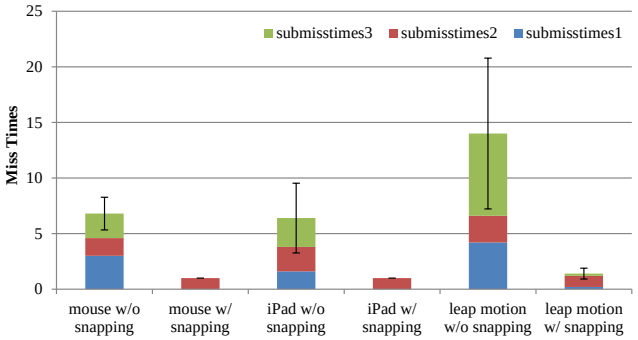


Figure 12: Miss Times of Devices with Snapping and without Snapping. The miss times of devices with snapping is much smaller than those of devices without snapping for all devices.

Device	P-value for Time	P-value for Miss Times
Mouse	0.0208	0.0007
iPad	0.0726	0.0131
Leap Motion	0.0012	0.0102

Table 5: One-sided Unpooled T Test

5.3.5 *Statistical Significance*

In order to determine whether the difference between the case with snapping and that without snapping is statistically

significant. We perform a one-sided unpooled t-test on the specification time and miss times, shown in Table 5.

5.4 Snapping Discoverability Experiments

We further measure the usability of *SnapToQuery* and attempt to find out whether the intended query draws the user's attention. 15 users are included in the experiment. It should be noted that in order to avoid bias and memory effects, these users are recruited *separately* from the previous experiment. DATA_{ROAD} with data contour method works as the testbed.

Parameters: The parameters of the experiment are the same as the query specification experiments except $l_s = 600$. We choose a different slider length since the range of third dimension is much larger than the first two dimensions.

Task: Each user is asked to manipulate the second handle of the third slider to discover the values at which the feedback occurs given five minutes with the mouse. Only one target range is involved in the experiment. In order to allow for discovery, we did not tell users what the feedback looks like and only tell them that some feedback exists when they move the handle around. After the experiment, a survey is performed to check the usability of snapping.

Discovery Time & Miss Times: Figure 13 shows users are able to find the snapping position in a short time and make few misses.

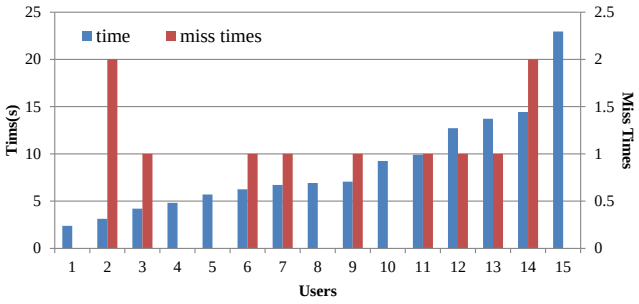


Figure 13: Discoverability. Users are able to find the snapping position in a short time and make few misses.

Survey: Although users feel the snapping and pay attention to the intended query, one thing to keep in mind is that we do not want the snapping to detract from the fluid interaction. It is also necessary to maintain the relative position between the cursor and the handle. After the experiment, we asked users of the discovery experiment to fill out a survey. Table 6 shows that all users think the snapping attracts their attention. Most users do not notice the mismatch between the cursor and think that snapping does not hurt the fluid interaction.

5.5 Insight from the User Studies

There are several observations we attain from the user study. First, since Leap Motion has the detection boundary, when users move the hand/finger out of the boundary, as we can imagine, the slider will not respond to user's action. However, users do not understand what is actually occurring here. They think that the system is stuck because of the dense computation, network problems, etc. We should give a warning when this kind of phenomenon happens. Second, in the survey feedback, several users talk about the visual feedback. The combination of visual and haptic feedback should help the query specification. Third, when users manipulate the slider with Leap Motion for a while, some users complained that their hands were getting fatigued. Designing the gesture with less effort and developing ways to relieve the fatigue are challenging problems for Leap Motion. Finally, when users move the handle quickly, it is pretty hard for them to notice the snapping. One way to solve this is to incorporate sliding speed into the snapping function or dynamic scaling factor: the faster the sliding speed is, the larger the snapping distance is and the lower scaling factor is.

Question	Yes	No
Pay attention to data point with snapping	15	0
Obvious mismatch b/w the cursor & handle	4	11
Snapping hurts the fluid interaction	4	11

Table 6: Discoverability Survey

5.6 Discussion of Parameters

We now discuss several key parameters that impact experimental results. The first one is the upper-bound of scaling factor of Leap Motion with snapping. This represents a trade-off problem – a higher upper-bound scaling factor allows users to specify the intended query much faster, however, it will be harder to specify non-intended queries. The second is the length of the slider. The longer the slider is, the easier it is to move to a specific position: it results in fewer miss times, but dampens the effects of *SnapToQuery*. A longer slider also means that it takes more time for users to move to the targeted position with the same scaling factor. The third is the snapping distance. A larger snapping distance means users will feel more obvious snapping and it is harder for users to miss the intended query. However, it also increases the interruption of the fluid interaction.

6. CONCLUSION AND FUTURE WORK

Guiding users during the data exploration is a challenging task. In this paper, we present the *SnapToQuery* concept, which provides feedback to the user and allows them to explore the query space in an efficient manner. We discuss several considerations ranging from the nature of the feedback at the user interface and device level, to data scale and performance considerations. We propose a data-driven approach to the exploratory query specification: given a neighborhood of queries, snapping feedback is provided on a representative query if the difference between the cardinalities of the adjacent representative queries is large. In order to maintain interactive performance, two data reduction strategies are detailed and evaluated over two real datasets, showing that both data reduction algorithms can provide good quality and interactive performance. We detail implementations of snapping functions on different devices, and evaluate them empirically. Through our user studies, we are able to demonstrate that the snapping feedback indeed helps users to specify their intended queries much faster and also prevents users from missing the intended query. The gains are particularly impressive for Leap Motion, which is prone to the jitter and oversensitivity issues that are pervasive in motion capture devices. Further, we observe that users pay

attention to the snapping positions, validating our intent of providing guidance about the data.

There are several avenues of future work that were beyond the scope of this paper. One key area is the generalization of guidance beyond just predicate specification; considering projections, joins, and aggregations. Since the resulting representation and visualization for different joins or aggregations will be very different, providing smooth and gradual feedback to guide the user through all possible options is nontrivial. Second, from an implementation perspective, it is useful to develop a more complete framework to allow users to easily define the snapping condition and feedback, possibly iterating and improving them over time. Creating an implementation framework makes it easy to incorporate different interaction devices and interfaces. This can be extended by constructing a *SnapToQuery* language, and laying out algorithmic hooks for feedback and snapping condition definitions. Another area of further study is in the specific area of motion capture-based interfaces. As seen in the experiments, our Leap Motion interface suffers from issues such as jitter and sensitivity. While query specification with Leap Motion is made comparable to mouse and touch-based interfaces using snapping, there is a lot of room for improvement. Developing other techniques to improve Leap Motion-style interfaces is an ideal follow-up work. Finally, as discussed in Section 5.6, some parameters, such as mapping ratio, have an important effect on the performance. While our experiments explore a large portion of the parameter space, the ability to tune for these parameters in an automated manner would be an ideal goal.

Acknowledgements: We acknowledge the generous support of NEC Labs America and the National Science Foundation under awards IIS-1422977 and CAREER IIS-1453582 towards this work.

7. REFERENCES

[1] BMW Demonstrates Future iDrive with Touchscreen, Gesture and Tablet Control. *CES 2015*.
[2] Gartner Says Worldwide Traditional PC, Tablet, Ultramobile and Mobile Phone Shipments to Grow 4.2 Percent in 2014. *Gartner*, 2014.
[3] A. Abouzied, J. Hellerstein, and A. Silberschatz. DataPlay: Interactive Tweaking and Example-driven Correction of Graphical Database Queries. *UIST*, 2012.
[4] S. Basu Roy, H. Wang, G. Das, U. Nambiar, and M. Mohania. Minimum-Effort Driven Dynamic Faceted Search in Structured Databases. *CIKM*, 2008.
[5] P. Baudisch, E. Cutrell, K. Hinckley, and A. Eversole. Snap-and-go: Helping Users Align Objects Without the Modality of Traditional Snapping. *SIGCHI*, 2005.
[6] E. A. Bier et al. Snap-dragging. *SIGGRAPH*, 1986.
[7] M. Bostock, V. Ogievetsky, and J. Heer. D^3 Data-Driven Documents. *TVCG*, 2011.
[8] U. Cetintemel, M. Cherniack, J. DeBrabant, et al. Query Steering for Interactive Data Exploration. *CIDR*, 2013.
[9] K. Chakrabarti, K. Porkaew, and S. Mehrotra. Efficient Query Refinement in Multimedia Databases. *ICDE*, 2000.
[10] G. Chatzopoulou et al. Query Recommendations for Interactive Database Exploration. *SSDBM*, 2009.
[11] Z. Chen and T. Li. Addressing Diverse User Preferences in SQL-Query-Result Navigation. *SIGMOD*, 2007.
[12] A. Cockburn and S. Brewster. Multimodal Feedback for the Acquisition of Small Targets. *Ergonomics*, 2005.
[13] M. Dörk et al. Accentuating Visualization Parameters to Guide Exploration. *SIGCHI*, 2013.
[14] S. M. Drucker, D. Fisher, R. Sadana, J. Herron, et al. TouchViz: A Case Study Comparing Two Interfaces for Data Analytics on Tablets. *SIGCHI*, 2013.
[15] S. Dudoit and J. Fridlyand. A Prediction-based Resampling Method for Estimating the Number of Clusters in a Dataset. *Genome Biology*, 2002.
[16] S. G. Eick and G. J. Wills. High Interaction Graphics. *European Journal of Operational Research*, 1995.
[17] J. Fernquist, G. Shoemaker, and K. S. Booth. Oh Snap–Helping Users Align Digital Objects on Touch Interfaces. *INTERACT*, 2011.
[18] M. C. Hao, U. Dayal, D. A. Keim, and T. Schreck. Multi-Resolution Techniques for Visual Exploration of Large Time-Series Data. *EuroVis*, 2007.
[19] HP Envy. Worlds First Notebook with Integrated Leap Motion. 2014.
[20] E. Jhonsa. Intel Launches Gesture Recognition Camera/Sensor for PCs. *Seeking Alpha*, 2014.
[21] A. Kashyap, V. Hristidis, and M. Petropoulos. Cost-Driven Exploration of Faceted Query Results. *CIKM*, 2010.
[22] M. Kaul et al. Building Accurate 3D Spatial Networks to Enable Intelligent Transportation Systems. *MDM*, 2013.
[23] E. Liarou and S. Idreos. dbTouch in Action Database Kernels for Touch-based Data Exploration. *ICDE*, 2014.
[24] R. L. Mandryk, M. E. Rodgers, and K. M. Inkpen. Sticky Widgets: Pseudo-haptic Widget Enhancements for Multi-Monitor Displays. *SIGCHI*, 2005.
[25] Merriam-Webster.com. Jitter.
[26] C. Mishra and N. Koudas. Interactive Query Refinement. In *EDBT*, 2009.
[27] A. Nandi and H. Jagadish. Guided interaction: Rethinking the Query-Result Paradigm. *VLDB*, 2011.
[28] A. Nandi, L. Jiang, and M. Mandel. Gestural Query Specification. *VLDB*, 2014.
[29] D. Newman, S. Hettich, C. Blake, and C. Merz. UCI Repository of Machine Learning Databases, 1998.
[30] K. O'hara, R. Harper, H. Mentis, A. Sellen, and A. Taylor. On the Naturalness of Touchless: Putting the Interaction back into NUI. *TOCHI*, 2013.
[31] D. Phillips and W. Stuerzlinger. Can Friction Improve Mouse-Based Text Selection. *TIC-STH*, 2009.
[32] M. Rakow and J. Rossi. CSS Scroll Snap Points Module Level 1. `http://dev.w3.org/csswg/css-snappoints/`, 2014.
[33] L. Rosenberg, J. Beamer, A. Braun, and D. Chang. Mouse Interface Device and Method for Providing Enhanced Cursor Control. *U.S. Patent*, 2010.
[34] J. M. Rzeszotarski and A. Kittur. Kinetica: Naturalistic Multi-touch Data Visualization. *SIGCHI*, 2014.
[35] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*, page 672. 2006.
[36] R. Tibshirani, G. Walther, and T. Hastie. Estimating the Number of Clusters in a Data Set via the Gap Statistic. *J.R.Statist.Soc.B*, 2001.
[37] W. Willett, J. Heer, and M. Agrawala. Scented Widgets: Improving Navigation Cues with Embedded Visualizations. *TVCG*, 2007.
[38] J. Yang, M. O. Ward, and E. A. Rundensteiner. InterRing: An Interactive Tool for Visually Navigating and Manipulating Hierarchical Structures. *INFOVIS*, 2002.
[39] T. G. Zimmerman et al. A Hand Gesture Interface Device. *SIGCHI Bulletin*, 1987.